The Neo4j Operations Manual

Table of Contents

This is the Neo4j Operations Manual, which includes all the operational details and instructions for installing and deploying Neo4j in a self-hosted environment or in the cloud.

The manual covers the series of Neo4j 5, including the long-term support release **5.26**.

Note:

For all information on **upgrading and migrating Neo4j**, see the Neo4j Upgrade and Migration Guide.

Neo4j AuraDB is a fully managed Neo4j database, hosted in the cloud and requires no installation. For more information, see the AuraDB product page and AuraDB documentation.

Documentation updates for Neo4j 5.x

- Restructured chapter on clustering.

  The new clustering infrastructure decouples servers from databases, improving scalability and cloud-readiness. As such, the documentation has been restructured and rewritten. For a detailed description and instructions on how to set up clustering, see Clustering.

- Composite databases.

  Fabric technology is used to improve the setting and management of sharded and federated databases with dynamic compositions of databases. A new surface to administer Fabric databases, named Composite databases, is new in 5.0. Configuration settings have been removed and Composite databases are now administered via Cypher commands. For more information, see the new chapter on Composite databases, which replaces the *Fabric* chapter from previous versions.

- `neo4j-admin` refresh.

  All admin functionalities have been consolidated into a single tool, and all commands have been grouped by scope. There is an optional neo4j-admin conf file and individual conf files for each command so that there is more control over values provided for each command. Improved features, more control, and a consistent set of arguments for the database administrator.
  See Neo4j Admin and Neo4j CLI for details.

- A major overhaul of backup and restore.

  The new backup subsystem provides:

  - Full and differential backup to an immutable file, and aggregation to compile a chain of backups into a full backup.
  - Differential backup reduces storage requirements and provides point-in-time restore on timestamps or transaction IDs.
  - A new backup API for backup management and operability and target multiple URIs in a single backup command to support clusters in Neo4j 5.x.

- Incremental offline import.

  The `neo4j-admin database import` command can now add more data to existing databases.
  Import chapter has been updated. You can find more details there.

- Log4j integration completion.

  The logging framework is fully integrated with Log4j, providing more functionality and better control of all the database logs.
  Configuration settings are located in the `<NEO4J_HOME>/conf` folder in the *user-logs.xml* used for *neo4j.log* and *server-logs.xml* used for *debug.log*, *security.log*, *http.log*, and *query.log*. Query log uses the same format as *neo4j.log*.
  Users are able to use/modify Log4j XML files.
  See the section on the logging mechanisms in Neo4j for more details.

- Updates in the **Cypher Shell** section.
  - Cypher Shell supports impersonation.

    Cypher Shell users can impersonate other users via `--impersonate` or the command `:impersonate` (requires `IMPERSONATE` privileges).
    Visit Cypher Shell page for more details.

  - Cypher Shell logging.

    New option `--log` is introduced to enable Java Driver logging to the specified file. When users report problems with Java Driver/Bolt, now they can use Cypher Shell to try and replicate the issue without having to edit the client app that uses the driver.
    You can find more information in the Cypher Shell section.

- Immutable privileges.

  Immutable privileges are useful for restricting the actions of users who themselves are able to administer privileges. Starting with Neo4j 5.26, immutable roles are also available.
  Cloud operators can use sidecar design pattern to control RBAC-based permissions.
  For more information, see Immutable roles and privileges.

- Changes to Neo4j indexes
  - The B-tree index type has been removed.
  - New Range and Point index types are available now.
  - Neo4j 5.1 introduces an improved index provider, `text-2.0`, for enhanced performance. New Text indexes will use the `text-2.0` provider by default.
  - Full-text indexes can now index lists of strings.

    See Index configuration for more details.

License

may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

NonCommercial

You may not use the material for commercial purposes.

ShareAlike

If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

No additional restrictions

You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

Notices

You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation.

No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material.

See https://creativecommons.org/licenses/by-nc-sa/4.0/ for further details. The full license text is available at https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode.

Documentation license: Creative Commons 4.0

Introduction

Neo4j is the world's leading graph database. The architecture is designed for optimal management, storage, and traversal of nodes and relationships. The graph database takes a property graph approach, which is beneficial for both traversal performance and operations runtime. Neo4j offers dedicated memory management and memory-efficient operations.

Neo4j is scalable and can be deployed as a standalone server or across multiple machines in a fault-tolerant cluster for production environments. Other features for production applications include hot backups and extensive monitoring.

Neo4j editions

There are two editions of self-managed Neo4j to choose from, the Community Edition (CE) and the Enterprise Edition (EE). The Enterprise Edition includes all that Community Edition offers, plus extra enterprise requirements such as backups, clustering, and failover capabilities.

Community Edition

The Community Edition is a fully functional edition of Neo4j, suitable for single-instance deployments. It fully supports key Neo4j features, such as ACID-compliant transactions, Cypher, and programming APIs. It is ideal for learning Neo4j, do-it-yourself projects, and applications in small workgroups.

Enterprise Edition

The Enterprise Edition extends the functionality of Community Edition to include key features for performance and scalability, such as a clustering architecture and online backup functionality. Additional security features include role-based access control and LDAP support, for example, Active Directory. It is the choice for production systems with requirements for scale and availability, such as commercial and critical internal solutions.

The following table compares the available key features in both editions:

Table 1. Community Edition vs Enterprise Edition key features

Feature	Community Edition	Enterprise Edition
Open source under GPLv3	✔	
Native Graph		
Property graph model	✔	✔
Native graph processing & storage	✔	✔
Standard and Aligned store format (34 Billion Nodes & Relationships) Standard is deprecated in 5.23	✔	✔
High_limit (1 Quadrillion Nodes & Relationships) Deprecated in 5.23		✔
Block format GA from 5.16		✔
Change Data Capture (CDC) Introduced in 5.13 GA from 5.23		✔

Feature	Community Edition	Enterprise Edition
ACID-compliant transactions	✔	✔
Cypher graph query language	✔	✔
Slotted Cypher runtime	✔	✔
Pipelined Cypher runtime		✔
Parallel Cypher runtime Introduced in 5.13		✔
Listing and terminating running queries	✔	✔
High-performance caching	✔	✔
Cost-based query optimizer	✔	✔
Clients and APIs		
Cypher Shell	✔	✔
Neo4j Browser with syntax highlighting	✔	✔
Bolt Protocol	✔	✔
Language drivers for .NET, Go, Java, JavaScript, and Python [1]	✔	✔
High-performance native API	✔	✔
APOC 450+ Core Procedures and Functions	✔	✔
Support for Neo4j Graph Data Science Community Edition [1]	✔	✔
Support for Neo4j Graph Data Science Enterprise Edition [1]		✔
Support for Neo4j Bloom	✔	✔
Indexes and constraints		
Fast writes via native label indexes	✔	✔
Composite indexes	✔	✔
Full-text node & relationship indexes	✔	✔
Vector indexes Introduced in Neo4j 5.13	✔	✔
Property uniqueness constraints	✔	✔
Property existence constraints		✔
Property type constraints		✔
Node and relationship key constraints		✔
Security		

Feature	Community Edition	Enterprise Edition
Role-based access control		✔
Sub-graph access control		✔
LDAP and Active Directory integration		✔
Kerberos security option		✔
Data management		
Offline import	✔	✔
Offline incremental import		✔
Auto-reuse of space	✔	✔
Store copy		✔
Offline backup (dump)	✔	✔
Scale and availability		
Online backup and restore		✔
Multiple databases (beyond the `system` and default databases)		✔
Autonomous clustering		✔
Composite databases		✔
Monitoring and management		
Endpoints and metrics for monitoring via Prometheus		✔
Neo4j Operations Manager		✔

Note:

By default, Neo4j Community Edition and Neo4j Enterprise Edition report a small amount of usage data. This helps Neo4j understand how its products are used and improve them. For more information about what data is collected, see Usage data report.

Versioning

Neo4j uses semantic versioning (Semantic Versioning Specification 2.0.0). Given a version number `MAJOR.MINOR.PATCH`, the increment is based on:

- `MAJOR` version - incompatible API changes towards the previous `MAJOR` version.
- `MINOR` version - functionality in a backward-compatible manner.
- `PATCH` release - backwards-compatible bug fixes.

Neo4j's fully managed cloud service Neo4j Aura uses only `MAJOR` versioning.

[1] Must be downloaded and installed separately.

Installation

Neo4j can be installed in different deployment contexts, such as Linux, macOS, and Windows.

The following topics are covered:

- System requirements — The system requirements for a production deployment of Neo4j.
- Linux — Installation instructions for Linux.
- macOS — Installation instructions for macOS.
- Windows — Installation instructions for Windows.
- Neo4j Desktop — About Neo4j Desktop.

Installation-free options

Tip:

Neo4j AuraDB is a fully managed Neo4j database, hosted in the cloud and requires no installation. For more information, see the AuraDB product page and AuraDB documentation.

Neo4j can be run in a **Docker** container. For information on running Neo4j on Docker, see Docker.

Note:

By default, Neo4j Community Edition and Neo4j Enterprise Edition report a small amount of usage data. This helps Neo4j understand how its products are used and improve them. For more information about what data is collected, see Usage data report.

System requirements

Neo4j can be installed in many environments and for different scopes, therefore system requirements largely depend on the use of the software. This section distinguishes between a personal/development installation and a server-based installation used for production workloads.

Tip:

Neo4j AuraDB is a fully managed Neo4j database, hosted in the cloud and requires no installation. For more information, see the AuraDB product page and AuraDB documentation.

Supported platforms

Neo4j is supported on systems with x86_64 and ARM architectures on physical, virtual, or containerized platforms.

Hardware requirements

In terms of minimum hardware requirements, follow these guidelines:

Table 2. Hardware requirement guidelines

CPU	Performance is generally memory or I/O bound for large graphs, and compute-bound for graphs that fit in memory.
Memory	More memory allows for larger graphs, but it needs to be configured properly to avoid disruptive garbage collection operations.
Storage	Aside from capacity, the performance characteristics of the disk are the most important when selecting storage: • Neo4j workloads tend significantly toward random reads. • Select media with low average seek time: SSD over spinning disks.

For personal use and software development:

Table 3. Hardware requirement guidelines for personal use and software development

CPU	Intel x86-x64 Core i3 minimum, Core i7 recommended. AMD x86-x64, Mac ARM.
Memory	2GB minimum, 16GB or more recommended.
Storage	10GB SATA Minimum, SSD with SATA Express or NVMe recommended.

For cloud environments:

Table 4. Hardware requirement guidelines for cloud environments

CPU	2vCPU minimum, 16+ recommended.
Memory	2GB minimum. Actual requirements depend on workloads. In some cases, it is recommended to use instances with memory that fits the size of the graph in use.
Storage	10GB minimum block storage, attached NVMe SSD recommended. Storage size depends on the size of the databases.

For server-based, on-premises environments:

Table 5. Hardware requirement guidelines for server-based, on-premises environments

CPU	Intel/AMD x86-x64. ARM64.
Memory	8GB minimum. Actual requirements depend on workloads. In some cases, it is recommended to use instances with memory that fits the size of the graph in use.
Storage	RAID/SAN or SSD with greater than 5000 IOPS. NVMe SSD is recommended. Storage size depends on the size of the databases.

Tip:

For more information, see Performance, and more specifically Memory Configuration and

Tuning of the garbage collector, and Disks, RAM and other tips.

Software requirements

For personal use and software development:

Table 6. Software requirements for personal use and software development

Operating System	Supported JDK
Debian 11, 12	OpenJDK 17/21, OracleJDK 17/21, and ZuluJDK 17/21
macOS 11, 12, 13, 14, 15 [1]	OpenJDK 17, ZuluJDK 17/21
SuSE Enterprise Desktop 15	OpenJDK 17/21, OracleJDK 17/21
Ubuntu Desktop 22.04, 24.04	OpenJDK 17/21, OracleJDK 17/21, and ZuluJDK 17/21
Windows 10, 11 [1]	OracleJDK 17/21, ZuluJDK 17/21

For cloud environments, and server-based, on-premises environments:

Table 7. Software requirements for cloud environments, and server-based, on-premises environments

Operating System	Supported JDK
Amazon Linux 2022 AMI, Amazon Linux 2023 AMI [2]	Amazon Corretto 17/21, and OracleJDK 17/21
CentOS Stream 8, 9, 10	OpenJDK 17/21, OracleJDK 17/21, and ZuluJDK 17/21
Debian 11, 12	OpenJDK 17/21, OracleJDK 17/21, and ZuluJDK 17/21
Red Hat Enterprise Linux Server 8.x, 9.x, 10.x	Red Hat OpenJDK 17/21, Oracle JDK 17/21, and ZuluJDK 17/21
Ubuntu Server 16.04, 18.04, 20.04, 22.04, 24.04 [2]	OpenJDK 17/21, OracleJDK 17/21, and ZuluJDK 17/21
Windows Server 2016, 2019, 2022, 2025 [2]	OracleJDK 17/21, ZuluJDK 17/21

For more information on Red Hat Enterprise Linux Life Cycle, refer to their official documentation.

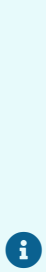
Neo4j 5.26.25 is built and tested against RHEL 8.10, 9.6, and 10.0, and assumes compatibility across other minor releases within the RHEL 8.x, 9.x, and 10.x major versions.

Filesystem

For proper ACID behavior, the filesystem must support flush operations such as *fsync* and *fdatasync*. Since databases can place a high and consistent load on a storage system for long periods, it is recommended to use a filesystem with strong aging characteristics.

On Linux systems, a POSIX-compliant filesystem is required. Network shares such as NFS do not meet this POSIX requirement. The only supported filesystems on Linux are EXT4 and XFS.

See Linux file system tuning for details on how to configure the filesystem in Linux for optimal performance.



Note:

If *tmp* is set to `noexec`, it is recommended to set `server.jvm.additional=-Djava.io.tmpdir=/home/neo4j` in *conf/neo4j.conf* and replace */home/neo4j* with a path that has `exec` permissions.

For */bin/cypher-shell*, set this via an environment variable: `export JAVA_OPTS=-Djava.io.tmpdir=/home/neo4j` and replace `/home/neo4j` with a path that has `exec` permissions.

For the Neo4j's uses of the Java Native Access (JNA) library, set `server.jvm.additional=-Djna.tmpdir=/tmp` in *conf/neo4j.conf* and replace `/tmp` with a path that has `exec` permissions.

Java

It is required to have a pre-installed, compatible Java Virtual Machine (JVM) to run a Neo4j instance. The minimum requirement is Java Runtime Environment (JRE).

Table 8. Neo4j version and JVM requirements

Neo4j Version	JVM compliancy
3.x	Java SE 8 Platform Specification
4.x	Java SE 11 Platform Specification
5.x	Java SE 17 Platform Specification
5.14	Java SE 17 and Java SE 21 Platform Specification
5.26 LTS	Java SE 17 and Java SE 21 Platform Specification

Note:

The Neo4j 5.26 LTS is the last version that uses the Java SE 17 Platform. It is recommended to move to **Java SE 21**, which is supported in Neo4j 5.14 onwards.

Neo4j Desktop is available for developers and personal users. Neo4j Desktop is bundled with a JVM. For more information on how to use Neo4j Desktop and its capabilities, see the Neo4j Desktop documentation.

Linux installation

You can install Neo4j on Linux using Debian or RPM packages, or from a TAR archive.

This section describes how to:

- Install Neo4j on Debian and Debian-based distributions (.deb) using the Neo4j Debian package.
- Install Neo4j on Red Hat, CentOS, Fedora, and Amazon Linux (.rpm) using the Neo4j RPM package.
- Install Neo4j on Linux from a tarball and run it as a console application or a service.
- Configure and operate the Neo4j system service.

Debian-based distributions (.deb)

You can install Neo4j on Debian, and Debian-based distributions like Ubuntu, using the Neo4j Debian package.

Java prerequisites

Neo4j 5 requires the Java 17 runtime. Java 17 is not included in Ubuntu 16.04 LTS and will have to be set up manually before installing or upgrading to Neo4j 5, as described below. Debian 11 and Ubuntu 18.04 onwards already have the OpenJDK Java 17 package available through `apt`. From version 5.14 onwards, Neo4j also supports JDK 21.

Oracle JDK, Zulu JDK, or Corretto JDK

If you wish to use a non-default JDK, it must be installed before starting the Neo4j installation. Otherwise, your package manager will install the default Java distribution for your operating system, usually OpenJDK.

Download and installation instructions can be found on the manufacturer's website:

- Oracle JDK
- Zulu JDK
- Amazon Corretto JDK

OpenJDK 17 on Ubuntu 16.04

Add the official OpenJDK package repository to `apt`:

```
sudo add-apt-repository -y ppa:openjdk-r/ppa
sudo apt-get update
```

You are now ready to install Neo4j, which will install Java 17 automatically if it is not already installed. See Dealing with multiple installed Java versions to make sure you can start Neo4j after installation.

Dealing with multiple installed Java versions

You must configure your default Java version to point to Java 17, or Neo4j 5.26.25 will be unable to start. Do so with the `update-java-alternatives` command.

1. List all your installed versions of Java with `update-java-alternatives --list`.

   Your results may vary, but this is an example of the output:

```
java-1.17.0-openjdk-amd64 1711 /usr/lib/jvm/java-1.17.0-openjdk-amd64
java-1.11.0-openjdk-amd64 1071 /usr/lib/jvm/java-1.11.0-openjdk-amd64
```

2. Identify your Java 17 version:

```
java -version
```

   In this case, it is `java-1.17.0-openjdk-amd64`.

3. Set it as the default by replacing `<java17name>` with its name:

```
sudo update-java-alternatives --jre --set <java17name>
```

4. Confirm which version of Java is the default using `java -version` again.

Installation

To install Neo4j using the Debian package manager `apt`, you need to add the Neo4j repository to your system's list of package sources, and then install the desired Neo4j package. The Debian package is available from https://debian.neo4j.com.

Add the repository

Run the following commands as a sudo user to add the Neo4j repository to the package manager:

1. Create the keyrings directory for the Neo4j GPG key if it does not already exist:

```
sudo mkdir -p /etc/apt/keyrings
```

2. Download and install the Neo4j GPG key:

```
wget -O - https://debian.neo4j.com/neotechnology.gpg.key | sudo gpg --dearmor -o
/etc/apt/keyrings/neotechnology.gpg > /dev/null
```

3. Ensure the key file is world-readable:

```
sudo chmod a+r /etc/apt/keyrings/neotechnology.gpg
```

4. Add the Neo4j APT repository:

```
echo 'deb [signed-by=/etc/apt/keyrings/neotechnology.gpg] https://debian.neo4j.com stable 5.26' | sudo
tee -a /etc/apt/sources.list.d/neo4j.list > /dev/null
```

5. Update package lists:

```
sudo apt-get update
```

6. Once the repository has been added to `apt`, you can verify which Neo4j versions are available by running:

```
apt list -a neo4j
```

> **Note:**
>
> In Ubuntu server installations, you also need to make sure that the `universe` repository is enabled. If the `universe` repository is not present, the Neo4j installation will fail with the error `Depends: daemon but it is not installable`.
>
> This can be fixed by running the command:
>
> ```
> sudo add-apt-repository universe
> ```

Install Neo4j

To install Neo4j, run one of the following commands depending on which version you want to install:

- Neo4j Community Edition:

```
sudo apt-get install neo4j=1:5.26.25
```

- Neo4j Enterprise Edition:

```
sudo apt-get install neo4j-enterprise=1:5.26.25
```

Note that the version includes an epoch version component (`1:`), following the Debian policy on versioning.

When installing Neo4j Enterprise Edition, you will be prompted to accept the license agreement. Once the license agreement is accepted installation begins. Your answer to the license agreement prompt will be remembered for future installations on the same system.

To forget the stored answer, and trigger the license agreement prompt on subsequent installation, use `debconf-communicate` to purge the stored answer:

```
echo purge | sudo debconf-communicate neo4j-enterprise
```

Non-interactive installation of Neo4j Enterprise Edition

For Neo4j Enterprise Edition, the license agreement is presented in an interactive prompt. If you require non-interactive installation of Neo4j Enterprise Edition, you can indicate that you have read and accepted the license agreement using `debconf-set-selections`:

For Neo4j 5.3 and later

```
echo "neo4j-enterprise neo4j/accept-license select Accept commercial license" | sudo debconf-set-
```

```
selections
```

For Neo4j 5.2 and earlier

```
echo "neo4j-enterprise neo4j/question select I ACCEPT" | sudo debconf-set-selections
echo "neo4j-enterprise neo4j/license note" | sudo debconf-set-selections
```

Verify the Java version

On newer Debian or Ubuntu operating systems, Java 19 is available by default, and `apt` may install OpenJDK 19, **even if you have Java 17 installed**.

If this happens, Neo4j will return the following warning on start:

```
WARNING! You are using an unsupported Java runtime.
* Please use Oracle(R) Java(TM) 17, OpenJDK(TM) 17 to run Neo4j.
* Please see https://neo4j.com/docs/ for Neo4j installation instructions.
```

To fix this, you can install Java 17 manually and then either uninstall OpenJDK 19 or set Java 17 as the default.

1. Run the following command to install OpenJDK 17 manually:

   ```
   sudo apt install openjdk-17-jre
   ```

   For other distributions of Java 17, see Java prerequisites.

2. Choose one of the following options to set it as the default:
   - Uninstall OpenJDK 19:

     ```
     sudo apt remove openjdk-19-jre-headless
     ```

   - If you want to keep OpenJDK 19 installed, set Java 17 as default by following the instructions in Dealing with multiple installed Java versions.

Offline installation

If you cannot reach `https://debian.neo4j.com`, perhaps due to a firewall, you need to obtain Neo4j via an alternative machine that has the relevant access, and then move the package manually.

Note:

It is important to note that using this method will mean that the offline machine will not receive the dependencies that are normally downloaded and installed automatically when using `apt` for installing Neo4j; Cypher Shell and Java (if not installed already):

- The Cypher Shell package can be downloaded from Neo4j Deployment Center.
- For information on supported versions of Java, see System requirements.

1. Run the following to download the required Debian software package:
   - Neo4j Enterprise Edition:

```
curl -O https://dist.neo4j.org/deb/neo4j-enterprise_5.26.25_all.deb
```

> **Note:**
>
> To list all files that the Debian software package (`.deb` file) installs:
>
> ```
> dpkg --contents neo4j_5.26.25_all.deb
> ```

   - Neo4j Community Edition:

```
curl -O https://dist.neo4j.org/deb/neo4j_5.26.25_all.deb
```

2. Manually move the downloaded Debian package to the offline machine.
3. Run the following on the offline machine to install Neo4j:

```
sudo dpkg -i <deb file name>
```

File locations

File locations for all Neo4j packages are documented here.

Operation

Most Neo4j configuration goes into *neo4j.conf*.

For operating systems using `systemd`, some package-specific options are set in *neo4j.service* and can be edited using `systemctl edit neo4j.service`.

For operating systems that are not using `systemd`, some package-specific options are set in */etc/default/neo4j*.

Environment variable	Default value	Details
NEO4J_SHUTDOWN_TIMEOUT	*120*	Timeout in seconds when waiting for Neo4j to stop. If it takes longer than this then the shutdown is considered to have failed. This may need to be increased if the system serves long-running transactions.
NEO4J_ULIMIT_NOFILE	*60000*	Maximum number of file handles that can be opened by the Neo4j process.

Starting the service automatically on system start

On Debian-based distributions, run the following command to ensure that Neo4j starts automatically at boot time:

```
sudo systemctl enable neo4j
```

> **Note:**
>
> Before starting up the database for the first time, it is recommended to use the `set-initial-password` command of `neo4j-admin` to define the password for the native user `neo4j`.
>
> If the password is not set explicitly using this method, it will be set to the default password `neo4j`. In that case, you will be prompted to change the default password at first login.
>
> For more information, see Set an initial password.

For more information on operating the Neo4j system service, see Neo4j system service.

Uninstall Neo4j

Follow these steps to uninstall Neo4j:

1. (Optional) Create a backup to avoid losing your data.
2. Uninstall Neo4j:

```
---
sudo apt remove neo4j
---
```

Red Hat, CentOS, Fedora, and Amazon Linux (.rpm)

You can deploy Neo4j on Red Hat, CentOS, Fedora, or Amazon Linux distributions using the Neo4j RPM package.

Java prerequisites

Neo4j 5.x runs on Java 17, and from Neo4j 5.14, it also supports Java 21.

OpenJDK Java 17

Most of our supported Linux distributions have OpenJDK Java 17 available by default. Consequently, no extra setup is required if you are using OpenJDK Java, the correct Java dependency will be installed by the package manager when installing Neo4j.

Oracle Java 17

There is some minor setup required for compatibility with Oracle Java 17 because Oracle and OpenJDK provide incompatible RPM packages for Java 17.

You can use an adapter for Oracle Java 17, which must be installed before Neo4j. The adapter contains no code but stops the package manager from installing OpenJDK 17 as a dependency despite an existing Oracle Java 17 installation.

1. Download and install the Oracle Java 17 JDK from the Oracle website.
2. Install the adapter:

   ```
   sudo yum install https://dist.neo4j.org/neo4j-java17-adapter.noarch.rpm
   ```

   The SHA-256 of the adapter package can be verified against https://dist.neo4j.org/neo4j-java17-adapter.noarch.rpm.sha256.

Zulu JDK 17 or Corretto 17

If you want to use a non-default JDK, it must be installed before starting the Neo4j installation. Otherwise, your package manager will install the default Java distribution for your operating system, usually OpenJDK.

Installation instructions can be found on the manufacturer's website:

- Zulu JDK
- Amazon Corretto JDK

Install on Red Hat, CentOS or Amazon Linux

Set up the repository

1. Import the Neo4j GPG public key into the system's RPM keyring. The key is required to verify the authenticity of the Neo4j packages you will install.

   ```
   rpm --import https://debian.neo4j.com/neotechnology.gpg.key
   ```

2. Create a `neo4j.repo` file in the `/etc/yum.repos.d/` directory. This file contains the repository configuration for Neo4j.

   ```
   cat <<EOF > /etc/yum.repos.d/neo4j.repo
   [neo4j]
   name=Neo4j RPM Repository
   baseurl=https://yum.neo4j.com/stable/5
   enabled=1
   gpgcheck=1
   EOF
   ```

   Note:

   If you are upgrading from Neo4j 4.4 or earlier, you may need to clear the package manager cache before Neo4j packages become available:

```
yum clean dbcache
```

3. Verify that the Neo4j repository is set up correctly by listing the available Neo4j packages versions:

```
yum list neo4j --showduplicates
```

Install Neo4j

Install Neo4j as `root` using the following commands depending on which edition you are using:

- Community Edition

```
yum install neo4j-5.26.25
```

- Enterprise Edition

  From Neo4j 5.4 onwards, you are required to accept either the commercial or the evaluation license agreement before running the Neo4j Enterprise Edition. The following are examples of using an interactive prompt and a non-interactive installation:

  Interactive installation of Enterprise Edition under the commercial license

```
yum install neo4j-enterprise-5.26.25
```

  You have to choose either a commercial license or an evaluation license before the interactive installation is allowed to complete.

  For a non-interactive installation, you can set the environment variable `NEO4J_ACCEPT_LICENSE_AGREEMENT` to `yes` (for the commercial license) or `eval` (for the evaluation license). This should be done in the same line as the package is installed, to ensure bash correctly passes the environment variable to the installer process. As in the following example:

  Non-interactive installation of Enterprise Edition under the commercial license

```
NEO4J_ACCEPT_LICENSE_AGREEMENT=yes yum install neo4j-enterprise-5.26.25
```

Install on SUSE

For SUSE-based distributions, the steps are as follows:

1. Use the following as `root` to add the repository:

```
zypper addrepo --refresh https://yum.neo4j.com/stable/5 neo4j-repository
```

2. Install Neo4j as `root` using the following commands depending on which edition you are using:
   - Community Edition

```
zypper install neo4j-5.26.25
```

- Enterprise Edition

  From Neo4j 5.4 onwards, you are required to accept either the commercial or the evaluation license agreement before running the Neo4j Enterprise Edition. The following are examples of using an interactive prompt and a non-interactive installation:

  Interactive installation of Enterprise Edition under the commercial license

  ```
  zypper install neo4j-enterprise-5.26.25
  ```

  You have to choose either a commercial license or an evaluation license before the interactive installation is allowed to complete.

  For a non-interactive installation, you can set the `NEO4J_ACCEPT_LICENSE_AGREEMENT` to `yes` (for the commercial license) or `eval` (for the evaluation license) as in the following example:

  Non-interactive installation of Enterprise Edition under the commercial license

  ```
  NEO4J_ACCEPT_LICENSE_AGREEMENT=yes zypper install neo4j-enterprise-5.26.25
  ```

Offline installation

If you cannot reach `https://yum.neo4j.com/stable/5` to install Neo4j using RPM, perhaps due to a firewall, you need to obtain Neo4j via an alternative machine that has the relevant access, and then move the RPM package manually.

> **Note:**
>
> It is important to note that using this method means that the offline machine cannot receive the dependencies that are normally downloaded and installed automatically when using `yum` for installing Neo4j, Neo4j Cypher Shell, and Java.

1. Download the Neo4j and Cypher Shell RPM installers from Deployment Center or run the following to obtain the required packages:
   - Cypher Shell:

     ```
     curl -O https://dist.neo4j.org/cypher-shell/cypher-shell-5.26.25-1.noarch.rpm
     ```

   - Neo4j Community Edition:

     ```
     curl -O https://dist.neo4j.org/rpm/neo4j-5.26.25-1.noarch.rpm
     ```

   - Neo4j Enterprise Edition:

     ```
     curl -O https://dist.neo4j.org/rpm/neo4j-enterprise-5.26.25-1.noarch.rpm
     ```

2. Manually move the downloaded RPM packages to the offline machine. Before installing Neo4j, you

must manually install the required Java 17 packages.

> **Note:**
>
> If using Oracle Java 17, the same dependency issues apply as with the Oracle Java prerequisites. You will need to additionally download and install the Java adaptor described in that section.

3. Install Neo4j and Cypher Shell as `root` using the following command depending on which edition you are using:

> **Note:**
>
> If you are upgrading from Neo4j 4.4 or earlier versions of 5.x, due to strict dependencies between Neo4j and Cypher Shell, both packages must be upgraded simultaneously. This must be one single command, and Neo4j Cypher Shell must be the first package in the command. For later versions, you can install them separately but still need to install Cypher Shell first.

   - Community Edition

     ```
     rpm --install cypher-shell-5.26.25-1.noarch.rpm neo4j-5.26.25-1.noarch.rpm
     ```

   - Enterprise Edition

     From Neo4j 5.4 onwards, you are required to accept either the commercial or the evaluation license agreement before running the Neo4j Enterprise Edition. The following example uses an interactive prompt:

     ```
     rpm --install cypher-shell-5.26.25-1.noarch.rpm neo4j-enterprise-5.26.25-1.noarch.rpm
     ```

     You have to choose either a commercial license or an evaluation license before the interactive installation is allowed to complete. For a non-interactive installation, you can set the `NEO4J_ACCEPT_LICENSE_AGREEMENT` to `yes` (for the commercial license) or `eval` (for the evaluation license) as in the following example:

     ```
     NEO4J_ACCEPT_LICENSE_AGREEMENT=yes rpm --install cypher-shell-5.26.25-1.noarch.rpm neo4j-
     enterprise-5.26.25-1.noarch.rpm
     ```

Start the Neo4j service automatically on system start

To enable Neo4j to start automatically on system boot, run the following command:

```
systemctl enable neo4j
```

> **Note:**

Before starting up the database for the first time, it is recommended to use the `set-initial-password` command of `neo4j-admin` to define the password for the native user `neo4j`.

If the password is not set explicitly using this method, it will be set to the default password `neo4j`. In that case, you will be prompted to change the default password at first login.

For more information, see Set an initial password.

For more information on operating the Neo4j system service, see Neo4j system service.

Uninstall Neo4j

Follow these steps to uninstall Neo4j:

1. (Optional) Create a backup to avoid losing your data.
2. Uninstall Neo4j:

```
sudo yum remove neo4j
```

Linux executable (.tar)

Before you install Neo4j on Linux from a tarball and run it as a console application or a service, check System Requirements to see if your setup is suitable.

Install Neo4j from a tarball

1. If it is not already installed, get OpenJDK 17 or Oracle Java 17. From version 5.14 onwards, Neo4j also supports JDK 21.
2. Download the latest Neo4j tarball from Neo4j Deployment Center and unpack it:

```
tar zxf neo4j-enterprise-5.26.25-unix.tar.gz
```

3. Move the extracted files to your server's /opt directory and create a symlink to it:

```
mv neo4j-enterprise-5.26.25 /opt/
ln -s /opt/neo4j-enterprise-5.26.25 /opt/neo4j
```

4. Create a `neo4j` user and group:

```
groupadd neo4j
useradd -g neo4j neo4j -s /bin/bash
```

5. Give the directory the correct ownership using one of the options:
   - **Ubuntu**

```
chown -R neo4j:adm /opt/neo4j-enterprise-5.26.25
```

- RedHat

```
chown -R neo4j /opt/neo4j-enterprise-5.26.25
```

6. Change the default locations of the *data*, *conf*, *certificates*, *licenses*, and *plugins* (if you plan to use custom plugins) directories by setting the environment variable `NEO4J_CONF` and the respective `server.directories.*` settings to point to the desired locations.

> **Note:**
>
> Storing your Neo4j files outside `NEO4J_HOME` will simplify the upgrade process later because you will be able to replace the DBMS binaries without affecting the configuration and state. Otherwise, these Neo4j files will remain in the old installation folder and may be accidentally overwritten during an upgrade or deleted during a subsequent uninstall.

7. Accept either the commercial or the evaluation license agreement before running the Neo4j Enterprise Edition. If you are using Community Edition, you can skip this step.
   - Use one of the following options to accept the commercial license agreement. See the Neo4j licensing page for details on the available agreements.
     - Set the environment variable `NEO4J_ACCEPT_LICENSE_AGREEMENT=yes`.
     - Run `<NEO4J_HOME>/bin/neo4j-admin server license --accept-commercial`
   - Use one of the following options to accept the Neo4j Evaluation Agreement for Neo4j Software.
     - Set the environment variable `NEO4J_ACCEPT_LICENSE_AGREEMENT=eval`.
     - Run `<NEO4J_HOME>/bin/neo4j-admin server license --accept-evaluation`.
8. Before starting up the database for the first time, it is recommended to use the `set-initial-password` command of `neo4j-admin` to define the password for the native user `neo4j`.

   If the password is not set explicitly using this method, it will be set to the default password `neo4j`. In that case, you will be prompted to change the default password at first login.
   For more information, see Set an initial password.

9. Start Neo4j:
   - To run Neo4j as a console application, use: `<NEO4J_HOME>/bin/neo4j console`.
   - To run Neo4j in a background process, use: `<NEO4J_HOME>/bin/neo4j start`.
10. Open *http://localhost:7474/* in your web browser.
11. Connect using the username `neo4j` with your password or the default password `neo4j`. You will then be prompted to change the password.
12. Stop the server by typing `Ctrl-C` in the console.

Configure Neo4j to start automatically on system boot

You can create a Neo4j service and configure it to start automatically on system boot.

1. Create the file */lib/systemd/system/neo4j.service* with the following contents:

```
[Unit]
Description=Neo4j Graph Database
After=network-online.target
Wants=network-online.target

[Service]
ExecStart=/opt/neo4j/bin/neo4j console
Restart=on-abnormal
User=neo4j
Group=neo4j
Environment="NEO4J_CONF=/opt/neo4j/conf" "NEO4J_HOME=/opt/neo4j"
LimitNOFILE=60000
TimeoutSec=120

[Install]
WantedBy=multi-user.target
```

2. Reload systemctl to pick up the new service file:

```
systemctl daemon-reload
```

3. Configure Neo4j to start at boot time:

```
systemctl enable neo4j
```

4. Before starting up the database for the first time, it is recommended to use the `set-initial-password` command of `neo4j-admin` to define the password for the native user `neo4j`.

   If the password is not set explicitly using this method, it will be set to the default password `neo4j`. In that case, you will be prompted to change the default password at first login.
   For more information, see Set an initial password.

5. Start Neo4j:

```
systemctl start neo4j
```

6. Check the status of the newly created service:

```
systemctl status neo4j
```

7. Reboot the system (if desired) to verify that Neo4j restarts on boot:

```
reboot
```

For more information on operating the Neo4j system service, see Neo4j system service.

Setting the number of open files

Linux platforms impose an upper limit on the number of concurrently open files per user and session. To check your limit for the current session, run the command `ulimit -n`. The default value is 1024.

```
ulimit -n
```

However, if you experience exceptions on `Too many open files` or `Could not stat() directory`, you have to increase the limit to 40000 or more, depending on your usage patterns. This is especially true when many indexes are used, or the server installation sees too many open network connections or sockets.

A quick solution is the command `ulimit -n <the-new-limit>`, but it will set a new limit only for the root user and will affect only the current session. If you want to set the value system-wide, follow the instructions for your platform.

The following steps set the open file descriptor limit to 60000 for the user *neo4j* under Ubuntu 16.04 LTS, Debian 8, CentOS 7, or later versions.

Running Neo4j as a service

1. Open the *neo4j.service* file with root privileges.

   ```
   sudo systemctl edit neo4j.service
   ```

2. Append the following to the `[Service]` section, created in Configure Neo4j to start automatically on system boot:

   ```
   [Service]
   ...
   LimitNOFILE=60000
   ```

Running Neo4j as an interactive user (e.g., for testing purposes)

1. Open the *user.conf* file with root privileges in a text editor. This example uses Vim:

   ```
   sudo vi /etc/systemd/user.conf
   ```

2. Uncomment and define the value of `DefaultLimitNOFILE`, found in the `[Manager]` section.

   ```
   [Manager]
   ...
   DefaultLimitNOFILE=60000
   ```

3. Open the */etc/security/limits.conf* file.

   ```
   sudo vi /etc/security/limits.conf
   ```

4. Define the following values:

```
neo4j   soft    nofile  60000
neo4j   hard    nofile  60000
```

5. Reload the `systemd` settings.

```
sudo systemctl daemon-reload
```

6. Reboot your machine.

Uninstall Neo4j

Follow these steps to uninstall Neo4j on Linux:

1. (Optional) Create a backup to avoid losing your data.
2. Stop all Neo4j running services:

```
---
sudo systemctl stop neo4j
sudo systemctl disable neo4j
---
```

3. Delete *NEO4J_HOME* and the file */lib/systemd/system/neo4j.service*:

```
---
rm /lib/systemd/system/neo4j.service
rm -rf NEO4J_HOME
---
```

Neo4j system service

This page covers configuring and operating the Neo4j system service. It assumes that your system has `systemd`, which is the case for most Linux distributions.

> **Note:**
>
> **Setting the number of open files.**
>
> For instructions on how to set the number of concurrent files that a user can have open, see Setting the number of open files.

Configuration

Configuration is stored in */etc/neo4j/neo4j.conf*. See Default file locations for a complete catalog of where files are found for the various packages.

Before starting up the database for the first time, it is recommended to use the `set-initial-password` command of `neo4j-admin` to define the password for the native user `neo4j`.

If the password is not set explicitly using this method, it will be set to the default password `neo4j`. In that

case, you will be prompted to change the default password at first login.

For more information, see Set an initial password.

Controlling the service

System services are controlled with the `systemctl` command. It accepts a number of commands:

```
systemctl {start|stop|restart} neo4j
```

Service customizations can be placed in a service override file. To edit your specific options, do the following command which will open up an editor of the appropriate file:

```
systemctl edit neo4j
```

Then place any customizations under a `[Service]` section. The following example lists default values that may be interesting to change for some users:

```
[Service]
# The user and group which the service runs as.
User=neo4j
Group=neo4j
# If it takes longer than this then the shutdown is considered to have failed.
# This may need to be increased if the system serves long-running transactions.
TimeoutSec=120
```

You can print the effective service, including possible overrides, with:

```
systemctl cat neo4j
```

Remember to restart neo4j if you change any settings.

```
systemctl restart neo4j
```

Log

The neo4j log is written to `journald` which can be viewed using the `journalctl` command:

```
journalctl -e -u neo4j
```

`journald` automatically rotates the log after a certain time and by default it commonly does not persist across reboots. Please see `man journald.conf` for further details.

Uninstall Neo4j

Follow these steps to uninstall Neo4j:

1. (Optional) Create a backup to avoid losing your data.

2. Uninstall Neo4j:

```
---
sudo apt remove neo4j
---
```

macOS installation

Before you install Neo4j on macOS, check System Requirements to see if your setup is suitable.

Unix console application

1. If it is not already installed, get OpenJDK 17 or ZuluJDK 17. From version 5.14 onwards, Neo4j also supports JDK 21.
2. Download the latest release from Neo4j Deployment Center.

   Select the appropriate tar.gz distribution for your platform.

3. Make sure to download Neo4j from Neo4j Deployment Center and always check that the SHA hash of the downloaded file is correct:
   a. To find the correct SHA hash, go to Neo4j Deployment Center and click on `SHA-256` which will be located below your downloaded file.
   b. Using the appropriate commands for your platform, display the `SHA-256` hash for the file that you downloaded.
   c. Ensure that the two are identical.
4. Extract the contents of the archive, using `tar -xf` <filename>. For example, `tar -xf neo4j-community-5.26.25-unix.tar.gz`.
5. Place the extracted files in a permanent home on your server and set the environment variable `NEO4J_HOME` to point to the extracted directory, for example, `export NEO4J_HOME=/path/to/_<NEO4J_HOME>` to make it easier to refer to it later.
6. Change the default locations of the *data*, *conf*, *certificates*, *licenses*, and *plugins* (if you plan to use custom plugins) directories by setting the environment variable `NEO4J_CONF` and the respective `server.directories.*` settings to point to the desired locations.

   > **Note:**
   >
   > Storing your Neo4j files outside `NEO4J_HOME` will simplify the upgrade process later because you will be able to replace the DBMS binaries without affecting the configuration and state. Otherwise, these Neo4j files will remain in the old installation folder and may be accidentally overwritten during an upgrade or deleted during a subsequent uninstall.

7. Accept either the commercial or the evaluation license agreement before running the Neo4j Enterprise Edition. If you are using Community Edition, you can skip this step.
   - Use one of the following options to accept the commercial license agreement. See the Neo4j

licensing page for details on the available agreements.

- Set the environment variable `NEO4J_ACCEPT_LICENSE_AGREEMENT=yes`.
- Run `$NEO4J_HOME/bin/neo4j-admin server license --accept-commercial`

◦ Use one of the following options to accept the Neo4j Evaluation Agreement for Neo4j Software:

- Set the environment variable `NEO4J_ACCEPT_LICENSE_AGREEMENT=eval`.
- Run `$NEO4J_HOME/bin/neo4j-admin server license --accept-evaluation`.

8. Before starting up the database for the first time, it is recommended to use the `set-initial-password` command of `neo4j-admin` to define the password for the native user `neo4j`.

   If the password is not set explicitly using this method, it will be set to the default password `neo4j`. In that case, you will be prompted to change the default password at first login.
   For more information, see Set an initial password.

9. Start Neo4j:
   ◦ To run Neo4j as a console application, use: `$NEO4J_HOME/bin/neo4j console`.
   ◦ To run Neo4j in a background process, use: `$NEO4J_HOME/bin/neo4j start`.
10. Open *http://localhost:7474/* in your web browser.
11. Connect using the username `neo4j` with your password or the default password `neo4j`. You will then be prompted to change the password.
12. Stop the server by typing `Ctrl-C` in the console.

When Neo4j runs in console mode, logs are printed to the terminal.

macOS service

Use the standard macOS system tools to create a service based on the `neo4j` command.

macOS file descriptor limits

The limit of *open file descriptors* may have to be increased if a database has many indexes or if there are many connections to the database. The currently configured open file descriptor limitation on your macOS system can be inspected with the `launchctl limit maxfiles` command. The method for changing the limit may differ depending on the version of macOS. Consult the documentation for your operating system in order to find out the appropriate command.

If you raise the limit above 10240, then you must also add the following setting to your *neo4j.conf* file:

```
server.jvm.additional=-XX:-MaxFDLimit
```

Without this setting, the file descriptor limit for the JVM will not be increased beyond 10240. Note, however, that this only applies to macOS. On all other operating systems, you should always leave the `MaxFDLimit` JVM setting enabled.

Uninstall Neo4j

Here are the steps to uninstall Neo4j on macOS:

1. (Optional) Create a backup to avoid losing your data.
2. Stop all Neo4j running services:

```
---
sudo systemctl stop neo4j
sudo systemctl disable neo4j
---
```

3. Delete *NEO4J_HOME* and the file */lib/systemd/system/neo4j.service*:

```
---
rm /lib/systemd/system/neo4j.service
rm -rf NEO4J_HOME
---
```

Windows installation

Before you install Neo4j on Windows, check System Requirements to see if your setup is suitable.

Windows console application

1. If it is not already installed, get OpenJDK 17 or Oracle Java 17. From version 5.14 onwards, Neo4j also supports JDK 21.
2. Download the latest release from Neo4j Deployment Center.

   Select the appropriate ZIP distribution.

3. Check that the SHA hash of the downloaded file is correct:
   a. To find the correct SHA hash, go to Neo4j Deployment Center and click `SHA-256`, which is located below your downloaded file.
   b. Using the appropriate commands for your platform, display the `SHA-256` hash for the file that you downloaded.
   c. Ensure that the two are identical.
4. Right-click the downloaded file and click **Extract All**.
5. Place the extracted files in a permanent home on your server and set the environment variable `NEO4J_HOME` to point to the extracted directory, for example, `export NEO4J_HOME=\path\to\_<NEO4J_HOME>` to make it easier to refer to it later.
6. Change the default locations of the *data*, *conf*, *certificates*, *licenses*, and *plugins* (if you plan to use custom plugins) directories by setting the environment variable `NEO4J_CONF` and the respective `server.directories.*` settings to point to the desired locations.

> **Note:**

Storing your Neo4j files outside `NEO4J_HOME` will simplify the upgrade process later because you will be able to replace the DBMS binaries without affecting the configuration and state. Otherwise, these Neo4j files will remain in the old installation folder and may be accidentally overwritten during an upgrade or deleted during a subsequent uninstall.

7. Accept either the commercial or the evaluation license agreement before running the Neo4j Enterprise Edition. If you are using Community Edition, you can skip this step.
   - Use one of the following options to accept the commercial license agreement. See the Neo4j licensing page for details on the available agreements.
     - Set it as an environment variable using `set NEO4J_ACCEPT_LICENSE_AGREEMENT=yes`.
     - Run `$NEO4J_HOME\bin\neo4j-admin server license --accept-commercial`
   - Use one of the following options to accept the Neo4j Evaluation Agreement for Neo4j Software.
     - Set it as an environment variable using `set NEO4J_ACCEPT_LICENSE_AGREEMENT=eval`.
     - Run `$NEO4J_HOME\bin\neo4j-admin server license --accept-evaluation`.
8. Before starting up the database for the first time, it is recommended to use the `set-initial-password` command of `neo4j-admin` to define the password for the native user `neo4j`.

   If the password is not set explicitly using this method, it will be set to the default password `neo4j`. In that case, you will be prompted to change the default password at first login.
   For more information, see Set an initial password.
9. Start Neo4j:
   - To run Neo4j as a console application, use: `$NEO4J_HOME\bin\neo4j console`.
   - To install Neo4j as a service use: `$NEO4J_HOME\bin\neo4j windows-service install`. For additional commands and to learn about the Windows PowerShell module included in the Zip file, see Windows PowerShell module.
10. Open *http://localhost:7474/* in your web browser.
11. Connect using the username `neo4j` with your password or the default password `neo4j`. You will then be prompted to change the password.
12. Stop the server by typing `Ctrl-C` in the console.

Windows service

Neo4j can also be run as a Windows service.

Warning:

By default, the Neo4j Windows service runs as the LocalSystem account, which has full access to the system. This is a security risk, and it is recommended to run the service as a user without full LocalSystem privileges.

Install the Windows service

Install the service with `bin\neo4j windows-service install`, and start it with `bin\neo4j start`.

The available commands for `bin\neo4j` are: `version`, `help`, `console`, `start`, `stop`, `restart`, `status`, and `windows-service`.

Note:

When installing a new release of Neo4j, you must first run `bin\neo4j windows-service uninstall` on any previously installed versions.

Change the Windows service configuration

When Neo4j is installed as a service, the Java options are stored in the service configuration file. If you want to change any of these options or environment variables after the service is installed, you must update and restart the service for the changes to take effect. For example, updating the value of `server.memory.heap.initial_size` in the default *neo4j.conf* file or by using the `NEO4J_CONF` environment variable will not automatically apply the changes. The service needs to be updated and restarted to pick them up. To update the service, run `bin\neo4j windows-service update`. Then restart the service to run it with the new configuration.

The same applies to the path to where Java is installed on the system. If the path changes, for example when upgrading to a new version of Java, it is necessary to run the `update-service` command and restart the service. Then, the new Java location will be used by the service.

Example 1. Update service example

1. Install service

```
bin\neo4j windows-service install
```

2. Change memory configuration

```
echo server.memory.heap.initial_size=8g >> conf\neo4j.conf
echo server.memory.heap.initial_size=16g >> conf\neo4j.conf
```

3. Update service

```
bin\neo4j windows-service update
```

4. Restart service

```
bin\neo4j restart
```

Windows PowerShell module

The Neo4j PowerShell module allows administrators to:

- Install, start, and stop Neo4j Windows® Services.
- Start tools, such as `Neo4j Admin` and `Cypher Shell`.

The PowerShell module is installed as part of the ZIP file distributions of Neo4j.

System requirements

- Requires PowerShell v2.0 or above.
- Supported on either 32 or 64-bit operating systems.

Manage Neo4j on Windows

On Windows, it is sometimes necessary to *Unblock* a downloaded ZIP file before you can import its contents as a module.

1. Right-click on the ZIP file and choose **Properties**.
   A dialog appears with an **Unblock** button.
2. Click the **Unblock** button to enable the import of the module.

   Running scripts has to be enabled on the system. This can, for example, be achieved by executing the following from an elevated PowerShell prompt:

   ```
   Set-ExecutionPolicy -ExecutionPolicy RemoteSigned
   ```

   For more information, see About execution policies.

The PowerShell module displays a warning if it detects that you do not have administrative rights.

Import the module file

The module file is located in the *bin* directory of your Neo4j installation.

1. Assuming that Neo4j is installed in *C:\Neo4j*, run the following command to import the module:

   ```
   Import-Module C:\Neo4j\bin\Neo4j-Management.psd1
   ```

   This adds the module to the current session.

2. Once the module is imported, you can start an interactive console version of a Neo4j Server:

   ```
   Invoke-Neo4j console
   ```

To stop the server, use `Ctrl-C` in the console window, created by the command.

Inspect the module

You can get all available commands in the module by running the following command:

```
Get-Command -Module Neo4j-Management
```

The output should be similar to the following:

```
CommandType     Name                                Version    Source
-----------     ----                                -------    ------
Function        Invoke-Neo4j                        5.26.25      Neo4j-Management
Function        Invoke-Neo4jAdmin                   5.26.25      Neo4j-Management
Function        Invoke-Neo4jBackup                  5.26.25      Neo4j-Management
Function        Invoke-Neo4jImport                  5.26.25      Neo4j-Management
Function        Invoke-Neo4jShell                   5.26.25      Neo4j-Management
```

The module also supports the standard PowerShell help commands

```
Get-Help Invoke-Neo4j
```

Run the following to see examples of help commands:

```
Get-Help Invoke-Neo4j -examples
```

Example usage

- List of available commands:

  ```
  Invoke-Neo4j
  ```

- Current status of the Neo4j service:

  ```
  Invoke-Neo4j status
  ```

- Install the service with verbose output:

  ```
  Invoke-Neo4j windows-service -Verbose
  ```

- Available commands for administrative tasks:

  ```
  Invoke-Neo4jAdmin
  ```

Common PowerShell parameters

The module commands support the common PowerShell parameter of `Verbose`.

Uninstall Neo4j

Here are the steps to uninstall Neo4j on Windows:

1. (Optional) Create a backup to avoid losing your data.
2. Stop all Neo4j processes by using the Task Manager.
3. Uninstall the Neo4j Windows service:

```
---
bin\neo4j windows-service uninstall
---
```

4. Delete *NEO4J_HOME*:

```
---
rmdir NEO4J_HOME
---
```

Neo4j Desktop

Neo4j Desktop is a convenient way for developers to work with local Neo4j databases.

> **Note:**
>
> Neo4j Desktop is not suited for production environments.

To install Neo4j Desktop, go to link:Neo4j Deployment Center and follow the instructions.

> **Tip:**
>
> For more information on how to use Neo4j Desktop and its capabilities, see the Neo4j Desktop documentation.

[1] In Neo4j 5.26 LTS, deploying Neo4j on MacOs 11, 12, and Windows 10 is deprecated and will be removed in the next release.
[2] In Neo4j 5.26 LTS, deploying Neo4j on Amazon Linux 2022 AMI, Unbuntu Server 16.04, 18.04, 20.04, and Windows Server 2016 and 2019 is deprecated and will be removed in the next release.

Cloud deployments

Neo4j's cloud marketplace listings represent a quick and easy way of getting started with graph databases on the cloud platform of your choice.

Cloud partner	Documentation link	Marketplace link
Amazon Web Services	Neo4j Enterprise on AWS	Neo4j in the AWS Marketplace
Microsoft Azure	Neo4j on Microsoft Azure	Neo4j in the Azure Marketplace

Note:

Deploying Neo4j on the GCP marketplace is supported only for version 2025.x. See Neo4j on GCP for more information.

Other cloud deployment options

Tip:

Neo4j Aura is a fully managed Neo4j database, hosted in the cloud and requires no installation. For more information, see the Aura product, support pages, and the Aura documentation.

Neo4j on AWS

Enterprise Edition

Neo4j can be easily deployed on EC2 instances in Amazon Web Services (AWS) by using the official listing for Neo4j Enterprise Edition on the AWS Marketplace.

The AWS Marketplace listing uses a CloudFormation template maintained by Neo4j, which can be customized to meet more complex or bespoke use cases. See Neo4j CloudFormation template for more information about the Neo4j CloudFormation template.

Note:

Neo4j does not provide Amazon Machine Images (AMIs) with a pre-installed version of the product. The Neo4j AWS Marketplace listings (and listings on GitHub) use CloudFormation templates that deploy and configure Neo4j dynamically with a shell script.

Supported Neo4j versions

The Neo4j AWS marketplace listing can be configured to deploy either Neo4j Enterprise Edition 2025.x or 5.26 LTS. The CloudFormation template always installs the latest available version.

Neo4j CloudFormation template

AWS CloudFormation is a declarative Infrastructure as Code (IaC) language that is based on YAML and instructs AWS to deploy a set of cloud resources.

The Neo4j CloudFormation template's code is available on GitHub. It takes several parameters as inputs, deploys a set of cloud resources, and provides outputs that can be used to connect to a Neo4j DBMS.

Important considerations

- The deployment of cloud resources incurs costs. Refer to the AWS pricing calculator for more information.
- The Neo4j CloudFormation template deploys a new VPC.
  - AWS accounts are limited to an initial quota of 5 VPCs (you can view your current quota on the **Limits** page of the Amazon EC2 console).
  - Your VPC quota can be increased if needed by contacting AWS support.
- The Neo4j CloudFormation template uses an Auto Scaling group (ASG) to deploy EC2 instances. This means that to stop or terminate EC2 instances, you must first remove them from the ASG, otherwise, the ASG will automatically replace them.
- SSH Keys are not generated as part of the CloudFormation template. Use EC2 Instance Connect (via the EC2 console) to connect to deployed EC2 instances if needed.

Input parameters

Parameter Name	Description	Allowed Values	Default Value
AMI ID	The AMI that will be used to launch EC2 resources.		`ami-07f83a354c934b879`
Neo4j License Type	Select `Enterprise` if you already have a Neo4j Enterprise License, otherwise select `Evaluation`.	`Enterprise` or `Evaluation`	`Evaluation`
Install Graph Data Science	An option to install Graph Data Science (GDS).	`Yes` or `No`	`False`
Graph Data Science License Key	License Key for Graph Data Science (License keys will be sent to and stored by Neo4j. This information will only be used for the purposes of product activation.)		`None`
Install Bloom	Install Neo4j Bloom.	`Yes` or `No`	`False`
Bloom License Key	License Key for Bloom (License keys will be sent to and stored by Neo4j. This information will only be used for the purposes of product activation.)		`None`

Parameter Name	Description	Allowed Values	Default Value
Password	A password for the `neo4j` user (minimum of 8 characters).		
Number of nodes	The number of desired EC2 instances to be used to form a Neo4j cluster (a minimum of 3 instances is required to form a cluster).	Between 1 and 10	`3`
Instance Type	EC2 instance type.	• `t3.medium` • `t3.large` • `t3.xlarge` • `t3.2xlarge` • `r6i.large` • `r6i.xlarge` • `r6i.2xlarge` • `r6i.4xlarge` • `r6i.8xlarge` • `r6i.12xlarge` • `r6i.16xlarge` • `r6i.24xlarge` • `r6i.32xlarge` • `r6a.8xlarge` • `r7a.medium` • `r7a.large` • `r7a.xlarge` • `r7a.2xlarge` • `r7a.4xlarge` • `r7a.8xlarge` • `r7a.12xlarge` • `r7a.16xlarge` • `r7a.24xlarge` • `r7a.32xlarge` • `r7a.48xlarge`	`t3.medium`
Disk Size	Size in GB of the EBS volume on each EC2 instance. Larger EBS volumes are typically faster than smaller ones, therefore 100GB is the recommended minimum size.	Minimum: 100	`100`

Parameter Name	Description	Allowed Values	Default Value
SSH CIDR	Specify an address range from which EC2 instances are accessible on port `22`, via SSH. You can use `0.0.0.0/0` to allow access from any IP address. This field must also be correctly populated to allow the use of EC2 instance-connect. Must be a valid CIDR range of the form x.x.x.x/x.	Minimum length: 9, Maximum length: 18, AllowedPattern: `1(\\d{1,3})\\.(\\d{1,3})\\.(\\d{1,3})\\.(\\d{1,3})/(\\d{1,2})`	

Deployed cloud resources

The environment created by the CloudFormation template consists of the following AWS resources:

- 1 VPC, with a CIDR range (address space) of `10.0.0.0/16`.
- 3 Subnets (if a cluster has been selected), distributed evenly across 3 Availability zones, with the following CIDR ranges:
  - `10.0.1.0/24`
  - `10.0.2.0/24`
  - `10.0.3.0/24`
- A single subnet (if a single instance has been selected) with the CIDR range `10.0.1.0/24`.
- An internal and external security group.
- An internet gateway.
- Routing tables (and associations) for all subnets.
- Neo4j http and bolt listeners and rules in the security group to allow inbound traffic on ports `7474` and `7687`.
- An auto-scaling group and launch configuration, which creates 1, or between 3 and 10 EC2 instances (Depending on whether a single instance or an autonomous cluster is selected).
- 1 Network (Layer 4) Load Balancer.
- A target group for the EC2 instances.

Template outputs

After the installation finishes successfully, the CloudFormation template provides the following outputs, which can be found in the **Outputs** tab of the CloudFormation page on the AWS console.

Output Name	Description
Neo4jBrowserURL	The http URL of the Neo4j Browser.
Neo4jURI	The Bolt URL of the Neo4j Browser.

Output Name	Description
Neo4jUsername	The username `neo4j` and a reminder to use the password that was specified earlier when filling out the CloudFormation template.

Neo4j cluster on AWS

Enterprise Edition

Cluster version consistency

When the CloudFormation template creates a new Neo4j cluster, an Auto Scaling group (ASG) is created and tagged with the monthly version of the installed Neo4j database. If you add more EC2 instances to your ASG, they will be installed with the same monthly version, ensuring that all Neo4j cluster servers are installed with the same version, regardless of when the EC2 instances were created.

Neo4j cluster and Auto Scaling Group

The Neo4j AWS CloudFormation template deploys a cluster into an ASG. However, managing a Neo4j cluster with ASG requires careful planning. Neo4j's clustering relies on stable servers identities. In contrast, ASGs are primarily designed for stateless, interchangeable workloads, which means they can terminate and recreate servers at any time, breaking their identity.

To provide stable servers identities, it is recommended to use a persistent disk, e.g., Amazon Elastic Block Store (EBS). For more information about EBS volumes, see Amazon EBS volumes. Amazon EBS provides block storage resources that can be used with Amazon EC2 instances.

If a server in Neo4j cluster does not remount its original EBS volume, it will either:

- Start with no data (store copy issue).
- Fail to rejoin the cluster correctly.

How to run Neo4j cluster in EC2 instances with ASGs

1. Create an EBS volume and tag it. You can follow the steps outlined in the guide Create an Amazon EBS volume.
2. Attach the EBS volume to an Amazon EC2 instance in the same Availability Zone. For more information, see Attach an Amazon EBS volume to an Amazon EC2 instance.
3. Ensure stable volume re-attachment. Use an instance startup script (via EC2 User data or `systemd` service) to:
    a. Identify its own logical identity (e.g., via private IP or hostname).
    b. Locate the correct EBS volume by tag.
    c. Attach the volume to the instance.
    d. Mount the volume to `/var/lib/neo4j` or appropriate data directory.
4. Use rolling updates only. Set `maxSurge = 0` and `maxUnavailable = 1` in any update mechanism

to avoid multiple restarts that could destabilize the cluster.

5. Avoid auto-healing on cluster members. ASG health checks should not terminate cluster members automatically. Use external monitoring (e.g., Prometheus, Neo4j's health checks) and manual intervention for cluster members.

Remove a server from the Neo4j cluster

Rolling updates on Amazon Machine Images (AMIs) often involve rotating the images. However, simply removing Neo4j servers from the target Network Load Balancer (NLB) one by one does not prevent requests from being routed to them. This occurs because the NLB and Neo4j server-side routing operate independently and do not share awareness of a server availability.

To correctly remove a server from the cluster and reintroduce it after the update, follow the steps outlined below:

1. Remove the server from the AWS NLB. This prevents external clients from sending requests to the server.
2. Since Neo4j's cluster routing (server-side routing) does not use the NLB, you need to ensure that queries are not routed to the server. To do this, you have to cleanly shut down the server.
   a. Run the following query to check servers are hosting all their assigned databases. The query should return no results:

   ```
   SHOW SERVERS YIELD name, hosting, requestedHosting, serverId WHERE requestedHosting <> hosting
   ```

   b. Use the following query to check all databases are in their expected state. The query should return no results:

   ```
   SHOW DATABASES YIELD name, address, currentStatus, requestedStatus, statusMessage WHERE
   currentStatus <> requestedStatus RETURN name, address, currentStatus, requestedStatus,
   statusMessage
   ```

   c. To stop the Neo4j service, run the following command:

   ```
   sudo systemctl stop neo4j
   ```

   To configure the timeout period for waiting on active transactions to either complete or be terminated before the shutdown, modify the setting `db.shutdown_transaction_end_timeout` in the *neo4j.conf* file. `db.shutdown_transaction_end_timeout` defaults to 10 seconds.

   The environment variable `NEO4J_SHUTDOWN_TIMEOUT` determines how long the system will wait for Neo4j to stop before forcefully terminating the process. You can change this using `systemctl edit neo4j.service`. By default, `NEO4J_SHUTDOWN_TIMEOUT` is set to 120 seconds. If the shutdown process exceeds this limit, it is considered failed. You may need to increase the value if the system serves long-running transactions.

   d. Verify that the shutdown process has finished successfully by checking the *neo4j.log* for relevant log messages confirming the shutdown.

3. When everything is updated or fixed, start the servers one by one again.
    a. Run `systemctl start neo4j`.
    b. Once the server has been restarted, confirm it is running successfully.

       Run the following command and check the server has state `Enabled` and health `Available`.

       ```
       SHOW SERVERS WHERE name = [server-id];
       ```

    c. Confirm that the server has started all the databases that it should.

       This command shows any databases that are not in their expected state:

       ```
       SHOW DATABASES YIELD name, address, currentStatus, requestedStatus, serverID WHERE currentStatus
       <> requestedStatus AND serverID = [server-id] RETURN name, address, currentStatus, requestedStatus
       ```

4. Reattach the server to the NLB. Once the server is stable and caught up, add it back to the AWS NLB target group.

Licensing

Installing and starting Neo4j from the AWS marketplace constitutes an acceptance of the Neo4j license agreement. When deploying Neo4j, users are required to confirm that they either have an enterprise license or accept the terms of the Neo4j evaluation license.

If you require the Enterprise version of either Graph Data Science or Bloom, you need to provide a key issued by Neo4j as this will be required during the installation.

To obtain a valid license for either Neo4j, Bloom, or GDS, reach out to your Neo4j account representative or get in touch using the contact form.

Delete CloudFormation Stack and destroy resources

Select the CloudFormation stack to be removed and click the **Delete** button. The stack deletion cleans up all AWS resources deployed by it.

Neo4j on Azure

Enterprise Edition

Neo4j can be easily deployed on Virtual Machine instances in Microsoft Azure by using the official listing for Neo4j Enterprise on the Azure Marketplace.

The Azure Marketplace listing uses an Azure Resource Manager (ARM) template maintained by Neo4j, which can be customized to meet more complex or bespoke use cases. See Neo4j ARM template for more information about the Neo4j ARM template.

Note:

Neo4j does not provide Azure Marketplace Virtual Machine Images with a pre-installed version of the product. The Neo4j Azure Marketplace listings (and listings on GitHub) use Azure Resource Manager (ARM) templates that deploy and configure Neo4j dynamically with a shell script.

Supported Neo4j versions

The Neo4j Azure marketplace listing can be configured to deploy Neo4j Enterprise Edition 5.26 LTS. The ARM template always installs the latest patch release.

Neo4j ARM template

Azure Resource Manager (ARM) is a declarative Infrastructure as Code (IaC) language that is based on JSON and instructs Azure to deploy a set of cloud resources.

The Neo4j ARM template's code is available on GitHub. The Neo4j ARM template takes several parameters as inputs, deploys a set of cloud resources, and provides outputs that can be used to connect to a Neo4j DBMS.

Important considerations

- The deployment of cloud resources incurs costs. Refer to the Azure pricing calculatorfor more information.
- You need to create a resource group. You need to choose an empty resource group or create a new one.
- An active Azure subscription is required.

Input parameters (Instance details)

Parameter Name	Description	Default Value
Region	The Azure region in which cloud resources should be deployed.	East US
Admin Username	The username for the administrator account on the Virtual Machine instances. This user will also be used to log into the Neo4j Browser.	`neo4j`
Admin Password	A password for the `neo4j` user. The password must be between 12 and 72 characters long, and contain characters from at least 3 of the following groups: uppercase characters, lowercase characters, numbers, and special characters.	

Input parameters (Neo4j Config)

Parameter Name	Description	Allowed Values	Default Value
Virtual Machine Size	The class of Azure VM instances to use.		`Standard_E8s_v5`

Parameter Name	Description	Allowed Values	Default Value
Node Count	The number of desired Virtual Machine instances to be used to form a Neo4j cluster (a minimum of 3 instances is required to form a cluster).	1 (for a single instance) or between 3 and 10 (for an autonomous cluster)	`1`
Disk Size	Size (in GB) of disk on each Azure VM instance.		`32 GB`
Graph Database Version	The version of the Neo4j database to install.		`5.26.x`
Install Graph Data Science	An option to install Graph Data Science (GDS). Accepted values are `Yes` or `No`.		`False`
Graph Data Science License Key	A valid GDS license key can be pasted into this field. License keys will be sent to and stored by Neo4j. This information is used only for product activation purposes.		`None`
Install Bloom	Optionally install Neo4j Bloom. Accepted values are `Yes` or `No`.		`False`
Bloom License Key	A valid Bloom license key can be pasted into this field. License keys will be sent to and stored by Neo4j. This information is used only for product activation purposes.		`None`
License Type	The type of Neo4j license to use.	Accepted values are `BYOL` or `Evaluation`.	`Evaluation`

Deployed cloud resources

The environment created by the ARM template consists of the following Azure resources:

- 1 Virtual Network with a CIDR range (address space) of `10.0.0.0/8`.
  - A single subnet with a CIDR range `10.0.0.0/16`.
  - A network security group.
- A Virtual Machine Scale-Set (VMSS), which creates 1, or between 3 and 10 Virtual Machine instances, depending on whether a single instance or an autonomous cluster is selected.
- 1 Load Balancer.

Template outputs

After the deployment finishes successfully, the ARM template provides the following outputs, which can be found in the **Outputs** section of the deployments page in the Azure console.

Note:

At the end of the deployment process, Azure runs a validation. If the validation fails, it might be because you have chosen VMs that are too large and exceed your Azure quota.

Output Name	Output Description
neo4jBrowserURL	The http URL of the Neo4j Browser.
username	The username `neo4j` which is required to log into the Neo4j Browser.

Cluster version consistency

When the ARM template creates a new Neo4j cluster, a Virtual Machine Scale Set (VMSS) is created and tagged with the minor version of the installed Neo4j database. If you add more VM instances to your VMSS, they will be installed using the same minor version, ensuring that all Neo4j cluster servers are installed with the same version, regardless of when the VM instances were created.

Licensing

Installing and starting Neo4j from the Azure marketplace constitutes an acceptance of the Neo4j license agreement. When deploying Neo4j, users are required to confirm that they either have an enterprise license.

If you require the enterprise version of either Graph Data Science or Bloom, you need to provide a key issued by Neo4j as this will be required during the installation.

To obtain a valid license for either Neo4j, Bloom, or GDS, reach out to your Neo4j account representative or get in touch using the contact form.

Delete Azure deployment Stack and destroy resources

In order to completely delete the deployment, the resource group can be deleted.

Docker

Neo4j can be run in a Docker container.

This chapter describes the following:

- Getting Started with Neo4j in Docker — Introduction to running Neo4j in a Docker container.
- Persisting data with Docker volumes — How and where to mount persistent storage to the Docker container.
- Modify the default configuration — How to configure Neo4j to run in a Docker container.
- Plugins — How to load plugins when using Neo4j in Docker.
- Deploy a Neo4j server with Docker Compose — How to set up a Neo4j server with Docker Compose using a basic authentication mechanism or Docker secrets.
- Deploy a Neo4j cluster on Docker — How to set up and deploy a Neo4j cluster on Docker.
- Docker specific operations — Descriptions of various `neo4j-admin` and `cypher-shell` operations that are specific to using Docker.
- Offline dump and load — How to perform dump and load of a containerized Neo4j database.
- Online backup and restore — How to perform backup and restore of a containerized Neo4j database. Enterprise Only.
- Security — Information about using encryption with a Neo4j Docker image.
- Docker specific configuration settings — A conversion table for the Neo4j configuration settings to Docker format.

Note:

Docker does not run natively on macOS or Windows. For running Docker on macOS and Windows, please consult the documentation provided by Docker.

Getting started with Neo4j in Docker

Docker can be downloaded for macOS, Windows, and Linux operating systems from https://www.docker.com/get-started/. DockerHub hosts an official Neo4j image that provides a standard, ready-to-run package of Neo4j Community Edition and Enterprise Edition for a variety of versions.

Getting the Neo4j image

Variants of the Neo4j image are tagged according to Community/Enterprise Edition and the operating system used as the base image.

Neo4j editions

Tags are available for both Community Edition and Enterprise Edition. Version-specific Enterprise Edition tags have an `-enterprise` suffix after the version number, for example: `neo4j:5.26.25-enterprise`.

Community Edition tags have no suffix, for example `neo4j:5.26.25`. The latest Neo4j Enterprise Edition release is available as `neo4j:enterprise`.

Neo4j Enterprise Edition license

To use Neo4j Enterprise Edition, you must accept the license agreement by setting the environment variable `NEO4J_ACCEPT_LICENSE_AGREEMENT=yes`.

Base operating system

The Neo4j image is available with either `debian:trixie-slim`, `debian:bullseye-slim`, `redhat/ubi10-minimal:latest`, or `redhat/ubi9-minimal:latest` as the base image.

Starting with 5.26.20, the default is `debian:trixie-slim` (updated from `debian:bullseye-slim`).

Tip:

If you are unsure which base image to use or have no preference, just use the default of `neo4j:5.26.25`.

To specify which base image to use, the image tags optionally have a `-trixie` `-bullseye`, `-ubi10`, or `-ubi9` suffix.

For example:

```
neo4j:5.26.25-trixie              # debian 13 community
neo4j:5.26.25-enterprise-bullseye # debian 11 enterprise
neo4j:5.26.25-ubi9               # redhat UBI9 community
neo4j:5.26.25-enterprise-ubi10   # redhat UBI10 enterprise
neo4j:5.26.25                    # debian 13 community
neo4j:5.26.25-enterprise         # debian 13 enterprise
```

Table 9. Base images and the corresponding tag suffix.

tag suffix	Base Image
`-trixie`	`debian:trixie-slim`
`-bullseye`	`debian:bullseye-slim`
`-ubi10`	`redhat/ubi10-minimal:latest`
`-ubi9`	`redhat/ubi9-minimal:latest`

tag suffix	Base Image
unspecified	`debian:trixie-slim`

Note:

- The Red Hat UBI9 variant images are only available from 5.17.0 and onwards.
- The Red Hat UBI10 variant images are only available from 5.26.20 and onwards.
- The Debian 13 variant images are only available from 5.26.20 and onwards.
- For earlier Neo4j versions, do not specify a base image.

Using the Neo4j Docker image

You can start a Neo4j container by using the following command. Note that this Neo4j container will not persist data between restarts and will have the default username/password.

```
docker run \
    --restart always \
    --publish=7474:7474 --publish=7687:7687 \
    neo4j:5.26.25
```

You can try out your Neo4j container by opening *http://localhost:7474/* (the Neo4j's Browser interface) in a web browser. By default, Neo4j requires authentication and prompts you to log in with a username/password of `neo4j/neo4j` at the first connection. You are then prompted to set a new password.

Note:

The default minimum password length is 8 characters.

Use the `dbms.security.auth_minimum_password_length` configuration setting to change the default minimum value.

The following sections provide more information about how to set an initial password, configure Neo4j to persist data between restarts, and use the Neo4j Docker image.

Persisting data between restarts

The `--volume` option maps a local folder to the container, where you can persist data between restarts. To persist the contents of the database between containers, mount a volume to the `/data` directory on starting the container:

```
docker run \
    --restart always \
    --publish=7474:7474 --publish=7687:7687 \
    --env NEO4J_AUTH=neo4j/your_password \
    --volume=/path/to/your/data:/data \
    neo4j:5.26.25
```

Caution:

The folders that you want to mount must exist before starting Docker, otherwise, Neo4j fails to start due to permissions errors.

For more information about mounting volumes, see Persisting data with Docker volumes.

Using `NEO4J_AUTH` to set an initial password

When using Neo4j in a Docker container, you can set the initial password for the database directly by specifying the `NEO4J_AUTH` in your run directive:

```
docker run \
    --restart always \
    --publish=7474:7474 --publish=7687:7687 \
    --env NEO4J_AUTH=neo4j/your_password \
    neo4j:5.26.25
```

Alternatively, you can disable authentication by specifying `NEO4J_AUTH` to `none`:

```
--env NEO4J_AUTH=none
```

Note that there is currently no way to change the initial username from `neo4j`.

Caution:

Setting `NEO4J_AUTH` does not override the existing authentication.

If you have mounted a */data* volume containing an existing database, setting `NEO4J_AUTH` will have no effect because that database already has authentication configured. The Neo4j Docker service will start, but you will need a username and password already associated with the database to log in.

Useful `docker run` options

This table lists some of the options available:

Table 10. Options for `docker run`

Option	Description	Example
`--name`	Name your container to avoid generic ID.	`docker run --name myneo4j neo4j`
`-p`	Specify which container port to expose.	`docker run -p7687:7687 neo4j`
`-d`	Detach container to run in the background.	`docker run -d neo4j`
`-v`	Bind mount a volume.	`docker run -v $HOME/neo4j/data:/data neo4j`
`--env`	Set config as environment variables for the Neo4j database.	`docker run --env NEO4J_AUTH=neo4j/your_password neo4j`

Option	Description	Example
`--user`	Run neo4j as the given user, instead of `neo4j`.	`docker run --user="$(id -u):$(id -g)" neo4j`
`--restart`	Control whether Neo4j containers start automatically when they exit, or when Docker restarts.	`docker run --restart always`
`--help`	Output full list of `docker run` options	`docker run --help`

Note:

The `--restart always` option sets the Neo4j container (and Neo4j) to restart automatically whenever the Docker daemon is restarted.

If you no longer want to have the container auto-start on machine boot, you can disable this setting using the flag `no`, for example, `docker update --restart=no <containerID>`.
For more information on Docker restart policies, see the official Docker documentation.

Offline installation of Neo4j Docker image

Docker provides the `docker save` command for downloading an image into a `.tar` package so that it can be used offline, or transferred to a machine without internet access.

This is an example command to save the `neo4j:5.26.25` image to a `.tar` file:

```
docker save -o neo4j-5.26.25.tar neo4j:5.26.25
```

To load a docker image from a `.tar` file created by `docker save`, use the `docker load` command. For example:

```
docker load --input neo4j-5.26.25.tar
```

For complete instructions on using the `docker save` and `docker load` commands, refer to:

- The official `docker save` documentation.
- The official `docker load` documentation.

Persisting data with Docker volumes

Docker containers are ephemeral. When a container is stopped, any data written to it is lost. Therefore, if you want to persist data when using Neo4j in Docker, you must mount storage to the container. Storages also allow you to get data in and out of the container.

Storage can be mounted to a container in two ways:

- A folder on the host file system.
- A Docker volume — a named storage location that is managed by Docker.

For instructions on *how* to mount storage to a Docker container, refer to the official Docker documentation Bind mounts and Volumes.

Neo4j provides several mount points for storage to simplify using Neo4j in Docker. The following sections describe the mount points and how to use them.

Neo4j mount points and permissions

The following table is a complete reference of the mount points recognized by the Neo4j Docker image, and file permissions.

All the listed mount points are **optional**. Neo4j can run in Docker without any volumes mounted at all. However, mounting storage to `/data` is considered essential for all but the most basic use cases.

Warning:

Running containerized Neo4j without a `/data` mount results in **unrecoverable data loss** if anything happens to the container.

Table 11. Mount points for the Neo4j container

Mount point	Permissions required	Description
`/data`	read, write	The data store for the Neo4j database. See Mounting storage to `/data`.
`/logs`	read, write	Output directory for Neo4j logs. See Mounting storage to `/logs`.
`/conf`	read[1]	Pass configuration files to Neo4j on startup. See Modify the default configuration.
`/plugins`	read[2]	Allows you to install plugins in containerized Neo4j. See Plugins.
`/licenses`	read	Provide licenses for Neo4j and any plugins by mounting the license folder. See Installing Plugin Licenses.
`/import`	read	Make csv and other importable files available to neo4j-admin import.
`/ssl`	read	Provide SSL certificates to Neo4j for message encryption. See SSL encryption in a Neo4j Docker container
`/metrics`	write	Enterprise Edition Output directory for metrics files. See Metrics.

Mounting storage to `/data`

Neo4j inside Docker stores database files in the `/data` folder. By mounting storage to `/data`, any data written to Neo4j will persist after the container is stopped.

Stopping the container and then restarting with the same folder mounted to `/data` starts a new containerized Neo4j instance with the same data.

Caution:

If Neo4j could not properly close down, it may have left data in a bad state and is likely to fail on startup. This is the same as if Neo4j is run outside a container and not closed properly.

Example 2. Two ways to mount storage to the `/data` mount point

Mounting a folder to `/data`

```
docker run -it --rm \
   --volume $HOME/neo4j/data:/data \
   neo4j:5.26.25
```

Creating a named volume and mounting it to `/data`

```
docker volume create neo4jdata
docker run -it --rm \
   --volume neo4jdata:/data \
   neo4j:5.26.25
```

Create a Docker volume named `neo4jdata`.

Mount the volume name `neo4jdata` to `/data`.

Mounting storage to `/logs`

Neo4j logging output is written to files in the */logs* directory. This directory is mounted as a */logs* volume. By mounting storage to `/logs`, the log files become available outside the container.

Tip:

For more information about configuring Neo4j, see Configuration.
For more information about the Neo4j log files, see Logging.

File permissions

For security reasons, by default, Neo4j runs as the `neo4j` user inside the container. This user has user ID `7474`. If `neo4j` needs read or write access to a mounted folder, but does not have it, the folder will be automatically re-owned to `7474`.

Note:

This is a convenient feature, so you do not have to worry about the finer details of file permissions in Docker and can get started more easily. It does however mean that mounted folders change ownership, and you may find you can no longer read your files without root access.

Docker `run` with `--user` flag

The `--user` flag to `docker run` forces Docker to run as the provided user. In this situation, if that user does not have the required read or write access to any mounted folders, Neo4j will fail to start.

Modify the default configuration

The default configuration provided by the Neo4j image is intended for learning about Neo4j but must be modified to make it suitable for production use. In particular, the default memory assignments to Neo4j are very limited (`NEO4J_server_memory_pagecache_size=512M` and `NEO4J_server_memory_heap_max__size=512M`), to allow multiple containers to be run on the same server. You can read more about configuring Neo4j in the Docker specific configuration settings.

There are three ways to modify the configuration:

- Set environment variables.
- Mount a */conf* volume.
- Build a new image.

Which one to choose depends on how much you need to customize the image.

Environment variables

Pass environment variables to the container when you run it, for example:

```
docker run \
    --detach \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j/data:/data \
    --volume=$HOME/neo4j/logs:/logs \
    --env NEO4J_dbms_memory_pagecache_size=4G \
    neo4j:5.26.25
```

Any configuration value (see Configuration settings) can be passed using the following naming scheme:

- Prefix with `NEO4J_`.
- Underscores must be written twice: `_` is written as `__`.
- Periods are converted to underscores: `.` is written as `_`.

As an example, `db.tx_log.rotation.size` could be set by specifying the following argument to Docker:

```
--env NEO4J_db_tx__log_rotation_size
```

Variables that can take multiple options, such as `NEO4J_server_jvm_additional`, must be defined just once, and include a concatenation of the multiple values. For example:

```
--env NEO4J_server_jvm_additional="-Dcom.sun.management.jmxremote.authenticate=true
-Dcom.sun.management.jmxremote.ssl=false -Dcom.sun.management.jmxremote.password.file=
$HOME/conf/jmx.password -Dcom.sun.management.jmxremote.access.file=$HOME/conf/jmx.access
```

```
-Dcom.sun.management.jmxremote.port=3637"
```

> **Note:**
>
> From Neo4j 5.6 onwards, Docker additional configuration settings via an environment variable for JVM no longer override the default JVM configurations but are appended to them.

Mounting the */conf* volume

To make arbitrary modifications to the Neo4j configuration, provide the container with a */conf* volume:

```
docker run \
    --detach \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j/data:/data \
    --volume=$HOME/neo4j/logs:/logs \
    --volume=$HOME/neo4j/conf:/conf \
    neo4j:5.26.25
```

The configuration files in the */conf* volume override the files provided by the image. So if you want to change one value in a file, you must ensure that the rest of the file is complete and correct. Environment variables passed to the container by Docker override the values in configuration files in */conf* volume.

> **Note:**
>
> If you use a configuration volume you must make sure to listen on all network interfaces. This can be done by setting `server.default_listen_address=0.0.0.0`.

To dump the initial set of configuration files, run the image with the `dump-config` command. You must set the `neo4j` user as the owner of *$HOME/neo4j/conf* to allow *write* access from the Neo4j Docker container:

```
sudo chown neo4j:neo4j $HOME/neo4j/conf
```

> **Note:**
>
> Processes in the Neo4j docker container run under the `neo4j` user by default.

```
docker run --rm \
    --volume=$HOME/neo4j/conf:/conf \
    neo4j:5.26.25 dump-config
```

Customize a Neo4j Docker image

To customize a Neo4j Docker image, you create a custom Dockerfile based on a Neo4j image (using the `FROM` instruction), build that image, and run a container based on it.

Tip:

It is recommended to specify an explicit version of the base Neo4j Docker image. For available Neo4j Docker images, see https://hub.docker.com/_/neo4j.

Additionally, you can pass `EXTENSION_SCRIPT` as an environment variable, pointing to a location in a folder you need to mount. You can use this script to perform an additional initialization or configuration of the environment, for example, loading credentials or dynamically setting *neo4j.conf* settings, etc. The Neo4j image `entrypoint` script will check for the presence of an `EXTENSION_SCRIPT` environment variable. If set, it will first execute the `entrypoint` code, then the extension script specified, and finally, it will start Neo4j.

The following is an example of how to create a custom Dockerfile based on a Neo4j image, build the image, and run a container based on it. It also shows how to use the `EXTENSION_SCRIPT` feature.

```
# Create a custom Dockerfile based on a Neo4j image:

/example/Dockerfile

FROM neo4j:5.26.25-enterprise
COPY extension_script.sh /extension_script.sh
ENV EXTENSION_SCRIPT=/extension_script.sh

/example/extension_script.sh

echo "extension logic"

# Build the custom image:

docker build --file /example/Dockerfile --tag neo4j:5.26.25-enterprise-custom-container-1 /example

# Create and run a container based on the custom image:

docker run --interactive --tty --name custom-container-1 -p7687:7687 -p7474:7474 -p7473:7473 --env
NEO4J_AUTH=neo4j/your_password --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes neo4j:5.26.25-enterprise-custom-
container-1
```

The recommended best practices and methods for building efficient Docker images can be found at the Docker documentation → Best practices for writing Dockerfiles.

Plugins

This page describes how to install plugins into a Neo4j instance running inside a Docker container. For instructions about plugins in general see Configuration → Plugins.

Installing plugins

To install plugins, including user-defined procedures, mount the folder or volume containing the plugin JARs to `/plugins`, for example:

```
docker run \
   --publish=7474:7474 --publish=7687:7687 \
   --volume=$HOME/neo4j/plugins:/plugins \
   neo4j:5.26.25
```

Neo4j automatically loads any plugins found in the `/plugins` folder on startup.

`NEO4J_PLUGINS` utility

The Neo4j Docker image includes a startup script that can automatically download and configure certain Neo4j plugins at runtime.

> **Note:**
>
> This feature is intended to facilitate the use of the Neo4j plugins in development environments, but it is not recommended for production environments.
>
> To use plugins in production with Neo4j Docker containers, see Install user-defined procedures.

The `NEO4J_PLUGINS` environment variable can be used to specify the plugins to install using this method. This should be set to a JSON-formatted list of the supported plugins.

> **Note:**
>
> Running Bloom in a Docker container requires Neo4j Docker image 4.2.3-enterprise or later.

If invalid `NEO4J_PLUGINS` values are passed, Neo4j returns a notification that the plugin is not known. For example, `--env NEO4J_PLUGINS='["gds"]'` returns the following notification:

Example output

```
"gds" is not a known Neo4j plugin. Options are:
apoc
apoc-extended
bloom
genai
graph-data-science
n10s
```

Example 3. Install the APOC Core plugin (`apoc`)

You can use the Docker argument `--env NEO4J_PLUGINS='["apoc"]'` and run the following command:

```
docker run -it --rm \
  --publish=7474:7474 --publish=7687:7687 \
  --env NEO4J_AUTH=none \
  --env NEO4J_PLUGINS='["apoc"]' \
  neo4j:5.26.25
```

Example 4. Install the APOC Core plugin (`apoc`) and the Graph Data Science plugin (`graph-data-science`)

You can use the Docker argument `--env NEO4J_PLUGINS='["apoc", "graph-data-science"]'` and run the following command:

```
docker run -it --rm \
  --publish=7474:7474 --publish=7687:7687 \
  --env NEO4J_AUTH=none \
  --env NEO4J_PLUGINS='["apoc", "graph-data-science"]' \
```

```
  neo4j:5.26.25
```

Storing downloaded plugins

In situations where bandwidth is limited, or Neo4j is stopped and started frequently, it may be desirable to download plugins once and re-use them rather than downloading them each time.

By using the `NEO4J_PLUGINS` utility in combination with mounting storage to `/plugins`, the plugin jars are downloaded into the `/plugins` folder. This can then be used again later to supply the same plugins to Neo4j without needing to set `NEO4J_PLUGINS`.

Example 5. Example of automatically downloading and re-using plugins with docker.

Get the APOC plugin and save it into `$HOME/neo4j/plugins`

```
docker run -it --rm \
  --publish=7474:7474 --publish=7687:7687 \
  --env NEO4J_AUTH=none \
  --env NEO4J_PLUGINS='["apoc"]' \
   --volume=$HOME/neo4j/plugins:/plugins \
  neo4j:5.26.25
```

Mounts host folder `$HOME/neo4j/plugins` to `/plugins`.

Verify the `apoc` *plugin is downloaded.*

```
docker kill <containerID/name>
ls $HOME/neo4j/plugins
  apoc.jar
```

Start a new container and verify `apoc` *is installed.*

```
docker run -it --rm \
  --publish=7474:7474 --publish=7687:7687 \
  --env NEO4J_AUTH=none \
   --volume=$HOME/neo4j/plugins:/plugins \
  neo4j:5.26.25

cypher-shell "RETURN apoc.version();"
```

Installing plugin licenses

If a plugin requires a license, the license file can be supplied to the container by mounting the folder or volume containing license file(s) to `/licenses`.

Note:

To check if the plugin requires a license, refer to the general plugin documentation.

Example 6. Installing plugins and licenses by mounting folders to the container

```
docker run \
   --publish=7474:7474 --publish=7687:7687 \
   --volume=$HOME/neo4j/plugins:/plugins \
   --volume=$HOME/neo4j/licenses:/licenses \
   neo4j:5.26.25
```

folder containing plugin jars.

folder containing license files.

The licenses must also be provided if using the `NEO4J_PLUGINS` utility to install the plugins.

Example 7. Installing plugins and licenses by mounting folders to the container using `NEO4J_PLUGINS` utility

```
docker run \
   --publish=7474:7474 --publish=7687:7687 \
   --env NEO4J_PLUGINS='["bloom"]' \
   --volume=$HOME/neo4j/licenses:/licenses \
   neo4j:5.26.25
```

A folder containing license files.

Deploy a Neo4j standalone server using Docker Compose

Introduced in 5.24

You can deploy a Neo4j standalone server using Docker Compose by defining the container configuration in a *docker-compose.yml* file and authenticating with basic authentication or Docker secrets.

Deploy a Neo4j server using basic authentication mechanism

Before you start, verify that you have installed Docker Compose. For more information, see the Install Docker Compose official documentation.

1. Create a project folder where you will store your *docker-compose.yml* file and run your Neo4j server.
2. Prepare your *docker-compose.yml* file using the following example. For more information, see the Docker Compose official documentation on Docker Compose specification.

   Example of a docker-compose.yml file

```
services:
  neo4j:
    image: neo4j:latest
    volumes:
        - /$HOME/neo4j/logs:/logs
        - /$HOME/neo4j/config:/config
        - /$HOME/neo4j/data:/data
        - /$HOME/neo4j/plugins:/plugins
    environment:
        - NEO4J_AUTH=neo4j/your_password
    ports:
```

```
      - "7474:7474"
      - "7687:7687"
    restart: always
```

Mount the */$HOME/neo4j/<..>*: directories to local directories on your host machine to store logs, configuration, data, and plugins. For more information about mounting volumes, see Persisting data with Docker volumes.

Set the `neo4j` username and password.

3. Deploy your Neo4j server by running `docker-compose up` from your project folder.

```
docker-compose up -d
```

The `-d` flag starts the container in detached mode.

Deploy a Neo4j server with Docker secrets

Recommended

It is advisable not to store sensitive information, such as the database username and password, in the *docker-compose.yml* file. You can instead store your credentials in files and use them in your *docker-compose.yml* file without exposing their values.

1. Create a file, for example, *neo4j_auth.txt*, containing the username and password for the Neo4j server to be used as a Docker secret.

```
neo4j/your_password
```

2. Prepare your *docker-compose.yml* file using the following example. For more information, see the Docker Compose official documentation on Docker Compose specification.

Example of a docker-compose.yml file

```
services:
  neo4j:
    image: neo4j:latest
    volumes:
        - /$HOME/neo4j/logs:/logs
        - /$HOME/neo4j/config:/config
        - /$HOME/neo4j/data:/data
        - /$HOME/neo4j/plugin:/plugins
    environment:
        - NEO4J_AUTH_FILE=/run/secrets/neo4j_auth_file
    ports:
      - "7474:7474"
      - "7687:7687"
    restart: always
    secrets:
      - neo4j_auth_file
secrets:
  neo4j_auth_file:
    file: ./neo4j_auth.txt
```

Mount the */$HOME/neo4j/<..>*: directories to local directories on your host machine to store logs, configuration, data, and plugins.

Path to the secret (`neo4j_auth_file`) containing the `neo4j` username and password. The secret value is read from the file specified in the `file` attribute of the `neo4j_auth_file` secret. Multiple secrets can be defined in the `secrets` section of the `neo4j` service. Secrets only support environment variables starting with `NEO4J_` and ending with `_FILE`.

The name of the secret, for example `neo4j_auth_file`.

Path to the *neo4j_auth.txt* file.

The name of the secret in the `neo4j` service.

Warning:

The secret value overrides the equivalent environment variable if they are both defined. So, for example, if you also define an environment variable `NEO4J_AUTH=neo4j/your_other_password` in the `environment` section of the `neo4j` service, the value of `NEO4J_AUTH_FILE` will be the one used.

3. Deploy your Neo4j server by running `docker-compose up` from your project folder.

```
docker-compose up -d
```

The `-d` flag starts the container in detached mode.

Deploy a Neo4j cluster on multiple Docker hosts

Neo4j supports clustering in a containerized environment without an orchestration tool. This section describes how to use Docker to set up a cluster across multiple machines. For a tutorial on how to set up a cluster locally for testing purposes, see Tutorials → Deploying a Neo4j cluster in a Docker container.

Note:

The examples on this page make use of both command expansion and DNS discovery method. For more information, see:

- Command expansion
- Discovery using DNS with multiple records

To create a highly-available cluster of containers, the Neo4j cluster servers can be deployed on different physical machines.

When each container is running on its own physical machine, and the Docker network is not used, you have to define the advertised addresses to enable communication between the physical machines. Each container must also bind to the host machine's network. For more information about container networking, see the Docker official documentation.

Example of a `docker run` command for invoking a cluster member:

```
docker run --name=server1 --detach \
```

```
        --network=host \
        --publish=7474:7474 --publish=7687:7687 \
        --publish=5000:5000 --publish=6000:6000 --publish=7000:7000 \
        --hostname=public-address \
        --env NEO4J_dbms_cluster_discovery_endpoints=server1-public-address:5000,server2-public-
address:5000,server3-public-address:5000 \
        --env NEO4J_server_discovery_advertised__address=public-address:5000 \
        --env NEO4J_server_cluster_advertised__address=public-address:6000 \
        --env NEO4J_server_cluster.raft.advertised__address=public-address:7000 \
        --env NEO4J_server_default__advertised__address=public-address \
        --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
        --env NEO4J_server_bolt_advertised__address=public-address:7687 \
        --env NEO4J_server_http_advertised__address=public-address:7474 \
        neo4j:5.26.25-enterprise
```

Where `public-address` is the public hostname or ip-address of the machine.

Docker-specific operations

You can use the Neo4j tools when running Neo4j in a Docker container.

Use Neo4j Admin

The Neo4j Admin tool can be run locally within a container using the following command:

```
docker exec --interactive --tty <containerID/name> neo4j-admin <category> <command>
```

To determine the container ID or name, run `docker ps` to list the currently running Docker containers.

For more information about the `neo4j-admin` commands, see Neo4j Admin and Neo4j CLI.

Use Neo4j Import

The Neo4j Import tool can be run locally within a container using the following commands:

```
docker exec --interactive --tty <containerID/name> neo4j-admin database import full <options>
```

and

```
docker exec --interactive --tty <containerID/name> neo4j-admin database import incremental <options>
```

For more information about the commands' syntax and options, see Full import and Incremental import.

Prerequisites

- Verify that you have created the folders that you want to mount as volumes to the Neo4j docker container.
- Verify that the CSV files that you want to load into Neo4j are formatted as per CSV header format.
- Verify that you have added the CSV files to the folder that will be mounted to */import* in your container.

Import CSV files into the Neo4j Docker container using the Neo4j import tool

This is an example of how to start a container with mounted volumes */data* and */import*, to ensure the persistence of the data in them, and load the CSV files using the `neo4j-admin database import full` command. You can add the flag `--rm` to automatically remove the container's file system when the container exits.

```
docker run --interactive --tty --rm \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j/data:/data \
    --volume=$HOME/neo4j/import:/import \
    neo4j:5.26.25 \
neo4j-admin database import full --nodes=Movies=/import/movies_header.csv,/import/movies.csv \
--nodes=Actors=/import/actors_header.csv,/import/actors.csv \
--relationships=ACTED_IN=/import/roles_header.csv,/import/roles.csv databasename
```

Use Neo4j Admin for memory recommendations

The neo4j-admin server memory-recommendation command with the argument `--docker` outputs environmental variables that can be passed to a Neo4j docker container. The following example shows how `neo4j-admin server memory-recommendation --docker` provides a memory recommendation in a docker-friendly format.

Example 8. Invoke `neo4j-admin server memory-recommendation --docker`

```
bin/neo4j-admin server memory-recommendation --memory=16g --docker

...
...
...
# Based on the above, the following memory settings are recommended:
NEO4J_server_memory_heap_initial__size='5g'
NEO4J_server_memory_heap_max__size='5g'
NEO4J_server_memory_pagecache_size='7g'
#
# It is also recommended turning out-of-memory errors into full crashes,
# instead of allowing a partially crashed database to continue running:
NEO4J_server_jvm_additional='-XX:+ExitOnOutOfMemoryError'
#
...
...
```

Use Neo4j Admin report

The Neo4j Admin report tool generates a report of the status of a running Neo4j database.
In a containerized environment, its command `neo4j-admin server report` must be invoked using the script `neo4j-admin-report`. This ensures that the reporter is running with all the necessary file permissions required to analyze the running Neo4j processes. This script takes all the arguments of the `neo4j-admin server report` command.

> **Note:**
>
> It is possible to use `docker exec` to run `neo4j-admin server report` directly inside the container, but the file permissions required to access running processes *and* to write files to

a mounted folder can be conflicting.

The `neo4j-admin-report` script is just a wrapper around `neo4j-admin server report` which automatically handles most permission problems.

Example 9. Example of getting a Neo4j report from a containerised Neo4j database

Start a Neo4j container and mount a folder for storing the reports.

```
docker run --interactive --tty --rm \
    --name=neo4j \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j/data:/data \
    --volume=$HOME/neo4j/reports:/reports \
    neo4j:5.26.25
```

The output folder for the reports.

Then, using `docker exec`, run the `neo4j-admin-report` wrapper script, specifying the output directory for the reports.

```
docker exec --interactive --tty <containerID/name> \
    neo4j-admin-report \
        --to-path=/reports \
```

If no `--to-path` option is specified, the reports will be written to `/tmp/reports`.

The `$HOME/neo4j/reports` folder should now contain a zip file of reports.

Use Cypher Shell

The Neo4j Cypher Shell tool can be run locally within a container using the following command:

```
docker exec --interactive --tty <containerID/name> cypher-shell <options>
```

For more information about the `cypher-shell` syntax and options, see Syntax.

Retrieve data from a database in a Neo4j Docker container

The following is an example of how to use the `cypher-shell` command to retrieve data from the `neo4j` database.

1. Run a new container, mounting the same volume /data as in the import example.

   ```
   docker run --interactive --tty --name <containerID/name> \
       --publish=7474:7474 --publish=7687:7687 \
       --volume=$HOME/neo4j/data:/data \
       neo4j:5.26.25
   ```

2. Use the container ID or name to get into the container, and then, run the `cypher-shell` command and

authenticate.

```
docker exec --interactive --tty <containerID/name> cypher-shell -u neo4j -p <password>
```

3. Retrieve some data.

```
neo4j@neo4j> match (n:Actors)-[r]->(m:Movies) return n.name AS Actors, m.title AS Movies, m.year AS
MovieYear;
+-------------------------------------------------------------+
| Actors               | Movies                   | MovieYear |
+-------------------------------------------------------------+
| "Keanu Reeves"       | "The Matrix Revolutions" | 2003      |
| "Keanu Reeves"       | "The Matrix Reloaded"    | 2003      |
| "Keanu Reeves"       | "The Matrix"             | 1999      |
| "Laurence Fishburne" | "The Matrix Revolutions" | 2003      |
| "Laurence Fishburne" | "The Matrix Reloaded"    | 2003      |
| "Laurence Fishburne" | "The Matrix"             | 1999      |
| "Carrie-Anne Moss"   | "The Matrix Revolutions" | 2003      |
| "Carrie-Anne Moss"   | "The Matrix Reloaded"    | 2003      |
| "Carrie-Anne Moss"   | "The Matrix"             | 1999      |
+-------------------------------------------------------------+

9 rows available after 61 ms, consumed after another 7 ms
```

Pass a Cypher script file to a Neo4j Docker container

There are different ways to pass a Cypher script file to a Neo4j Docker container, all of them using the Cypher Shell tool.

- Using the `--file` option of the `cypher-shell` command followed by the file name. After the statements are executed `cypher-shell` shuts down.
- Using the `:source` command followed by the file name when in the Cypher interactive shell.
- Using the commands `cat` or `curl` with `cypher-shell` to pipe the contents of your script file into your container.

Note:

To use the `--file` option or the `:source` command of Cypher Shell, the Cypher script file must be readable from inside the container, otherwise `cypher-shell` will not be able to open the file. The folder containing the examples must be mounted to the container when the container is started.

The following are syntax examples of how to use these commands:

example.cypher script

```
match (n:Actors)-[r]->(m:Movies) return n.name AS Actors, m.title AS Movies, m.year AS MovieYear;
```

Invoke `cypher-shell` with the `--file` option

```
# Put the example.cypher file in the local folder ./examples.

# Start a Neo4j container and mount the ./examples folder inside the container:

docker run --rm \
```

```
--volume /path/to/local/examples:/examples \
--publish=7474:7474 \
--publish=7687:7687 \
--env NEO4J_AUTH=neo4j/your_password \
neo4j:5.26.25

# Run the Cypher Shell tool with the --file option passing the example.cypher file:

docker exec --interactive --tty <containerID/name> cypher-shell -u neo4j -p <password> --file
/examples/example.cypher
```

Use the `:source` command to run a Cypher script file

```
# Put the example.cypher file in the local folder ./examples.

# Start a Neo4j container and mount the ./examples folder inside the container:

docker run --rm \
--volume /path/to/local/examples:/examples \
--publish=7474:7474 \
--publish=7687:7687 \
--env NEO4J_AUTH=neo4j/your_password \
neo4j:5.26.25

# Use the container ID or name to get into the container, and then, run the cypher-shell command and
authenticate.

docker exec --interactive --tty <containerID/name> cypher-shell -u neo4j -p <password>

# Invoke the :source command followed by the file name.

neo4j@neo4j> :source example.cypher
```

Invoke `curl` with Cypher Shell

```
curl http://mysite.com/config/example.cypher | sudo docker exec --interactive <containerID/name> cypher-
shell -u neo4j -p <password>
```

Invoke `cat` with Cypher Shell

```
cat example.cypher | sudo  docker exec --interactive  <containerID/name> cypher-shell -u neo4j -p
<password>
```

Example output

```
Actors, Movies, MovieYear
"Keanu Reeves", "The Matrix Revolutions", 2003
"Keanu Reeves", "The Matrix Reloaded", 2003
"Keanu Reeves", "The Matrix", 1999
"Laurence Fishburne", "The Matrix Revolutions", 2003
"Laurence Fishburne", "The Matrix Reloaded", 2003
"Laurence Fishburne", "The Matrix", 1999
"Carrie-Anne Moss", "The Matrix Revolutions", 2003
"Carrie-Anne Moss", "The Matrix Reloaded", 2003
"Carrie-Anne Moss", "The Matrix", 1999
```

These commands take the contents of the script file and pass it into the Docker container using Cypher Shell. Then, they run a Cypher example, `LOAD CSV` dataset, which might be hosted somewhere on a server (with `curl`), create indexes, constraints, or do other administrative operations.

Dump and load a Neo4j database (offline)

The `neo4j-admin database dump` and `neo4j-admin database load` commands can be run locally to dump

and load an offline database.

The following are examples of how to dump and load the default `neo4j` database. Because these commands are run on a stopped database, you have to launch a container for each operation (dump and load), with the `--rm` flag.

Example 10. Invoke `neo4j-admin database dump` to dump your database.

```
docker run --interactive --tty --rm \
   --volume=$HOME/neo4j/data:/data \
   --volume=$HOME/neo4j/backups:/backups \
   neo4j/neo4j-admin:5.26.25 \
neo4j-admin database dump neo4j --to-path=/backups
```

The volume that contains the database that you want to dump.

The volume that will be used for the dumped database.

Example 11. Invoke `neo4j-admin database load` to load your data into the new database.

```
docker run --interactive --tty --rm \
    --volume=$HOME/neo4j/newdata:/data \
    --volume=$HOME/neo4j/backups:/backups \
    neo4j/neo4j-admin:5.26.25 \
neo4j-admin database load neo4j --from-path=/backups
```

The volume that will contain the database, into which you want to load the dumped data.

The volume that stores the database dump.

Finally, you launch a container with the volume that contains the newly loaded database, and start using it.

Launching a container from restored data

```
docker run --interactive --tty --rm \
    --volume=$HOME/neo4j/newdata:/data \
    neo4j:5.26.25
```

The volume containing the restored data

For more information on the `neo4j-admin database dump and load` syntax and options, see `neo4j-admin database dump` and `neo4j-admin database load`.
For more information on managing volumes, see the official Docker documentation.

Back up and restore a Neo4j database (online)

The Neo4j backup and restore commands can be run locally to backup and restore a live database.

You can also get a `neo4j-admin` image that can be run on a dedicated machine, under the terms of an

existing Enterprise licensing agreement.

If Neo4j (a single instance or any member of a Neo4j cluster) is running inside a Docker container, you can use `docker exec` to invoke `neo4-admin` from inside the container and take a backup of a database.

Back up a database using `docker exec`

To back up a database, you must first mount the host backup folder onto the container. Because Docker does not allow new mounts to be added to a running container, you have to do this when starting the container.

Example 12. A `docker run` command that mounts the host backup folder to a Neo4j container.

```
docker run --name <container name> \
    --detach \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j-enterprise/data:/data \
    --volume=$HOME/neo4j-enterprise/backups:/backups \
    --user="$(id -u):$(id -g)" \
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
    --env NEO4J_server_backup_enabled=true \
    neo4j:5.26.25-enterprise
```

The volume that contains the database that you want to back up.

The volume that will be used for the database backup.

The environment variable that accepts the Neo4j Enterprise Edition license agreement.

The environment variable that enables online backups.

Example 13. Invoke `neo4j-admin database backup` to back up an online database, using `docker exec`:

```
docker exec --interactive --tty <container name> neo4j-admin database backup --to-path=/backups
<database name>
```

Note:

For more information on the `neo4j-admin database backup` syntax and options, see Back up an online database.

Back up a database using `neo4j-admin` image

To perform a backup, the cluster needs at least one server with backup enabled and the backup listen address port set and exposed. Ports cannot be exposed on a Docker container once it has started, so this must be done when starting the container.

Example 14. A `docker run` command that starts a database configured for backing up.

```
docker run \
```

```
    --detach \
    --publish=7474:7474 \
    --publish=7687:7687 \
    --publish=6362:6362 \
    --volume=$HOME/neo4j-enterprise/data:/data \
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
    --env NEO4J_server_backup_enabled=true \
    --env NEO4J_server_backup_listen__address=0.0.0.0:6362 \
    neo4j:5.26.25-enterprise
```

The server.backup.listen_address port defined in 5.

The volume that contains the database that you want to back up.

The environment variable that accepts the Neo4j Enterprise Edition license agreement.

The environment variable that enables online backups.

The environment variable that sets the server.backup.listen_address.

Once you have a backup enabled cluster node, the `neo4j/neo4j-admin:5.26.25-enterprise` docker image can be used to backup the database.

Example 15. Invoke `neo4j-admin` docker image to back up your database.

```
docker run --interactive --tty --rm \
   --volume=$HOME/neo4j-enterprise/backups:/backups \
   --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
   neo4j/neo4j-admin:5.26.25-enterprise \
      neo4j-admin database backup <database name> \
         --to-path=/backups \
         --from=<backup node IP address>:6362
```

The volume that will be used for the backup database files.

The environment variable that accepts the Neo4j Enterprise Edition license agreement.

The IP address of the backup cluster node and the server.backup.listen_address port.

Restore a database using `docker exec`

The following are examples of how to restore a database backup on a stopped database in a running Neo4j instance.

Example 16. A `docker run` command that creates a container to be used for restoring a database backup.

```
docker run --name <container name> \
    --detach \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j-enterprise/data:/data \
    --volume=$HOME/neo4j-enterprise/backups:/backups \
    --user="$(id -u):$(id -g)" \
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
    neo4j:5.26.25-enterprise
```

The volume that contains all your databases.

The volume that contains the database backup.

The environment variable that accepts the Neo4j Enterprise Edition license agreement.

Example 17. Invoke `cypher-shell` to stop the database that you want to use for the backup restore.

```
docker exec -it <containerID/name> cypher-shell -u neo4j -p <my-password> -d system "stop database
<database name>;"
```

Example 18. Invoke `neo4j-admin database restore` to restore a database backup.

```
docker exec --interactive --tty <containerID/name> neo4j-admin database restore --from
=/backups/<databasename>-<timestamp>.backup --overwrite-destination <database name>
```

Restore a database using `neo4j-admin` image

The `neo4j-admin database restore` action cannot be performed remotely, as it requires access to the neo4j */data* folder. Consequently, backup files must be copied over to the new machine prior to a restore, and the `neo4j-admin` docker image must be run on the same machine as the database to be restored.

Example 19. A `docker run` command that creates a container to be used for restoring a database backup.

```
docker run --name <container name> \
    --detach \
    --volume=$HOME/neo4j-enterprise/data:/data \
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
    neo4j:5.26.25-enterprise
```

The volume that contains, or will contain, all your database data.

The environment variable accepts the Neo4j Enterprise Edition license agreement.

Example 20. Stop the old database, then restore the backup database using `neo4j/neo4j-admin:5.26.25-enterprise`. Finally start the database again containing the new data.

```
docker exec -it <containerID/name> cypher-shell -u neo4j -p <my-password> -d system "stop database
<database name>;"
```

```
docker run --interactive --tty --rm \
   --volume=$HOME/neo4j-enterprise/data:/data \
   --volume=$HOME/neo4j-enterprise/backups:/backups \
   --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
   neo4j/neo4j-admin:{neo4j-version-exact}-enterprise \
      neo4j-admin database restore \
         --from=/backups/<databasename>-<timestamp>.backup \
         --overwrite-destination \
          <database name> \
```

```
docker exec -it <containerID/name> cypher-shell -u neo4j -p <my-password> -d system "start database
<database name>;"
```

The volume that contains, or will contain, all your database data. This must be the same data folder that the Neo4j DBMS container is using.

The volume that contains the database backup.

The environment variable that accepts the Neo4j Enterprise Edition license agreement.

Note:

For more information on the `neo4j-admin database restore` syntax and options, see Restore a database backup.

Finally, you can use the Cypher Shell tool to verify that your data has been restored.

SSL encryption in a Neo4j Docker container

Neo4j on Docker supports Neo4j's native SSL Framework for setting up secure Bolt and HTTPS communications. To configure these settings in Docker, you either set them in the *neo4j.conf* file, or pass them to Docker as Docker environment variables.

Set up your certificate folders

1. Verify that you have SSL public certificate(s) and private key(s).

   The certificates must be issued by a trusted certificate authority (CA), such as https://www.openssl.org/ or https://letsencrypt.org/.

   The default file names are *private.key* and *public.crt*.

2. Create a local folder to store your certificates.

   For example, *$HOME/neo4j/certificates*. This folder will be later mounted to */ssl* of your container.

3. In your local folder (e.g. *$HOME/neo4j/certificates*), create a folder for the SSL policy of each of your communication channels that you want to secure. There, you will store your certificates and private keys.

   It is recommended to use different certificates for the different communication channels (`bolt` and `https`).

   In the following examples, `<scope>` substitutes the name of the communication channel.

   ```
   $ mkdir $HOME/neo4j/certificates/<scope>
   ```

4. In each of your <scope> folders, create a */trusted* and a */revoked* folder for the trusted and revoked certificates.

   ```
   $ mkdir $HOME/neo4j/certificates/<scope>/trusted
   $ mkdir $HOME/neo4j/certificates/<scope>/revoked
   ```

5. Finally, you add your certificates to the respective <scope> folder.

   The <scope> folder(s) should now show the following listings:

```
$ ls $HOME/neo4j/certificates/<scope>
-r-------- ... private.key
-rw-r--r-- ... public.crt
drwxr-xr-x ... revoked
drwxr-xr-x ... trusted
```

Configure SSL via *neo4j.conf*

In the *neo4j.conf* file, configure the following settings for the policies that you want to use:

```
# Https SSL configuration
server.https.enabled=true
dbms.ssl.policy.https.enabled=true
dbms.ssl.policy.https.base_directory=certificates/https
dbms.ssl.policy.https.private_key=private.key
dbms.ssl.policy.https.public_certificate=public.crt


# Bolt SSL configuration
dbms.ssl.policy.bolt.enabled=true
dbms.ssl.policy.bolt.base_directory=certificates/bolt
dbms.ssl.policy.bolt.private_key=private.key
dbms.ssl.policy.bolt.public_certificate=public.crt
```

Note:

For more information on configuring SSL policies, see Configuration.

For more information on configuring connectors, see Configuration options.

Example 21. A `docker run` command that launches a container with SSL policy enabled via neo4j.conf.

```
docker run \
    --publish=7473:7473 \
    --publish=7687:7687 \
    --user="$(id -u):$(id -g)" \
    --volume=$HOME/neo4j/certificates:/ssl \
    --volume=$HOME/neo4j/conf:/conf \
    neo4j:5.26.25
```

The port to access the HTTPS endpoint.

Docker will be started as the current user (assuming the current user has read access to the certificates).

The volume that contains the SSL policies that you want to set up Neo4j to use.

The volume that contains the *neo4j.conf* file. In this example, the *neo4j.conf* is in the `$HOME/neo4j/conf` folder of the host.

Configure SSL via Docker environment variables

As an alternative to configuring SSL via the *neo4j.conf* file, you can set an SSL policy by passing its configuration values to the Neo4j Docker container as environment variables. For more information on how to convert the Neo4j settings to the form accepted by Docker, see Environment variables:

Example 22. A `docker run` command that launches a container with SSL policy enabled via Docker environment variables.

```
docker run \
    --publish=7473:7473 \
    --publish=7687:7687 \
    --user="$(id -u):$(id -g)" \
    --volume=$HOME/neo4j/certificates:/ssl \
    --env NEO4J_dbms_connector_https_enabled=true \
    --env NEO4J_dbms_ssl_policy_https_enabled=true \
    --env NEO4J_dbms_ssl_policy_https_base__directory=/ssl/https \
    neo4j:5.26.25
```

The port to access the HTTPS endpoint.

Docker will be started as the current user (assuming the current user has read access to the certificates).

The volume that contains the SSL policies that you want to set up Neo4j to use.

The HTTPS connector is disabled by default. Therefore, you must set `server.https.enabled` to `true`, for Neo4j to be able to listen for incoming connections on the HTTPS port. However, for the Bolt SSL policy, you do not have to pass this parameter as the Bolt connector is enabled by default.

The SSL policy that you want to set up for Neo4j.

The base directory under which SSL certificates and keys are searched for. Note that the value is the docker volume folder */ssl/https* and not the */certificate/https* folder of the host.

Docker-specific configuration settings

The Neo4j configuration settings can be passed to a Docker container using the following naming scheme:

- Prefix with `NEO4J_`.
- Underscores convert to double underscores: `_` is written as `__`.
- Periods convert to underscores: `.` is written as `_`.

For example, `browser.post_connect_cmd` converts to `NEO4J_browser_post__connect__cmd`, or in other words, `s/\./_/g` and `s/_/__/g`.

The following table is a reference of the Neo4j configuration settings converted to the Docker-supported format.

For more information on the configuration descriptions, valid values, and default values, see Configuration settings.

Neo4j format	Docker format
`browser.allow_outgoing_connections`	`NEO4J_browser_allow__outgoing__connections`

Neo4j format	Docker format
`browser.credential_timeout`	`NEO4J_browser_credential__timeout`
`browser.post_connect_cmd`	`NEO4J_browser_post__connect__cmd`
`browser.remote_content_hostname_whitelist`	`NEO4J_browser_remote__content__hostname__whitelist`
`browser.retain_connection_credentials`	`NEO4J_browser_retain__connection__credentials`
`browser.retain_editor_history`	`NEO4J_browser_retain__editor__history`
`client.allow_telemetry`	`NEO4J_client_allow__telemetry`
`db.checkpoint`	`NEO4J_db_checkpoint`
`db.checkpoint.interval.time`	`NEO4J_db_checkpoint_interval_time`
`db.checkpoint.interval.tx`	`NEO4J_db_checkpoint_interval_tx`
`db.checkpoint.interval.volume`	`NEO4J_db_checkpoint_interval_volume`
`db.checkpoint.iops.limit`	`NEO4J_db_checkpoint_iops_limit`
`db.cluster.catchup.pull_interval`	`NEO4J_db_cluster_catchup_pull__interval`
`db.cluster.raft.apply.buffer.max_bytes`	`NEO4J_db_cluster_raft_apply_buffer_max__bytes`
`db.cluster.raft.apply.buffer.max_entries`	`NEO4J_db_cluster_raft_apply_buffer_max__entries`
`db.cluster.raft.in_queue.batch.max_bytes`	`NEO4J_db_cluster_raft_in__queue_batch_max__byte`
`db.cluster.raft.in_queue.max_bytes`	`NEO4J_db_cluster_raft_in__queue_max__bytes`
`db.cluster.raft.leader_transfer.priority_group`	`NEO4J_db_cluster_raft_leader__transfer_priority__group`
`db.cluster.raft.log.prune_strategy`	`NEO4J_db_cluster_raft_log_prune__strategy`
`db.cluster.raft.log_shipping.buffer.max_bytes`	`NEO4J_db_cluster_raft_log__shipping_buffer_max__bytes`
`db.cluster.raft.log_shipping.buffer.max_entries`	`NEO4J_db_cluster_raft_log__shipping_buffer_max__entrie s`
`db.filewatcher.enabled`	`NEO4J_db_filewatcher_enabled`
`db.format`	`NEO4J_db_format`
`db.import.csv.buffer_size`	`NEO4J_db_import_csv_buffer__size`
`db.import.csv.legacy_quote_escaping`	`NEO4J_db_import_csv_legacy__quote__escaping`
`db.index.fulltext.default_analyzer`	`NEO4J_db_index_fulltext_default__analyzer`
`db.index.fulltext.eventually_consistent`	`NEO4J_db_index_fulltext_eventually__consistent`
`db.index.fulltext.eventually_consistent_index_update_q ueue_max_length`	`NEO4J_db_index_fulltext_eventually__consistent__index_ _update__queue__max__length`
`db.index_sampling.background_enabled`	`NEO4J_db_index__sampling_background__enabled`
`db.index_sampling.sample_size_limit`	`NEO4J_db_index__sampling_sample__size__limit`
`db.index_sampling.update_percentage`	`NEO4J_db_index__sampling_update__percentage`

Neo4j format	Docker format
`db.lock.acquisition.timeout`	`NEO4J_db_lock_acquisition_timeout`
`db.logs.query.annotation_data_as_json_enabled`	`NEO4J_db_logs_query_annotation__data__as__json__enabled`
`db.logs.query.annotation_data_format`	`NEO4J_db_logs_query_annotation__data__format`
`db.logs.query.early_raw_logging_enabled`	`NEO4J_db_logs_query_early__raw__logging__enabled`
`db.logs.query.enabled`	`NEO4J_db_logs_query_enabled`
`db.logs.query.max_parameter_length`	`NEO4J_db_logs_query_max__parameter__length`
`db.logs.query.obfuscate_literals`	`NEO4J_db_logs_query_obfuscate__literals`
`db.logs.query.parameter_logging_enabled`	`NEO4J_db_logs_query_parameter__logging__enabled`
`db.logs.query.plan_description_enabled`	`NEO4J_db_logs_query_plan__description__enabled`
`db.logs.query.threshold`	`NEO4J_db_logs_query_threshold`
`db.logs.query.transaction.enabled`	`NEO4J_db_logs_query_transaction_enabled`
`db.logs.query.transaction.threshold`	`NEO4J_db_logs_query_transaction_threshold`
`db.memory.pagecache.warmup.enable`	`NEO4J_db_memory_pagecache_warmup_enable`
`db.memory.pagecache.warmup.preload`	`NEO4J_db_memory_pagecache_warmup_preload`
`db.memory.pagecache.warmup.preload.allowlist`	`NEO4J_db_memory_pagecache_warmup_preload_allowlist`
`db.memory.pagecache.warmup.profile.interval`	`NEO4J_db_memory_pagecache_warmup_profile_interval`
`db.memory.transaction.max`	`NEO4J_db_memory_transaction_max`
`db.memory.transaction.total.max`	`NEO4J_db_memory_transaction_total_max`
`db.recovery.fail_on_missing_files`	`NEO4J_db_recovery_fail__on__missing__files`
`db.relationship_grouping_threshold`	`NEO4J_db_relationship__grouping__threshold`
`db.shutdown_transaction_end_timeout`	`NEO4J_db_shutdown__transaction__end__timeout`
`db.store.files.preallocate`	`NEO4J_db_store_files_preallocate`
`db.temporal.timezone`	`NEO4J_db_temporal_timezone`
`db.track_query_cpu_time`	`NEO4J_db_track__query__cpu__time`
`db.transaction.bookmark_ready_timeout`	`NEO4J_db_transaction_bookmark__ready__timeout`
`db.transaction.concurrent.maximum`	`NEO4J_db_transaction_concurrent_maximum`
`db.transaction.monitor.check.interval`	`NEO4J_db_transaction_monitor_check_interval`
`db.transaction.sampling.percentage`	`NEO4J_db_transaction_sampling_percentage`
`db.transaction.timeout`	`NEO4J_db_transaction_timeout`
`db.transaction.tracing.level`	`NEO4J_db_transaction_tracing_level`
`db.tx_log.buffer.size`	`NEO4J_db_tx_log_buffer_size`

Neo4j format	Docker format
db.tx_log.preallocate	NEO4J_db_tx__log_preallocate
db.tx_log.rotation.retention_policy	NEO4J_db_tx__log_rotation_retention__policy
db.tx_log.rotation.size	NEO4J_db_tx__log_rotation_size
db.tx_state.memory_allocation	NEO4J_db_tx__state_memory__allocation
dbms.cluster.catchup.client_inactivity_timeout	NEO4J_dbms_cluster_catchup_client__inactivity__timeout
dbms.cluster.discovery.endpoints	NEO4J_dbms_cluster_discovery_endpoints
dbms.cluster.discovery.log_level	NEO4J_dbms_cluster_discovery_log__level
dbms.cluster.discovery.resolver_type	NEO4J_dbms_cluster_discovery_resolver__type
dbms.cluster.minimum_initial_system_primaries_count	NEO4J_dbms_cluster_minimum__initial__system__primaries__count
dbms.cluster.network.handshake_timeout	NEO4J_dbms_cluster_network_handshake__timeout
dbms.cluster.network.max_chunk_size	NEO4J_dbms_cluster_network_max__chunk__size
dbms.cluster.network.supported_compression_algos	NEO4J_dbms_cluster_network_supported__compression__algos
dbms.cluster.raft.binding_timeout	NEO4J_dbms_cluster_raft_binding__timeout
dbms.cluster.raft.client.max_channels	NEO4J_dbms_cluster_raft_client_max__channels
dbms.cluster.raft.election_failure_detection_window	NEO4J_dbms_cluster_raft_election__failure__detection__window
dbms.cluster.raft.leader_failure_detection_window	NEO4J_dbms_cluster_raft_leader__failure__detection__window
dbms.cluster.raft.leader_transfer.balancing_strategy	NEO4J_dbms_cluster_raft_leader__transfer_balancing__strategy
dbms.cluster.raft.log.pruning_frequency	NEO4J_dbms_cluster_raft_log_pruning__frequency
dbms.cluster.raft.log.reader_pool_size	NEO4J_dbms_cluster_raft_log_reader__pool__size
dbms.cluster.raft.log.rotation_size	NEO4J_dbms_cluster_raft_log_rotation__size
dbms.cluster.raft.membership.join_max_lag	NEO4J_dbms_cluster_raft_membership_join__max__lag
dbms.cluster.raft.membership.join_timeout	NEO4J_dbms_cluster_raft_membership_join__timeout
dbms.cluster.store_copy.max_retry_time_per_request	NEO4J_dbms_cluster_store__copy_max__retry__time__per__request
dbms.cypher.forbid_exhaustive_shortestpath	NEO4J_dbms_cypher_forbid__exhaustive__shortestpath
dbms.cypher.hints_error	NEO4J_dbms_cypher_hints__error
dbms.cypher.lenient_create_relationship	NEO4J_dbms_cypher_lenient__create__relationship
dbms.cypher.min_replan_interval	NEO4J_dbms_cypher_min__replan__interval
dbms.cypher.planner	NEO4J_dbms_cypher_planner

Neo4j format	Docker format
dbms.cypher.render_plan_description	NEO4J_dbms_cypher_render__plan__description
dbms.cypher.statistics_divergence_threshold	NEO4J_dbms_cypher_statistics__divergence__threshold
dbms.databases.seed_from_uri_providers	NEO4J_dbms_databases_seed__from__uri__providers
dbms.db.timezone	NEO4J_dbms_db_timezone
dbms.kubernetes.address	NEO4J_dbms_kubernetes_address
dbms.kubernetes.ca_crt	NEO4J_dbms_kubernetes_ca__crt
dbms.kubernetes.cluster_domain	NEO4J_dbms_kubernetes_cluster__domain
dbms.kubernetes.label_selector	NEO4J_dbms_kubernetes_label__selector
dbms.kubernetes.namespace	NEO4J_dbms_kubernetes_namespace
dbms.kubernetes.service_port_name	NEO4J_dbms_kubernetes_service__port__name
dbms.kubernetes.token	NEO4J_dbms_kubernetes_token
dbms.logs.http.enabled	NEO4J_dbms_logs_http_enabled
db.lock.acquisition.timeout	NEO4J_dbms_lock_acquisition_timeout
server.logs.gc.enabled	NEO4J_server_logs_gc_enabled
server.logs.gc.options	NEO4J_server_logs_gc_options
server.logs.gc.rotation.keep_number	NEO4J_server_logs_gc_rotation_keep__number
server.logs.gc.rotation.size	NEO4J_server_logs_gc_rotation_size
dbms.logs.http.enabled	NEO4J_dbms_logs_http_enabled
dbms.max_databases	NEO4J_dbms_max__databases
dbms.memory.tracking.enable	NEO4J_dbms_memory_tracking_enable
dbms.memory.transaction.total.max	NEO4J_dbms_memory_transaction_total_max
dbms.netty.ssl.provider	NEO4J_dbms_netty_ssl_provider
dbms.routing.client_side.enforce_for_domains	NEO4J_dbms_routing_client__side_enforce__for__domains
dbms.routing.default_router	NEO4J_dbms_routing_default__router
dbms.routing.driver.connection.connect_timeout	NEO4J_dbms_routing_driver_connection_connect__timeout
dbms.routing.driver.connection.max_lifetime	NEO4J_dbms_routing_driver_connection_max__lifetime
dbms.routing.driver.connection.pool.acquisition_timeout	NEO4J_dbms_routing_driver_connection_pool_acquisition__timeout
dbms.routing.driver.connection.pool.idle_test	NEO4J_dbms_routing_driver_connection_pool_idle__test
dbms.routing.driver.connection.pool.max_size	NEO4J_dbms_routing_driver_connection_pool_max__size
dbms.routing.driver.logging.level	NEO4J_dbms_routing_driver_logging_level
dbms.routing.enabled	NEO4J_dbms_routing_enabled

Neo4j format	Docker format
`dbms.routing.load_balancing.plugin`	`NEO4J_dbms_routing_load__balancing_plugin`
`dbms.routing.load_balancing.shuffle_enabled`	`NEO4J_dbms_routing_load__balancing_shuffle__enabled`
`dbms.routing.reads_on_primaries_enabled`	`NEO4J_dbms_routing_reads__on__primaries__enabled`
`dbms.routing.reads_on_writers_enabled`	`NEO4J_dbms_routing_reads__on__writers__enabled`
`dbms.routing_ttl`	`NEO4J_dbms_routing__ttl`
`dbms.security.allow_csv_import_from_file_urls`	`NEO4J_dbms_security_allow__csv__import__from__file__urls`
`dbms.security.auth_cache_max_capacity`	`NEO4J_dbms_security_auth__cache__max__capacity`
`dbms.security.auth_cache_ttl`	`NEO4J_dbms_security_auth__cache__ttl`
`dbms.security.auth_cache_use_ttl`	`NEO4J_dbms_security_auth__cache__use__ttl`
`dbms.security.auth_enabled`	`NEO4J_dbms_security_auth__enabled`
`dbms.security.auth_lock_time`	`NEO4J_dbms_security_auth__lock__time`
`dbms.security.auth_max_failed_attempts`	`NEO4J_dbms_security_auth__max__failed__attempts`
`dbms.security.authentication_providers`	`NEO4J_dbms_security_authentication__providers`
`dbms.security.authorization_providers`	`NEO4J_dbms_security_authorization__providers`
`dbms.security.cluster_status_auth_enabled`	`NEO4J_dbms_security_cluster__status__auth__enabled`
`dbms.security.http_access_control_allow_origin`	`NEO4J_dbms_security_http__access__control__allow_origin`
`dbms.security.http_auth_allowlist`	`NEO4J_dbms_security_http__auth__allowlist`
`dbms.security.http_strict_transport_security`	`NEO4J_dbms_security_http__strict__transport__security`
`dbms.security.key.name`	`NEO4J_dbms_security_key_name`
`dbms.security.keystore.password`	`NEO4J_dbms_security_keystore_password`
`dbms.security.keystore.path`	`NEO4J_dbms_security_keystore_path`
`dbms.security.ldap.authentication.attribute`	`NEO4J_dbms_security_ldap_authentication_attribute`
`dbms.security.ldap.authentication.cache_enabled`	`NEO4J_dbms_security_ldap_authentication_cache__enabled`
`dbms.security.ldap.authentication.mechanism`	`NEO4J_dbms_security_ldap_authentication_mechanism`
`dbms.security.ldap.authentication.search_for_attribute`	`NEO4J_dbms_security_ldap_authentication_search__for__attribute`
`dbms.security.ldap.authentication.user_dn_template`	`NEO4J_dbms_security_ldap_authentication_user__dn__template`
`dbms.security.ldap.authorization.access_permitted_group`	`NEO4J_dbms_security_ldap_authorization_access__permitted__group`
`dbms.security.ldap.authorization.group_membership_attributes`	`NEO4J_dbms_security_ldap_authorization_group__membership__attributes`

Neo4j format	Docker format
dbms.security.ldap.authorization.group_to_role_mapping	NEO4J_dbms_security_ldap_authorization_group__to__role__mapping
dbms.security.ldap.authorization.nested_groups_enabled	NEO4J_dbms_security_ldap_authorization_nested__groups__enabled
dbms.security.ldap.authorization.nested_groups_search_filter	NEO4J_dbms_security_ldap_authorization_nested__groups__search__filter
dbms.security.ldap.authorization.system_password	NEO4J_dbms_security_ldap_authorization_system__password
dbms.security.ldap.authorization.system_username	NEO4J_dbms_security_ldap_authorization_system__username
dbms.security.ldap.authorization.use_system_account	NEO4J_dbms_security_ldap_authorization_use__system__account
dbms.security.ldap.authorization.user_search_base	NEO4J_dbms_security_ldap_authorization_user__search__base
dbms.security.ldap.authorization.user_search_filter	NEO4J_dbms_security_ldap_authorization_user__search__filter
dbms.security.ldap.connection_timeout	NEO4J_dbms_security_ldap_connection__timeout
dbms.security.ldap.host	NEO4J_dbms_security_ldap_host
dbms.security.ldap.read_timeout	NEO4J_dbms_security_ldap_read__timeout
dbms.security.ldap.referral	NEO4J_dbms_security_ldap_referral
dbms.security.ldap.use_starttls	NEO4J_dbms_security_ldap_use__starttls
dbms.security.log_successful_authentication	NEO4J_dbms_security_log__successful__authentication
dbms.security.oidc.<provider>.audience	NEO4J_dbms_security_oidc_<provider>_audience
dbms.security.oidc.<provider>.auth_endpoint	NEO4J_dbms_security_oidc_<provider>_auth__endpoint
dbms.security.oidc.<provider>.auth_flow	NEO4J_dbms_security_oidc_<provider>_auth__flow
dbms.security.oidc.<provider>.auth_params	NEO4J_dbms_security_oidc_<provider>_auth__params
dbms.security.oidc.<provider>.authorization.group_to_role_mapping	NEO4J_dbms_security_oidc_<provider>_authorization_group__to__role__mapping
dbms.security.oidc.<provider>.claims.groups	NEO4J_dbms_security_oidc_<provider>_claims_groups
dbms.security.oidc.<provider>.claims.username	NEO4J_dbms_security_oidc_<provider>_claims_username
dbms.security.oidc.<provider>.client_id	NEO4J_dbms_security_oidc_<provider>_client__id
dbms.security.oidc.<provider>.config	NEO4J_dbms_security_oidc_<provider>_config
dbms.security.oidc.<provider>.display_name	NEO4J_dbms_security_oidc_<provider>_display__name
dbms.security.oidc.<provider>.get_groups_from_user_info	NEO4J_dbms_security_oidc_<provider>_get__groups__from__user__info

Neo4j format	Docker format
dbms.security.oidc.<provider>.get_username_from_user_info	NEO4J_dbms_security_oidc_<provider>_get__username__from__user__info
dbms.security.oidc.<provider>.issuer	NEO4J_dbms_security_oidc_<provider>_issuer
dbms.security.oidc.<provider>.jwks_uri	NEO4J_dbms_security_oidc_<provider>_jwks__uri
dbms.security.oidc.<provider>.params	NEO4J_dbms_security_oidc_<provider>_params
dbms.security.oidc.<provider>.token_endpoint	NEO4J_dbms_security_oidc_<provider>_token__endpoint
dbms.security.oidc.<provider>.token_params	NEO4J_dbms_security_oidc_<provider>_token__params
dbms.security.oidc.<provider>.user_info_uri	NEO4J_dbms_security_oidc_<provider>_user__info__uri
dbms.security.oidc.<provider>.well_known_discovery_uri	NEO4J_dbms_security_oidc_<provider>_well__known__discovery__uri
dbms.security.procedures.allowlist	NEO4J_dbms_security_procedures_allowlist
dbms.security.procedures.unrestricted	NEO4J_dbms_security_procedures_unrestricted
initial.dbms.database_allocator	NEO4J_initial_dbms_database__allocator
initial.dbms.default_database	NEO4J_initial_dbms_default__database
initial.dbms.default_primaries_count	NEO4J_initial_dbms_default__primaries__count
initial.dbms.default_secondaries_count	NEO4J_initial_dbms_default__secondaries__count
initial.server.allowed_databases	NEO4J_initial_server_allowed__databases
initial.server.denied_databases	NEO4J_initial_server_denied__databases
initial.server.mode_constraint	NEO4J_initial_server_mode__constraint
server.backup.enabled	NEO4J_server_backup_enabled
server.backup.listen_address	NEO4J_server_backup_listen__address
server.backup.store_copy_max_retry_time_per_request	NEO4J_server_backup_store__copy__max__retry__time__per__request
server.bolt.advertised_address	NEO4J_server_bolt_advertised__address
server.bolt.connection_keep_alive	NEO4J_server_bolt_connection__keep__alive
server.bolt.connection_keep_alive_for_requests	NEO4J_server_bolt_connection__keep__alive__for__requests
server.bolt.connection_keep_alive_probes	NEO4J_server_bolt_connection__keep__alive__probes
server.bolt.connection_keep_alive_streaming_scheduling_interval	NEO4J_server_bolt_connection__keep__alive__streaming__scheduling__interval
server.bolt.enabled	NEO4J_server_bolt_enabled
server.bolt.listen_address	NEO4J_server_bolt_listen__address
server.bolt.ocsp_stapling_enabled	NEO4J_server_bolt_ocsp__stapling__enabled
server.bolt.thread_pool_keep_alive	NEO4J_server_bolt_thread__pool__keep__alive

Neo4j format	Docker format
server.bolt.thread_pool_max_size	NEO4J_server_bolt_thread__pool__max__size
server.bolt.thread_pool_min_size	NEO4J_server_bolt_thread__pool__min__size
server.bolt.tls_level	NEO4J_server_bolt_tls__level
server.cluster.advertised_address	NEO4J_server_cluster_advertised__address
server.cluster.catchup.connect_randomly_to_server_group	NEO4J_server_cluster_catchup_connect__randomly__to__server__group
server.cluster.catchup.upstream_strategy	NEO4J_server_cluster_catchup_upstream__strategy
server.cluster.catchup.user_defined_upstream_strategy	NEO4J_server_cluster_catchup_user__defined__upstream__strategy
server.cluster.listen_address	NEO4J_server_cluster_listen__address
server.cluster.network.native_transport_enabled	NEO4J_server_cluster_network_native__transport__enabled
server.cluster.raft.advertised_address	NEO4J_server_cluster_raft_advertised__address
server.cluster.raft.listen_address	NEO4J_server_cluster_raft_listen__address
server.cluster.system_database_mode	NEO4J_server_cluster_system__database__mode
server.config.strict_validation.enabled	NEO4J_server_config_strict__validation_enabled
server.databases.default_to_read_only	NEO4J_server_databases_default__to__read__only
server.databases.read_only	NEO4J_server_databases_read__only
server.databases.writable	NEO4J_server_databases_writable
server.db.query_cache_size	NEO4J_server_db_query__cache__size
server.default_advertised_address	NEO4J_server_default__advertised__address
server.default_listen_address	NEO4J_server_default__listen__address
server.directories.cluster_state	NEO4J_server_directories_cluster__state
server.directories.data	NEO4J_server_directories_data
server.directories.dumps.root	NEO4J_server_directories_dumps_root
server.directories.import	NEO4J_server_directories_import
server.directories.lib	NEO4J_server_directories_lib
server.directories.licenses	NEO4J_server_directories_licenses
server.directories.logs	NEO4J_server_directories_logs
server.directories.metrics	NEO4J_server_directories_metrics
server.directories.neo4j_home	NEO4J_server_directories_neo4j__home
server.directories.plugins	NEO4J_server_directories_plugins
server.directories.run	NEO4J_server_directories_run

Neo4j format	Docker format
`server.directories.script.root`	`NEO4J_server_directories_script_root`
`server.directories.transaction.logs.root`	`NEO4J_server_directories_transaction_logs_root`
`server.discovery.advertised_address`	`NEO4J_server_discovery_advertised__address`
`server.discovery.listen_address`	`NEO4J_server_discovery_listen__address`
`server.dynamic.setting.allowlist`	`NEO4J_server_dynamic_setting_allowlist`
`server.groups`	`NEO4J_server_groups`
`server.http.advertised_address`	`NEO4J_server_http_advertised__address`
`server.http.enabled`	`NEO4J_server_http_enabled`
`server.http.listen_address`	`NEO4J_server_http_listen__address`
`server.http_enabled_modules`	`NEO4J_server_http__enabled__modules`
`server.https.advertised_address`	`NEO4J_server_https_advertised__address`
`server.https.enabled`	`NEO4J_server_https_enabled`
`server.https.listen_address`	`NEO4J_server_https_listen__address`
`server.jvm.additional`	`NEO4J_server_jvm_additional`
`server.logs.config`	`NEO4J_server_logs_config`
`server.logs.debug.enabled`	`NEO4J_server_logs_debug_enabled`
`server.logs.gc.enabled`	`NEO4J_server_logs_gc_enabled`
`server.logs.gc.options`	`NEO4J_server_logs_gc_options`
`server.logs.gc.rotation.keep_number`	`NEO4J_server_logs.gc_rotation_keep__number`
`server.logs.gc.rotation.size`	`NEO4J_server_logs_gc_rotation_size`
`server.logs.user.config`	`NEO4J_server_logs_user_config`
`server.max_databases`	`NEO4J_server._max__databases`
`server.memory.heap.initial_size`	`NEO4J_server_memory_heap_initial__size`
`server.memory.heap.max_size`	`NEO4J_server_memory_heap_max__size`
`server.memory.off_heap.block_cache_size`	`NEO4J_server_memory_off__heap_block__cache__size`
`server.memory.off_heap.max_cacheable_block_size`	`NEO4J_server_memory_off__heap_max__cacheable__block__size`
`server.memory.off_heap.max_size`	`NEO4J_server_memory_off__heap_max__size`
`server.memory.pagecache.directio`	`NEO4J_server_memory_pagecache_directio`
`server.memory.pagecache.flush.buffer.enabled`	`NEO4J_server_memory_pagecache_flush_buffer_enabled`
`server.memory.pagecache.flush.buffer.size_in_pages`	`NEO4J_server_memory_pagecache_flush_buffer_size__in__pages`

Neo4j format	Docker format
server.memory.pagecache.scan.prefetchers	NEO4J_server_memory_pagecache_scan_prefetchers
server.memory.pagecache.size	NEO4J_server_memory_pagecache_size
server.metrics.csv.enabled	NEO4J_server_metrics_csv_enabled
server.metrics.csv.interval	NEO4J_server_metrics_csv_interval
server.metrics.csv.rotation.compression	NEO4J_server_metrics_csv_rotation_compression
server.metrics.csv.rotation.keep_number	NEO4J_server_metrics_csv_rotation_keep__number
server.metrics.csv.rotation.size	NEO4J_server_metrics_csv_rotation_size
server.metrics.enabled	NEO4J_server_metrics_enabled
server.metrics.filter	NEO4J_server_metrics_filter
server.metrics.graphite.enabled	NEO4J_server_metrics_graphite_enabled
server.metrics.graphite.interval	NEO4J_server_metrics_graphite_interval
server.metrics.graphite.server	NEO4J_server_metrics_graphite_server
server.metrics.jmx.enabled	NEO4J_server_metrics_jmx_enabled
server.metrics.prefix	NEO4J_server_metrics_prefix
server.metrics.prometheus.enabled	NEO4J_server_metrics_prometheus_enabled
server.metrics.prometheus.endpoint	NEO4J_server_metrics_prometheus_endpoint
server.panic.shutdown_on_panic	NEO4J_server_panic_shutdown__on__panic
server.routing.advertised_address	NEO4J_server_routing_advertised__address
server.routing.listen_address	NEO4J_server_routing_listen__address
server.threads.worker_count	NEO4J_server_threads_worker__count
server.unmanaged_extension_classes	NEO4J_server_unmanaged__extension__classes
server.windows_service_name	NEO4J_server_windows__service__name

[1] Write permissions are required when using the `dump-config` feature.
[2] Write permissions are required when using the `NEO4J_PLUGINS` feature to download and store plugins.

Kubernetes

> **Note:**
>
> The Neo4j Helm charts replace the Labs Helm charts project at https://neo4j.com/labs. This is the recommended way to run Neo4j on Kubernetes. For more information on how to move from the Labs Helm charts to the Neo4j Helm charts, see the Migrate Neo4j from the Labs Helm charts to the Neo4j Helm charts (offline).

This chapter describes the following:

- Introduction — Introduction to running Neo4j on a Kubernetes cluster using Neo4j Helm charts.
- Configure the Neo4j Helm chart repository — Configure the Neo4j Helm chart repository and check for the available charts.
- Quickstart: Deploy a standalone instance — Deploy a Neo4j standalone instance to a cloud (GKE, AWS, AKS) or a local (via Docker Desktop for macOS) Kubernetes cluster.
- Quickstart: Deploy a cluster — Deploy a Neo4j cluster to a cloud (GKE, AWS, AKS) Kubernetes cluster.
- Quickstart: Deploy a Neo4j cluster for analytic queries — Deploy an analytics Neo4j cluster with 1 primary and N secondary servers to a local or a cloud (GKE, AWS, AKS) Kubernetes cluster.
- Volume mounts and persistent volumes — Use persistent volumes with the Neo4j Helm chart and what types Neo4j supports.
- Customizing a Neo4j Helm chart — Configure a Neo4j deployment using a customized *values.yaml* file.
- Configuring SSL — Configure SSL for a Neo4j deployment running on Kubernetes.
- Authentication and authorization — Configure LDAP and SSO for a Neo4j deployment running on Kubernetes.
- Plugins - Configure APOC, Bloom or GDS plugins for a Neo4j deployment running on Kubernetes.
- Accessing Neo4j — Access Neo4j running on Kubernetes.
- Accessing Neo4j using Kubernetes Ingress — Access Neo4j using Kubernetes Ingress via Reverse-Proxy Helm chart.
- Importing data — Import data into a Neo4j database.
- Monitoring — Monitor a Neo4j deployment running on Kubernetes.
- Operations — Perform operations on a Neo4j deployment running on Kubernetes.
    - Maintenance mode
    - Reset the neo4j user password
    - Restart a Neo4j pod after a configuration change
    - Dump and load databases (offline)
    - Back up and restore a single database (online)
    - Upgrade Neo4j on Kubernetes
    -

Introduction

Neo4j supports both a standalone and a cluster deployment of Neo4j on Kubernetes using the Neo4j Helm charts.

Tip:

Helm (https://helm.sh/) is a “package manager for Kubernetes”. It usually runs on a machine outside of Kubernetes and creates resources in Kubernetes by calling the Kubernetes API. Helm installs and manages applications on Kubernetes using *Helm charts*, which are distributed via *Helm chart repositories*.

The Neo4j Helm chart repository

The Neo4j Helm chart repository contains a helm chart for the Neo4j standalone server and cluster installations (*neo4j/neo4j*), and support charts to simplify configuration and operations. For more details on how to configure the Neo4j Helm chart repository, see Configure the Neo4j Helm chart repository. The source code of the Neo4j Helm charts is licensed under **Apache License 2.0.**

Using the Neo4j Helm chart repository

When using the Neo4j Helm chart, you are responsible for defining *values.yaml* files. The YAML files specify what you want to achieve with the Helm chart and the Neo4j configuration. There is no *neo4j.conf* file in this setup.

Then, you run `helm install` selecting the chart to install and passing in the *values.yaml* file to customize the behavior. The Helm chart creates Kubernetes entities, which in some cases also spawn outside the resources in the cloud environment where they are run (e.g., cloud load balancers).

For more information about the Helm chart and the Kubernetes and Cloud resources they instantiate when installed, see Neo4j Helm chart for standalone server deployment and Neo4j Helm chart for cluster deployments.

Configuring the Neo4j Helm chart repository

To deploy a Neo4j DBMS or cluster on Kubernetes, you have to configure the Neo4j Helm chart repository.

Prerequisites

- Helm v3 (https://helm.sh).

Configure the Neo4j Helm chart repository

1. Add the Neo4j Helm chart repository.

```
helm repo add neo4j https://helm.neo4j.com/neo4j
```

2. Update the repository:

```
helm repo update
```

Check for the available Neo4j Helm charts

```
helm search repo neo4j/ --versions | grep 5.26.25
```

The output should be similar to the following:

```
neo4j/neo4j                       5.26.25  5.26.25  Neo4j is the world's leading graph database
neo4j/neo4j-admin                 5.26.25  5.26.25  Neo4j is the world's leading graph database
neo4j/neo4j-headless-service      5.26.25  -       Neo4j is the world's leading graph database
neo4j/neo4j-persistent-volume     5.26.25  -       Sets up persistent disks suitable for a Neo4j H...
neo4j/neo4j-reverse-proxy         5.26.25  5.26.25  Sets up an http server and a reverse proxy for ...
```

If you want to see all the versions available, use the option `--versions`.

The utility Helm charts *neo4j/neo4j-docker-desktop-pv* and *neo4j/neo4j-persistent-volume* can be used as an alternative way of creating persistent volumes in those environments.

Quickstart: Deploy a standalone instance

This quickstart guide walks through the basics of deploying a Neo4j standalone instance to a cloud or a local Kubernetes cluster using the Neo4j Helm chart.

The quickstart for deploying a Neo4j standalone server contains the following:

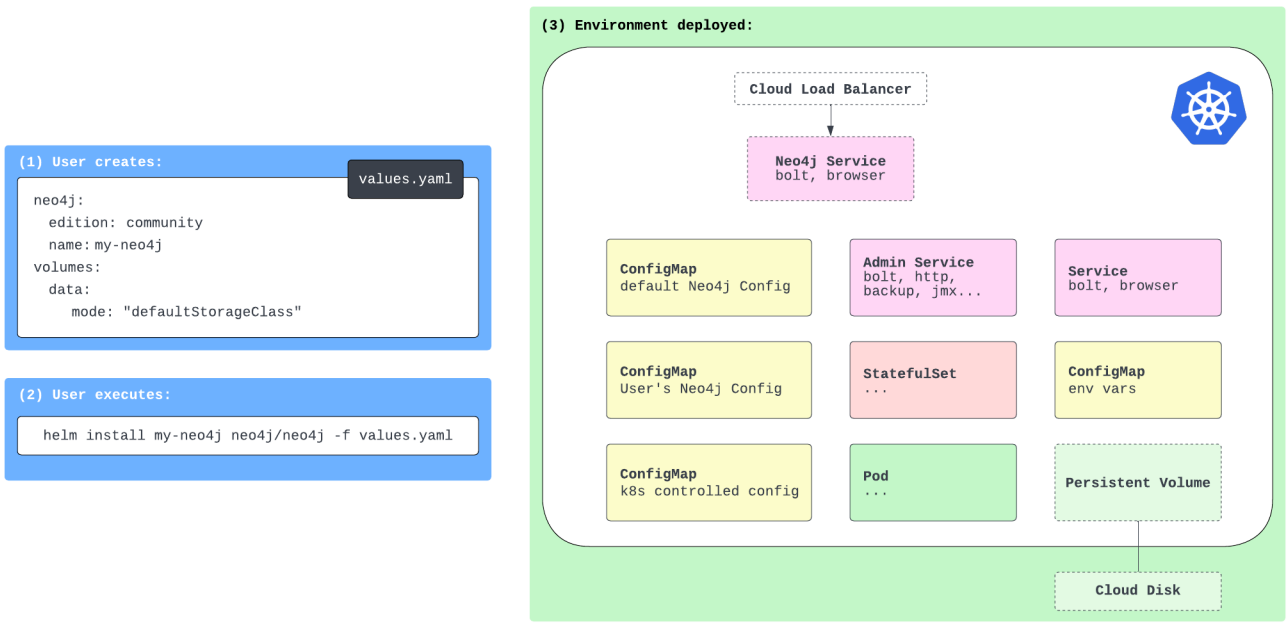
- Neo4j Helm chart for standalone server deployments — A schematic representation of how to use the Neo4j Helm chart for deploying a standalone server.
- Prerequisites — Set up your environment for deploying a Neo4j standalone instance on Kubernetes.
- Create a Helm deployment *values.yaml* file — Create a Helm deployment *values.yaml* file with all Neo4j configurations.
- Install a Neo4j standalone instance — Install a Neo4j standalone instance using the deployment *values.yaml* file and the *neo4j/neo4j* Helm chart.
- Verify the installation — Verify the Neo4j installation.
- Uninstall Neo4j and clean up the created resources — Uninstall Neo4j and clean up the created resources.

Neo4j Helm chart for standalone server deployments

In the standalone server setup, the user is responsible for defining a single YAML file, containing all the configurations for the Neo4j standalone instance, and for running the `helm install my-neo4j-release neo4j/neo4j -f values.yaml` command to deploy the Neo4j DBMS. Then, the Neo4j Helm chart creates Kubernetes entities needed for running and accessing Neo4j.

The following diagram is a schematic representation of the Helm chart and the Kubernetes and Cloud resources it instantiates when installed:

Neo4j standalone server setup



Prerequisites

Before you can deploy a Neo4j standalone instance on Kubernetes, you need to:

General prerequisites

- Configure the Neo4j Helm chart repository.
- Obtain a valid license if you want to install Neo4j Enterprise Edition or set the configuration parameters `edition: "enterprise"` to `"eval"` if you want to accept the Neo4j evaluation license. The Neo4j Community Edition (default for the standalone chart) does not require a license.
  For more information on how to obtain a commercial license, see https://neo4j.com/licensing/ and link:https://neo4j.com/terms/licensing/, or use the form Contact Neo4j.
- Install the Kubernetes client command-line tool `kubectl` (https://kubernetes.io/docs/tasks/tools/).
- Set up a Kubernetes cluster with sufficient CPU and memory for your Neo4j deployment.

> **Note:**
>
> This guide works with minimum CPU and memory allocated per Neo4j instance.

However, the Neo4j system requirements largely depend on the use of the software. Therefore, for running Neo4j in development or production environments, please refer to System requirements.

If you do not have a Kubernetes cluster, you can configure a single-node one as per your environment, see the next section Environment-specific prerequisites.

Environment-specific prerequisites

Select the tab as per your Kubernetes environment and complete all prerequisites on it.

1. Install the `gcloud` command-line interface (CLI) (https://cloud.google.com/sdk/docs/install).
2. All the shell commands in this guide assume that the GCP Project, compute zone, and region to use have been set using the `CLOUDSDK_CORE_PROJECT`, `CLOUDSDK_COMPUTE_ZONE`, and `CLOUDSDK_COMPUTE_REGION` environment variables, for example:

```
export CLOUDSDK_CORE_PROJECT="my-neo4j-project"
export CLOUDSDK_COMPUTE_ZONE="europe-west2-a"
export CLOUDSDK_COMPUTE_REGION="europe-west2"
```

3. If you do not have a Google Kubernetes Engine (GKE) cluster, you can create a single-node one using:

```
gcloud container clusters create my-neo4j-gke-cluster --num-nodes=1 --machine-type "e2-standard-2"
```

> **Note:**
>
> e2-standard-2 is the minimum instance type required for running the examples of this startup guide on GKE.

4. Configure `kubectl` to use your GKE cluster using:

```
gcloud container clusters get-credentials my-neo4j-gke-cluster
```

```
Fetching cluster endpoint and auth data.
kubeconfig entry generated for my-neo4j-gke-cluster.
```

1. Install the `aws` command-line interface (CLI) (https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html). Make sure you complete the AWS configuration step (https://docs.aws.amazon.com/cli/latest/userguide/getting-started-quickstart.html).
2. Install the `eksctl` command-line interface (CLI) (https://docs.aws.amazon.com/eks/latest/userguide/getting-started-eksctl.html).

3. All the shell commands in this guide assume that the AWS region to use has been set using the `AWS_DEFAULT_REGION` environment variable, for example:

```
export AWS_DEFAULT_REGION="eu-west-1"
```

4. If you do not have an AWS Elastic Kubernetes Service (EKS) cluster, you can create a single-node one using the following command:

Note:

This command requires that you have a public key named *id_rsa.pub*. If you do not have one, you can generate it by running:

```
ssh-keygen -t rsa -C "your-name@example.com"
```

```
eksctl create cluster --name "my-neo4j-eks-cluster" --region "${AWS_DEFAULT_REGION}"
--nodegroup-name "neo4j-nodes" --nodes-min 1 --nodes-max 2 --node-type c4.xlarge --nodes 1
--node-volume-size 10 --ssh-access --with-oidc
```

5. Create an IAM role (e.g., `AmazonEKS_EBS_CSI_DriverRole`) and attach the required AWS-managed policy to it (e.g., `AmazonEBSCSIDriverPolicy`).

```
eksctl create iamserviceaccount \
  --name ebs-csi-controller-sa \
  --namespace kube-system \
  --cluster my-neo4j-eks-cluster \
  --attach-policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy \
  --approve \
  --role-only \
  --role-name AmazonEKS_EBS_CSI_DriverRole
```

6. Add the EBS CSI Driver as an Amazon EKS add-on to your cluster:

```
eksctl create addon \
  --name aws-ebs-csi-driver \
  --cluster my-neo4j-eks-cluster \
  --service-account-role-arn arn:aws:iam::<aws-account-id>:role/AmazonEKS_EBS_CSI_DriverRole
\
  --force
```

Note:

Make sure to replace `<aws-account-id>` with your AWS account ID.

7. Configure `kubectl` to use your EKS cluster using:

```
aws eks update-kubeconfig --name my-neo4j-eks-cluster
```

1. Install the `az` command-line interface (CLI) (https://learn.microsoft.com/en-us/cli/azure/

install-azure-cli?view=azure-cli-latest).

2. Verify that you have a Resource Group with:
   - An Azure Kubernetes Service (AKS) cluster.
   - The AKS cluster principal needs to be assigned roles that allow it to manage Microsoft.Compute/disks in the Resource Group.
3. Set the Resource group and the location to use as defaults using:

```
az configure --defaults group=<MyResourceGroup>
az configure --defaults location=<MyAzureLocation>
```

4. If you do not have an AKS cluster, follow the steps to create a single-node one.
   a. Create a cluster by running:

```
az aks create --name my-neo4j-aks-cluster --node-count=1
```

   b. Configure `kubectl` to use your AKS cluster using:

```
az aks get-credentials --name my-neo4j-aks-cluster --admin
```

1. Install Docker Desktop for macOS. For more information, see Docker official documentation.
2. Enable the Docker Desktop Kubernetes engine. For more information, see Docker official documentation.
3. Verify that you do not have a running instance of Neo4j (e.g., via Neo4j Desktop or Neo4j Browser) to avoid port clashes.

Create a Helm deployment *values.yaml* file

You create a Helm deployment YAML file containing all the configurations for your Neo4j standalone instance.

Important configuration parameters

`neo4j.name`

Starting from Neo4j 5.0.0, standalone servers and cluster servers have no distinction. This means a standalone server can be upgraded to a cluster by adding more servers. Therefore, the `neo4j.name` parameter, which value links together servers in a cluster, is mandatory, and the installation will fail if it is not specified. `neo4j.name` must be unique within a namespace.

`neo4j.resources`

The size of a Neo4j instance is defined by the values of the `neo4j.resources.cpu` and `neo4j.resources.memory` parameters. The minimum is `0.5` CPU and `2GB` memory. If invalid or less than

the minimum values are provided, Helm will throw an error, for example:

```
Error: template: neo4jtemplates/_helpers.tpl:157:11: executing "neo4j.resources.evaluateCPU" at <fail
(printf "Provided cpu value %s is less than minimum. \n %s" (.Values.neo4j.resources.cpu) (include
"neo4j.resources.invalidCPUMessage" .))>: error calling fail: Provided cpu value 0.25 is less than
minimum.
 cpu value cannot be less than 0.5 or 500m
```

For more information, see Important configuration parameters.

`neo4j.password`

The password for the `neo4j` user.
If you do not provide a password, the Neo4j Helm chart will automatically generate one for you. (Make a note of it.)

> **Note:**
>
> You cannot use `neo4j` as the initial password as this is the default password.

`neo4j.edition` and `neo4j.acceptLicenseAgreement`

By default, the standalone Helm chart installs Neo4j Community Edition.
If you want to install Neo4j Enterprise Edition, set the configuration parameters `edition: "enterprise"` and acknowledge license compliance by setting `neo4j.acceptLicenseAgreement` to `"yes"` if you have a valid license or to `"eval"` if you want to accept the Neo4j evaluation license.
For more information on how to obtain a valid license for Neo4j Enterprise Edition, see https://neo4j.com/licensing/ and link:https://neo4j.com/terms/licensing/, or use the form Contact Neo4j.

`volumes.data`

The `volumes.data` parameter maps the `data` volume mount of your Neo4j to the persistent volume that you have for that instance. For more information, see Volume mounts and persistent volumes.

> **Note:**
>
> For details of all Neo4j Helm chart configuration options, see Configure a Neo4j Helm deployment.

Create a *values.yaml* file

Select the tab as per your Kubernetes environment and using the provided example, create a YAML file for your standalone instance.

This guide assumes that the YAML file is named *my-neo4j.values.yaml*.

```yaml
neo4j:
  name: my-standalone
  resources:
    cpu: "0.5"
    memory: "2Gi"
```

```
  # Uncomment to set the initial password
  #password: "my-initial-password"

  # Uncomment to use enterprise edition
  #edition: "enterprise"
  #acceptLicenseAgreement: "yes"

volumes:
  data:
    mode: "dynamic"
    dynamic:
      # In GKE;
      # * premium-rwo provisions SSD disks (recommended)
      # * standard-rwo provisions balanced SSD-backed disks
      # * standard provisions HDD disks
      storageClassName: premium-rwo
```

```
neo4j:
  name: my-standalone
  resources:
    cpu: "0.5"
    memory: "2Gi"

  # Uncomment to set the initial password
  #password: "my-initial-password"

  # Uncomment to use enterprise edition
  #edition: "enterprise"
  #acceptLicenseAgreement: "yes"

volumes:
  data:
    mode: "dynamic"
    dynamic:
     # gp2 is a general-purpose SSD volume
     storageClassName: gp2
```

```
neo4j:
  name: my-standalone
  resources:
    cpu: "0.5"
    memory: "2Gi"

  # Uncomment to set the initial password
  #password: "my-initial-password"

  # Uncomment to use enterprise edition
  #edition: "enterprise"
  #acceptLicenseAgreement: "yes"

volumes:
  data:
    mode: "dynamic"
    dynamic:
      # * managed-csi-premium provisions premium SSD disks (recommended)
      # * managed-csi provisions standard SSD-backed disks
      storageClassName: managed-csi-premium
```

```
neo4j:
  name: my-standalone
  resources:
    cpu: "0.5"
    memory: "2Gi"
```

```
  # Uncomment to set the initial password
  #password: "my-initial-password"

  # Uncomment to use enterprise edition
  #edition: "enterprise"
  #acceptLicenseAgreement: "yes"

volumes:
  data:
    mode: defaultStorageClass
    defaultStorageClass:
      requests:
        storage: 2Gi
```

Install a Neo4j standalone instance

Getting everything to work in Kubernetes requires that certain K8s objects have specific names that are referenced elsewhere. Each individual Neo4j instance is a Helm "release" and has a *release name*. All other names and labels of K8s objects created by the Helm charts derive from both `neo4j.name` and *release name*.
Release name must consist of lowercase alphanumeric characters, `-` or `.`, and must start and end with an alphanumeric character. This guide assumes the release name is `my-neo4j-release` and `neo4j.name` is `my-standalone`.

1. Install Neo4j using the deployment *values.yaml* file, created in Create a value.yaml file, and the *neo4j/neo4j* Helm chart:

   a. Create a `neo4j` namespace and configure it to be used in the current context:

   ```
   kubectl create namespace neo4j
   kubectl config set-context --current --namespace=neo4j
   ```

   b. Install the Neo4j standalone server:

   ```
   helm install my-neo4j-release neo4j/neo4j --namespace neo4j -f my-neo4j.values.yaml
   ```

   Example output

   ```
   LAST DEPLOYED: Wed Oct 26 15:19:17 2022
   NAMESPACE: neo4j
   STATUS: deployed
   REVISION: 1
   TEST SUITE: None
   NOTES:
   Thank you for installing neo4j.

   Your release "my-neo4j-release" has been installed in namespace "neo4j".

   The neo4j user's password has been set to "my-password".To view the progress of the rollout try:

     $ kubectl --namespace "neo4j" rollout status --watch --timeout=600s statefulset/my-neo4j-release

   Once rollout is complete you can log in to Neo4j at "neo4j://my-neo4j-
   release.neo4j.svc.cluster.local:7687". Try:

     $ kubectl run --rm -it --namespace "neo4j" --image "neo4j:5.1.0" cypher-shell \
        -- cypher-shell -a "neo4j://my-neo4j-release.neo4j.svc.cluster.local:7687" -u neo4j -p "my-
   password"
   ```

```
Graphs are everywhere!
```

2. Run the `kubectl rollout` command provided in the output of `helm install` to watch the Neo4j's rollout until it is complete.

```
kubectl rollout status --watch --timeout=600s statefulset/my-neo4j-release
```

> **Note:**
>
> Since you have not passed a password for the `neo4j` user, the Neo4j Helm chart has set an automatically generated one. You can find it in the Helm install output. Please make a note of it.

Verify the installation

1. Check that the `statefulset` is OK.

```
kubectl get statefulsets
```

```
NAME               READY   AGE
my-neo4j-release   1/1     2m11s
```

2. Check that the pod is `Running`:

```
kubectl get pods
```

```
NAME                 READY   STATUS    RESTARTS   AGE
my-neo4j-release-0   1/1     Running   0          16m
```

3. Check that the pod logs look OK:

```
kubectl exec my-neo4j-release-0 -- tail -n50 /logs/neo4j.log
```

```
2022-10-26 14:19:51.728+0000 INFO  Command expansion is explicitly enabled for configuration
2022-10-26 14:19:51.733+0000 WARN  Unrecognized setting. No declared setting with name:
server.panic.shutdown_on_panic.
2022-10-26 14:19:51.749+0000 INFO  Starting...
2022-10-26 14:19:53.062+0000 INFO  This instance is ServerId{cb9f2f3c} (cb9f2f3c-cd70-40b1-
ac8e-13d9c4d26173)
2022-10-26 14:19:54.970+0000 INFO  ======== Neo4j 5.1.0 ========
2022-10-26 14:19:59.528+0000 INFO  Bolt enabled on 0.0.0.0:7687.
2022-10-26 14:20:01.523+0000 INFO  Remote interface available at http://localhost:7474/
2022-10-26 14:20:01.530+0000 INFO  id:
EF772BAFBDCD3C4921D00A5707C88D6EDE514915DBCC7134E8704AFA15DC19C8
2022-10-26 14:20:01.530+0000 INFO  name: system
2022-10-26 14:20:01.531+0000 INFO  creationDate: 2022-10-26T14:19:56.631Z
2022-10-26 14:20:01.531+0000 INFO  Started.
```

4. Check that the services look OK:

```
kubectl get services
```

```
NAME                                    TYPE           CLUSTER-IP    EXTERNAL-IP
PORT(S)                         AGE
service/my-neo4j-release-lb-neo4j       LoadBalancer   10.36.5.34    34.105.179.172
7474:30288/TCP,7687:30584/TCP   14m
service/kubernetes                      ClusterIP      10.36.0.1     <none>
443/TCP                         22h
service/my-neo4j-release                ClusterIP      10.36.11.18   <none>
7687/TCP,7474/TCP               14m
service/my-neo4j-release-admin          ClusterIP      10.36.3.238   <none>
6362/TCP,7687/TCP,7474/TCP      14m
```

5. Use the external IP of the LoadBalancer to access Neo4j from an application outside the Kubernetes cluster. For more information, see Applications accessing Neo4j from outside Kubernetes.
   a. In a web browser, open the Neo4j Browser at *http://EXTERNAL_IP:7474/browser*.
   b. Use the automatically-generated password (as printed in the output of the `helm install` command) or the one you have configured in the *my-neo4j.values.yaml* file.

1. Check that `statefulset` is OK.

```
kubectl get statefulsets
```

```
NAME               READY   AGE
my-neo4j-release   1/1     5m11s
```

2. Check that the PVC is OK (the `STATUS` must be `Bound`):

```
kubectl get pvc
```

```
NAME                        STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
data-my-neo4j-release-0     Bound    pvc-c35b7abd-1778-4c15-ada5-98912722b7c5   100Gi
RWO            premium-rwo    7m57s
```

3. Check that the pod is `READY`:

```
kubectl get pods
```

```
NAME                  READY   STATUS       RESTARTS   AGE
my-neo4j-release-0    1/1     Running      0          5m53s
```

4. Check that the pod logs look OK:

```
kubectl exec my-neo4j-release-0 -- tail -n50 /logs/neo4j.log
```

```
Changed password for user 'neo4j'.
Directories in use:
  home:         /var/lib/neo4j
  config:       /config/
  logs:         /data/logs
  plugins:      /var/lib/neo4j/plugins
  import:       /var/lib/neo4j/import
  data:         /var/lib/neo4j/data
  certificates: /var/lib/neo4j/certificates
  run:          /var/lib/neo4j/run
Starting Neo4j.
2021-06-02 17:38:27.791+0000 INFO  Command expansion is explicitly enabled for configuration
2021-06-02 17:38:27.819+0000 INFO  Starting...
2021-06-02 17:38:31.195+0000 INFO  ======== Neo4j 5.1.0 ========
2021-06-02 17:38:34.168+0000 INFO  Initializing system graph model for component 'security-
users' with version -1 and status UNINITIALIZED
2021-06-02 17:38:34.188+0000 INFO  Setting up initial user from `auth.ini` file: neo4j
2021-06-02 17:38:34.190+0000 INFO  Creating new user 'neo4j' (passwordChangeRequired=false,
suspended=false)
2021-06-02 17:38:34.205+0000 INFO  Setting version for 'security-users' to 2
2021-06-02 17:38:34.214+0000 INFO  After initialization of system graph model component
'security-users' have version 2 and status CURRENT
2021-06-02 17:38:34.223+0000 INFO  Performing postInitialization step for component
'security-users' with version 2 and status CURRENT
2021-06-02 17:38:34.561+0000 INFO  Bolt enabled on 0.0.0.0:7687.
2021-06-02 17:38:36.910+0000 INFO  Remote interface available at http://localhost:7474/
2021-06-02 17:38:36.912+0000 INFO  Started.
```

5. Check that the services look OK:

```
kubectl get services
```

```
NAME                     TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
kubernetes               ClusterIP      10.96.0.1        <none>        443/TCP
3d1h
my-neo4j-release         ClusterIP      10.103.103.142   <none>
7687/TCP,7474/TCP,7473/TCP                      2d8h
my-neo4j-release-admin   ClusterIP      10.99.11.122     <none>
6362/TCP,7687/TCP,7474/TCP,7473/TCP             2d8h
my-neo4j-release-neo4j   LoadBalancer   10.110.138.165   localhost
7474:31237/TCP,7473:32026/TCP,7687:32169/TCP    2d3h
```

6. In a web browser, open the Neo4j Browser at *http://localhost:7474/*.
7. Use the automatically-generated password (as printed in the output of the `helm install` command) or the one you have set up with the `helm install` command.

Uninstall Neo4j and clean up the created resources

Uninstall Neo4j Helm deployment

Uninstall the Neo4j Helm deployment.

```
helm uninstall my-neo4j-release
```

Example output

```
release "my-neo4j-release" uninstalled
```

Fully remove all the data and resources

Uninstalling the Helm release does not remove the created resources and data. Therefore, after uninstalling the helm deployment, you also have to delete all the data and resources.

1. Delete all persistent volume claims in the `neo4j` namespace:

```
kubectl delete pvc --all --namespace neo4j
```

2. Delete the entire Kubernetes cluster in your cloud provider:

```
gcloud container clusters delete my-neo4j-gke-cluster
```

```
eksctl delete cluster --name=my-neo4j-eks-cluster
```

```
az aks delete --name my-neo4j-aks-cluster --resource-group <MyResourceGroup>
```

Quickstart: Deploy a cluster

The quickstart for deploying a Neo4j cluster contains the following:

> **Note:**
>
> This guide shows how to deploy a cluster with three servers.

- Neo4j Helm chart for cluster deployments — A schematic representation of how to use the Neo4j Helm chart for deploying a cluster.
- Prerequisites — Set up your environment for deploying a Neo4j cluster on Kubernetes.
- Create Helm deployment values files — Create a Helm deployment *values.yaml* file for each Neo4j cluster member.
- Install Neo4j cluster servers — Install each of your Neo4j cluster servers using its deployment YAML file and the *neo4j/neo4j* Helm chart.
- Verify cluster formation — Verify that the Neo4j servers have formed a cluster.
- Access the Neo4j cluster from inside Kubernetes — Access the Neo4j cluster from inside Kubernetes using a specific server or the headless service.
- Access the Neo4j cluster from outside Kubernetes — Access the Neo4j cluster from outside Kubernetes using a load balancer.

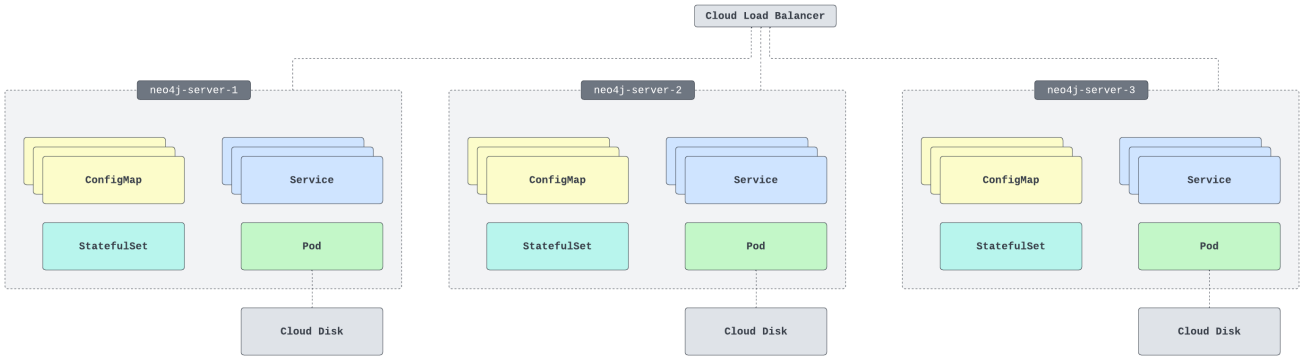
- Uninstall the Neo4j cluster and clean up the created resources — Uninstall all Neo4j Helm deployments and clean up the created resources.

Neo4j Helm chart for cluster deployments

To offer the flexibility required to cluster users, the Helm chart for clusters takes a modular approach: the typical cluster deployment requires several Helm chart installations. For example, a cluster with three servers requires installing the *neo4j* chart three times. By separating the charts up, users can create a different topology of Neo4j and a different topology in their Kubernetes clusters to suit their needs.

The following diagram is a schematic representation of the Helm charts involved and the Kubernetes and Cloud resources they instantiate when installed:

Neo4j cluster setup



The diagram shows an example of a Neo4j cluster setup with three servers. The Kubernetes setup includes a headless service for accessing the cluster from inside Kubernetes and a load-balancer service for accessing the cluster from outside Kubernetes.

Prerequisites

Before you can deploy a Neo4j cluster on Kubernetes, you need to:

General prerequisites

- Configure the Neo4j Helm chart repository.
- Obtain a valid license if you want to install Neo4j Enterprise Edition or set the configuration parameters `edition: "enterprise"` to `"eval"` if you want to accept the Neo4j evaluation license.
  For more information on how to obtain a commercial license, see https://neo4j.com/licensing/ and link:https://neo4j.com/terms/licensing/, or use the form Contact Neo4j.
- Install the Kubernetes client command-line tool `kubectl` (https://kubernetes.io/docs/tasks/tools/).
- Set up a Kubernetes cluster with sufficient CPU and memory for your Neo4j deployment.

> **Note:**
>
> This guide works with minimum CPU and memory allocated per Neo4j instance. However, the Neo4j system requirements largely depend on the use of the software.

Therefore, for running Neo4j in development or production environments, please refer to System requirements.

If you do not have a Kubernetes cluster, you can configure a multi-node one as per your environment, see the next section Environment-specific prerequisites.

Environment-specific prerequisites

Select the tab as per your Kubernetes environment and complete all prerequisites on it.

1. Install the `gcloud` command-line interface (CLI) (https://cloud.google.com/sdk/docs/install).
2. All the shell commands in this guide assume that the GCP Project, compute zone, and region to use, have been set using the `CLOUDSDK_CORE_PROJECT`, `CLOUDSDK_COMPUTE_ZONE`, and `CLOUDSDK_COMPUTE_REGION` environment variables, for example:

```
export CLOUDSDK_CORE_PROJECT="neo4j-helm"
export CLOUDSDK_COMPUTE_ZONE="europe-west6-c"
export CLOUDSDK_COMPUTE_REGION="europe-west6"
export CLOUDSDK_CONTAINER_CLUSTER="my-neo4j-gke-cluster"
```

3. If you do not have a Google Kubernetes Engine (GKE) cluster, you can create a multi-node cluster (one node per Neo4j instance) using:

```
gcloud container clusters create my-neo4j-gke-cluster --num-nodes=<num> --machine-type "e2-standard-2"
```

> **Note:**
>
> e2-standard-2 is the minimum instance type required for running the examples of this startup guide on GKE.

4. Configure `kubectl` to use your GKE cluster using:

```
gcloud container clusters get-credentials my-neo4j-gke-cluster
```

```
Fetching cluster endpoint and auth data.
kubeconfig entry generated for my-neo4j-gke-cluster.
```

1. Install the `aws` command-line interface (CLI) (https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html). Make sure you complete the AWS configuration step (https://docs.aws.amazon.com/cli/latest/userguide/getting-started-quickstart.html).
2. Install the `eksctl` command-line interface (CLI) (https://docs.aws.amazon.com/eks/latest/userguide/getting-started-eksctl.html).
3. All the shell commands in this guide assume that the AWS region to use has been set using

the `AWS_DEFAULT_REGION` environment variable, for example:

```
export AWS_DEFAULT_REGION="eu-west-1"
```

4. If you do not have an AWS Elastic Kubernetes Service (EKS) cluster, you can create a multi-node cluster (one node per Neo4j instance) using the following command:

> **Note:**
>
> This command requires that you have a public key named *id_rsa.pub*. If you do not have one, you can generate it by running:
>
> ```
> ssh-keygen -t rsa -C "your-name@example.com"
> ```

```
eksctl create cluster --name "my-neo4j-eks-cluster" --region "${AWS_DEFAULT_REGION}"
--nodegroup-name "neo4j-nodes" --nodes-min 1 --nodes-max 4 --node-type c4.xlarge --nodes 4
--node-volume-size 10 --ssh-access --with-oidc
```

5. Create an IAM role (e.g., `AmazonEKS_EBS_CSI_DriverRole`) and attach the required AWS-managed policy to it (e.g., `AmazonEBSCSIDriverPolicy`).

```
eksctl create iamserviceaccount \
  --name ebs-csi-controller-sa \
  --namespace kube-system \
  --cluster my-neo4j-eks-cluster \
  --attach-policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy \
  --approve \
  --role-only \
  --role-name AmazonEKS_EBS_CSI_DriverRole
```

6. Add the EBS CSI Driver as an Amazon EKS add-on to your cluster:

```
eksctl create addon \
  --name aws-ebs-csi-driver \
  --cluster my-neo4j-eks-cluster \
  --service-account-role-arn arn:aws:iam::<aws-account-id>:role/AmazonEKS_EBS_CSI_DriverRole
\
  --force
```

> **Note:**
>
> Make sure to replace `<aws-account-id>` with your AWS account ID.

7. Configure `kubectl` to use your EKS cluster using:

```
aws eks update-kubeconfig --name my-neo4j-eks-cluster
```

1. Install the `az` command-line interface (CLI) (https://learn.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest).

2. Verify that you have a Resource Group with:
   - An Azure Kubernetes Service (AKS) cluster.
   - The AKS cluster principal needs to be assigned roles that allow it to manage Microsoft.Compute/disks in the Resource Group.
3. Set the Resource group and the location to use as defaults using:

```
az configure --defaults group=<MyResourceGroup>
az configure --defaults location=<MyAzureLocation>
```

> **Note:**
>
> If you do not have an AKS cluster, follow the steps to create a multi-node cluster (one node per Neo4j instance).
>
> 1. Create a cluster by running:
>
> ```
> az aks create --name my-neo4j-aks-cluster --node-count=<num>
> ```
>
> 2. Configure `kubectl` to use your AKS cluster using:
>
> ```
> az aks get-credentials --name my-neo4j-aks-cluster --admin
> ```

Create Helm deployment values files

You create a Helm deployment YAML file for each Neo4j cluster member with all the configuration settings.

Important configuration parameters

`neo4j.name`

Starting from Neo4j 5.0.0, standalone servers and cluster servers have no distinction. This means a standalone server can be upgraded to a cluster by adding more servers. All Neo4j cluster servers are linked together by the value of the parameter `neo4j.name`. When installed via the Neo4j Helm chart, they will join the cluster identified by `neo4j.name`.
`neo4j.name` is different from the Helm *release name*, which is used to reference a cluster server elsewhere in Kubernetes.
All other names and labels of K8s objects created by the Helm chart derive from both `neo4j.name` and *release name*. The `neo4j.name` parameter is mandatory, and the installation will fail if it is not specified. `neo4j.name` must be unique within a namespace.

> **Note:**
>
> If you want to install 2 clusters in the same namespace, they must have different values

for `neo4j.name`.

`neo4j.minimumClusterSize`

Starting from Neo4j 5.0.0, standalone servers and cluster servers have no distinction. By default, servers in a cluster can host primary and secondary databases. See the Operational view for more details.
`neo4j.minimumClusterSize` is set to 1 by default, which means the server starts without waiting for the other servers. When installing a cluster, you should set `neo4j.minimumClusterSize` to the number of desired members in the cluster. If you later decide to add an extra cluster server in excess of `neo4j.minimumClusterSize`, you need to manually enable it using the Cypher command `ENABLE SERVER`.
From Neo4j 5.20 onwards, you can also use the configuration parameter `neo4j.operations.enableServer` to automatically enable the server when it joins the cluster. For more information, see Horizontal scaling.

`neo4j.resources`

The size of a Neo4j cluster member is defined by the values of the `neo4j.resources.cpu` and `neo4j.resources.memory` parameters. The minimum for a Neo4j instance is `0.5` CPU and `2GB` memory. If invalid or less than the minimum values are provided, Helm will throw an error, for example:

```
Error: template: neo4j/templates/_helpers.tpl:157:11: executing "neo4j.resources.evaluateCPU" at <fail
(printf "Provided cpu value %s is less than minimum. \n %s" (.Values.neo4j.resources.cpu) (include
"neo4j.resources.invalidCPUMessage" .))>: error calling fail: Provided cpu value 0.25 is less than
minimum.
 cpu value cannot be less than 0.5 or 500m
```

For more information, see Important configuration parameters.

`neo4j.password`

The password for the `neo4j` user. The same password must be set for all cluster members.
If you do not provide a password, the Neo4j Helm chart will automatically generate one for you. (Make a note of it.)

Note:

You cannot use `neo4j` as the initial password as this is the default password.

`neo4j.edition` and `neo4j.acceptLicenseAgreement`

By default, the Helm chart installs Neo4j Community Edition.
If you want to install Neo4j Enterprise Edition, set the configuration parameters `edition: "enterprise"` and acknowledge license compliance by setting `neo4j.acceptLicenseAgreement` to `"yes"` if you have a valid license or to `"eval"` if you want to accept the link:Neo4j evaluation license.
For more information on how to obtain a valid license for Neo4j Enterprise Edition, see https://neo4j.com/licensing/ and link:https://neo4j.com/terms/licensing/, or use the form Contact Neo4j.

`volumes.data`

The `volumes.data` parameter maps the `data` volume mount of each cluster member to the persistent volume for that member. For more information, see Volume mounts and persistent volumes.

> **Note:**
>
> For details of all Neo4j Helm chart configuration options, see Configure a Neo4j Helm deployment.

Create a values.yaml file for each server

Select the tab as per your Kubernetes environment, and using the provided example, create a YAML file for each of your cluster servers with all the configuration settings.

This guide assumes that the YAML files are named *server-1.values.yaml*, *server-2.values.yaml* and *server-3.values.yaml*.

The examples use storage classes provided by the vendor. If you want to use a different type of storage class, such as regional disks, consult the cloud vendor documentation on creating new storage classes.

```yaml
neo4j:
  name: "my-cluster"
  minimumClusterSize: 3
  resources:
    cpu: "0.5"
    memory: "2Gi"
  password: "my-password"
  edition: "enterprise"
  acceptLicenseAgreement: "yes"

volumes:
  data:
    mode: "dynamic"
    dynamic:
      # * premium-rwo provisions SSD disks (recommended)
      # * standard-rwo provisions balanced SSD-backed disks
      # * standard provisions HDD disks
      storageClassName: premium-rwo
```

```yaml
neo4j:
  name: "my-cluster"
  minimumClusterSize: 3
  resources:
    cpu: "0.5"
    memory: "2Gi"
  password: "my-password"
  edition: "enterprise"
  acceptLicenseAgreement: "yes"

volumes:
  data:
    mode: "dynamic"
    dynamic:
     # gp2 is a general-purpose SSD volume
     storageClassName: gp2
```

```yaml
neo4j:
  name: "my-cluster"
  minimumClusterSize: 3
```

```
  resources:
    cpu: "0.5"
    memory: "2Gi"
  password: "my-password"
  edition: "enterprise"
  acceptLicenseAgreement: "yes"

volumes:
  data:
    mode: "dynamic"
    dynamic:
      # * managed-csi-premium provisions premium SSD disks (recommended)
      # * managed-csi provisions standard SSD backed disks
      storageClassName: managed-csi-premium
```

Install Neo4j cluster servers

Getting everything to work in Kubernetes requires that certain K8s objects have specific names that are referenced elsewhere. Each Neo4j instance is a Helm "release" and has a *release name*. Release name must consist of lowercase alphanumeric characters, `-` or `.`, and must start and end with an alphanumeric character.

> **Note:**
>
> The following example installations use `server-1`, `server-2`, and `server-3` as release names for the cluster members.

1. Install each server individually using the deployment *server-<num>.values.yaml* file created in Create Helm deployment values files and the *neo4j/neo4j* Helm chart.

   a. Create a `neo4j` namespace and configure it to be used in the current context:

   ```
   kubectl create namespace neo4j
   kubectl config set-context --current --namespace=neo4j
   ```

   b. Install `server-1`:

   ```
   helm install server-1 neo4j/neo4j --namespace neo4j -f server-1.values.yaml
   ```

   Example output

   ```
   NAME: server-1
   LAST DEPLOYED: Wed Oct 26 11:50:41 2022
   NAMESPACE: neo4j
   STATUS: deployed
   REVISION: 1
   TEST SUITE: None
   NOTES:
   Thank you for installing neo4j.

   Your release "server-1" has been installed in namespace "neo4j".

   The neo4j user's password has been set to "my-password".

   This release creates a single member of a Neo4j cluster. It will not become ready until it is able
   to form a working Neo4j cluster by joining other Neo4j servers. To create a working cluster at
   least 3 servers are required.
   ```

```
Once you have a working Neo4j cluster, you can access the Neo4j browser using the IP address of
the my-cluster-lb-neo4j service
eg. http://[SERVICE_IP]:7474

Graphs are everywhere!
```

c. Install `server-2`:

```
helm install server-2 neo4j/neo4j --namespace neo4j -f server-2.values.yaml
```

Example output

```
NAME: server-2
LAST DEPLOYED: Wed Oct 26 11:51:27 2022
NAMESPACE: neo4j
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Thank you for installing neo4j.

Your release "server-2" has been installed in namespace "neo4j".

The neo4j user's password has been set to "my-password".

This release creates a single member of a Neo4j cluster. It will not become ready until it is able
to form a working Neo4j cluster by joining other Neo4j servers. To create a working cluster at
least 3 servers are required.

Once you have a working Neo4j cluster, you can access the Neo4j browser using the IP address of
the my-cluster-lb-neo4j service
eg. http://[SERVICE_IP]:7474

Graphs are everywhere!
```

d. Install `server-3`:

```
helm install server-3 neo4j/neo4j --namespace neo4j -f server-3.values.yaml
```

Example output

```
NAME: server-3
LAST DEPLOYED: Wed Oct 26 11:52:02 2022
NAMESPACE: neo4j
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Thank you for installing neo4j.

Your release "server-3" has been installed in namespace "neo4j".

The neo4j user's password has been set to "my-password".

This release creates a single member of a Neo4j cluster. It will not become ready until it is able
to form a working Neo4j cluster by joining other Neo4j servers. To create a working cluster at
least 3 servers are required.

Once you have a working Neo4j cluster, you can access the Neo4j browser using the IP address of
the my-cluster-lb-neo4j service
eg. http://[SERVICE_IP]:7474

Graphs are everywhere!
```

Note:

If you have not passed a password for the `neo4j` user, the Neo4j Helm chart has automatically generated one for you. The password is the same for all cluster members. You can find it in the Helm install outputs. Make a note of it.

2. Verify that the installed servers have formed a cluster. See the next section Verify the Neo4j cluster formation.

Verify the Neo4j cluster formation

You check the pods and services to verify that the servers have managed to form a cluster.

1. Check that the pods are `READY`.

   Typically, it takes a minute or two after the pods are running. They become `READY` after they form a cluster. You can watch pod status changes by adding the `-w` option to the `kubectl` command (`kubectl get pods -w`).

   ```
   kubectl get pods
   ```

   ```
   NAME       READY   STATUS    RESTARTS   AGE
   server-1-0   1/1     Running   0          147m
   server-2-0   1/1     Running   0          147m
   server-3-0   1/1     Running   0          147m
   ```

2. You can also check the logs of the pods, for example:

   ```
   kubectl exec server-2-0 -- tail /logs/neo4j.log
   ```

   ```
   2022-10-26 11:37:42.516+0000 INFO  Remote interface available at http://localhost:7474/
   2022-10-26 11:37:42.539+0000 INFO  id:
   72419021DEA149919ABFFD4930EDE02131E52E77F7EADF62C6323BC4F2D01EC5
   2022-10-26 11:37:42.539+0000 INFO  name: system
   2022-10-26 11:37:42.539+0000 INFO  creationDate: 2022-10-26T11:37:28.784Z
   2022-10-26 11:37:42.540+0000 INFO  Started.
   2022-10-26 11:37:47.177+0000 INFO  This instance bootstrapped the 'neo4j' database.
   2022-10-26 11:37:58.644+0000 INFO  Connected to server-3-
   internals.neo4j.svc.cluster.local/10.24.0.131:7000 [RAFT version:1.0]
   2022-10-26 12:01:32.066+0000 INFO  Direct driver instance 577812770 created for server address server-
   1-internals.neo4j.svc.cluster.local:7688
   2022-10-26 12:02:40.172+0000 INFO  Closing driver instance 577812770
   2022-10-26 12:02:40.174+0000 INFO  Closing connection pool towards server-1-
   internals.neo4j.svc.cluster.local:7688
   ```

3. Check that the services look good:

   ```
   kubectl get services
   ```

   ```
   NAME            TYPE          CLUSTER-IP      EXTERNAL-IP    PORT(S)
   AGE
   my-cluster-lb-neo4j   LoadBalancer   10.28.12.119   35.234.152.117
   7474:32169/TCP,7473:32145/TCP,7687:32624/TCP                              148m
   server-1              ClusterIP      10.28.13.216   <none>           7687/TCP,7474/TCP,7473/TCP
   148m
   server-1-admin        ClusterIP      10.28.13.244   <none>
   6362/TCP,7687/TCP,7474/TCP,7473/TCP                                       148m
   ```

```
server-1-internals    ClusterIP      None           <none>
6362/TCP,7687/TCP,7474/TCP,7473/TCP,7688/TCP,5000/TCP,7000/TCP,6000/TCP   148m
server-2              ClusterIP      10.28.10.237   <none>           7687/TCP,7474/TCP,7473/TCP
148m
server-2-admin        ClusterIP      10.28.7.113    <none>
6362/TCP,7687/TCP,7474/TCP,7473/TCP                                      148m
server-2-internals    ClusterIP      None           <none>
6362/TCP,7687/TCP,7474/TCP,7473/TCP,7688/TCP,5000/TCP,7000/TCP,6000/TCP   148m
server-3              ClusterIP      10.28.3.164    <none>           7687/TCP,7474/TCP,7473/TCP
148m
server-3-admin        ClusterIP      10.28.14.77    <none>
6362/TCP,7687/TCP,7474/TCP,7473/TCP                                      148m
server-3-internals    ClusterIP      None           <none>
6362/TCP,7687/TCP,7474/TCP,7473/TCP,7688/TCP,5000/TCP,7000/TCP,6000/TCP   148m
```

For more information about the Neo4j services, see Accessing Neo4j.

Access the Neo4j cluster from inside Kubernetes

By default, client-side routing is used for accessing a Neo4j cluster from inside Kubernetes.

Access the Neo4j cluster using a specific member

You run `cypher-shell` in a new pod and point it directly to one of the servers.

1. Run `cypher-shell` in a pod to access, for example, `server-3`:

```
kubectl run --rm -it --env=NEO4J_ACCEPT_LICENSE_AGREEMENT=yes --image "neo4j:5.26.25-enterprise"
cypher-shell \
     -- cypher-shell -a "neo4j://server-3.neo4j.svc.cluster.local:7687" -u neo4j -p "my-password"
```

```
If you don't see a command prompt, try pressing enter.


Connected to Neo4j using Bolt protocol version 5 at neo4j://server-3.neo4j.svc.cluster.local:7687 as
user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
```

2. Run the Cypher command `SHOW DATABASES` to verify that all cluster servers are online.

```
SHOW DATABASES;
```

```
+-------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------+
| name     | type       | aliases | access       | address                                | role
| writer | requestedStatus | currentStatus | statusMessage | default | home  | constituents |
+-------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------+
| "neo4j"  | "standard" | []      | "read-write" | "server-2.neo4j.svc.cluster.local:7687" | "primary"
| TRUE   | "online"        | "online"      | ""            | TRUE    | TRUE  | []           |
| "neo4j"  | "standard" | []      | "read-write" | "server-1.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""            | TRUE    | TRUE  | []           |
| "neo4j"  | "standard" | []      | "read-write" | "server-3.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""            | TRUE    | TRUE  | []           |
| "system" | "system"   | []      | "read-write" | "server-2.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""            | FALSE   | FALSE | []           |
| "system" | "system"   | []      | "read-write" | "server-1.neo4j.svc.cluster.local:7687" | "primary"
| TRUE   | "online"        | "online"      | ""            | FALSE   | FALSE | []           |
| "system" | "system"   | []      | "read-write" | "server-3.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""            | FALSE   | FALSE | []           |
+-------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------+
```

```
-------------------------------------------------------------------------------------------+

6 rows
ready to start consuming query after 27 ms, results consumed after another 243 ms
```

3. Run the Cypher command `SHOW SERVERS` to verify that all cluster servers are enabled:

```
SHOW SERVERS;
```

```
+-------------------------------------------------------------------------------------------------------
-----------------------------+
| name                                 | address                                  | state     |
health      | hosting             |
+-------------------------------------------------------------------------------------------------------
-----------------------------+
| "ad5c3cf1-541a-44f8-a19b-28bc36030914" | "server-3.neo4j.svc.cluster.local:7687" | "Enabled" |
"Available" | ["system", "neo4j"] |
| "cbdebc59-64c2-4542-a041-24a1f051e64f" | "server-1.neo4j.svc.cluster.local:7687" | "Enabled" |
"Available" | ["system", "neo4j"] |
| "f37e98a7-15ec-4dc4-a6bf-df9e418a7488" | "server-2.neo4j.svc.cluster.local:7687" | "Enabled" |
"Available" | ["system", "neo4j"] |
+-------------------------------------------------------------------------------------------------------
-----------------------------+

3 rows
ready to start consuming query after 27 ms, results consumed after another 363 ms
```

4. Exit `cypher-shell`. Exiting `cypher-shell` automatically deletes the pod created to run it.

```
:exit;
```

```
Bye!
Session ended, resume using 'kubectl attach cypher-shell -c cypher-shell -i -t' command when the pod
is running
pod "cypher-shell" deleted
```

Access the Neo4j cluster using headless service

To allow for an application running inside Kubernetes to access the Neo4j cluster without using a specific server for bootstrapping, you need to install the *neo4j-cluster-headless-service* Helm chart. This will create a K8s Service with a DNS entry that includes all the Neo4j servers. You can use the created DNS entry to bootstrap drivers connecting to the cluster.

The headless service is a Kubernetes term for a service that has no ClusterIP. For more information, see the Kubernetes official documentation.

1. Install the headless service using the release name `headless`, *neo4j/neo4j-cluster-headless-service* Helm chart, and the name of your cluster as a value of the `neo4j.name` parameter.

> **Note:**
>
> Alternatively, you can create a *values.yaml* file with all the configurations for the service. To see what options are configurable on the *neo4j/neo4j-cluster-headless-service* Helm chart, use `helm show values neo4j/neo4j-headless-service`.

```
helm install headless neo4j/neo4j-headless-service --namespace neo4j --set neo4j.name=my-cluster
```

```
NAME: headless
LAST DEPLOYED: Wed Oct 26 13:11:14 2022
NAMESPACE: neo4j
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Thank you for installing neo4j-cluster-headless-service.

Your release "headless" has been installed in namespace "neo4j".

Once rollout is complete you can connect to your Neo4j cluster using "neo4j://headless-
neo4j.neo4j.svc.cluster.local:7687". Try:

  $ kubectl run --rm -it --namespace "neo4j" --image "neo4j:5.26.25-enterprise" cypher-shell \
     -- cypher-shell -a "neo4j://headless-neo4j.neo4j.svc.cluster.local:7687"

Graphs are everywhere!
```

1. Check that the `headless` service is available:

```
export NEO4J_NAME=my-cluster
kubectl get service ${NEO4J_NAME}-headless
```

```
NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)                      AGE
my-cluster-headless   ClusterIP   None         <none>        7474/TCP,7687/TCP   113s
```

2. Use `kubectl describe service` to see the service details:

```
kubectl describe service ${NEO4J_NAME}-headless
```

```
Name:              my-cluster-headless
Namespace:         neo4j
Labels:            app=my-cluster
                   app.kubernetes.io/managed-by=Helm
                   helm.neo4j.com/neo4j.name=my-cluster
Annotations:       cloud.google.com/neg: {"ingress":true}
                   meta.helm.sh/release-name: headless
                   meta.helm.sh/release-namespace: neo4j
Selector:          app=my-cluster,helm.neo4j.com/neo4j.loadbalancer=include
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                None
IPs:               None
Port:              http  7474/TCP
TargetPort:        7474/TCP
Endpoints:         10.24.0.131:7474,10.24.1.3:7474,10.24.1.67:7474
Port:              https  7473/TCP
TargetPort:        7473/TCP
Endpoints:         10.24.0.131:7473,10.24.1.3:7473,10.24.1.67:7473
Port:              tcp-bolt  7687/TCP
TargetPort:        7687/TCP
Endpoints:         10.24.0.131:7687,10.24.1.3:7687,10.24.1.67:7687
Session Affinity:  None
Events:            <none>
```

You should see three “endpoints” for each port in the service — these are the IP addresses of the three Neo4j servers. These endpoints are contacted to bootstrap the drivers used by applications running in

Kubernetes. The drivers will use them to obtain the initial routing table.

3. Run `cypher-shell` in another pod and connect to the cluster servers via the headless service:

```
kubectl run --rm -it --namespace "neo4j" --image "neo4j:5.26.25-enterprise" cypher-shell -- cypher-
shell -a "neo4j://my-cluster-headless.neo4j.svc.cluster.local:7687" -u neo4j -p "my-password"
```

```
If you don't see a command prompt, try pressing enter.
Connected to Neo4j using Bolt protocol version 5 at neo4j://headless-
neo4j.default.svc.cluster.local:7687 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
```

4. Run the Cypher command `SHOW DATABASES` to verify that all cluster servers are online.

```
SHOW DATABASES;
```

```
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+
| name     | type       | aliases | access       | address                                | role
| writer | requestedStatus | currentStatus | statusMessage | default | home  | constituents |
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+
| "neo4j"  | "standard" | []      | "read-write" | "server-3.neo4j.svc.cluster.local:7687" | "primary"
| TRUE   | "online"        | "online"      | ""           | TRUE    | TRUE  | []           |
| "neo4j"  | "standard" | []      | "read-write" | "server-2.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""           | TRUE    | TRUE  | []           |
| "neo4j"  | "standard" | []      | "read-write" | "server-1.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""           | TRUE    | TRUE  | []           |
| "system" | "system"   | []      | "read-write" | "server-3.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""           | FALSE   | FALSE | []           |
| "system" | "system"   | []      | "read-write" | "server-2.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""           | FALSE   | FALSE | []           |
| "system" | "system"   | []      | "read-write" | "server-1.neo4j.svc.cluster.local:7687" | "primary"
| TRUE   | "online"        | "online"      | ""           | FALSE   | FALSE | []           |
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+
6 rows
ready to start consuming query after 4 ms, results consumed after another 42 ms
```

5. Exit `cypher-shell`. Exiting `cypher-shell` automatically deletes the pod created to run it.

```
:exit;
```

```
Bye!
Session ended, resume using 'kubectl attach cypher-shell -c cypher-shell -i -t' command when the pod
is running
pod "cypher-shell" deleted
```

Access the Neo4j cluster from outside Kubernetes

By default, server-side routing is used for accessing a Neo4j cluster from outside Kubernetes.

Access the Neo4j cluster using a load balancer and Cypher Shell

A LoadBalancer service is created to access a Neo4j cluster from outside Kubernetes.

1. Check that the LoadBalancer service is available using the `neo4j.name` used for installation:

```
export NEO4J_NAME=my-cluster
kubectl get service ${NEO4J_NAME}-lb-neo4j
```

```
NAME                  TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)
AGE
my-cluster-lb-neo4j   LoadBalancer   10.28.12.119   82.21.42.42
7474:32169/TCP,7473:32145/TCP,7687:32624/TCP   2m1s
```

2. Use `kubectl describe service` to see the service details:

```
kubectl describe service ${NEO4J_NAME}-lb-neo4j
```

```
Name:                     my-cluster-lb-neo4j
Namespace:                neo4j
Labels:                   app=my-cluster
                          app.kubernetes.io/managed-by=Helm
                          helm.neo4j.com/neo4j.name=my-cluster
                          helm.neo4j.com/service=neo4j
Annotations:              cloud.google.com/neg: {"ingress":true}
                          meta.helm.sh/release-name: server-1
                          meta.helm.sh/release-namespace: neo4j
Selector:                 app=my-
cluster,helm.neo4j.com/clustering=true,helm.neo4j.com/neo4j.loadbalancer=include
Type:                     LoadBalancer
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.28.12.119
IPs:                      10.28.12.119
LoadBalancer Ingress:     82.21.42.42
Port:                     http  7474/TCP
TargetPort:               7474/TCP
NodePort:                 http  32169/TCP
Endpoints:                10.24.0.131:7474,10.24.1.3:7474,10.24.1.67:7474
Port:                     https  7473/TCP
TargetPort:               7473/TCP
NodePort:                 https  32145/TCP
Endpoints:                10.24.0.131:7473,10.24.1.3:7473,10.24.1.67:7473
Port:                     tcp-bolt  7687/TCP
TargetPort:               7687/TCP
NodePort:                 tcp-bolt  32624/TCP
Endpoints:                10.24.0.131:7687,10.24.1.3:7687,10.24.1.67:7687
Session Affinity:         None
External Traffic Policy:  Local
HealthCheck NodePort:     30621
Events:
  Type    Reason                Age    From                Message
  ----    ------                ----   ----                -------
  Normal  EnsuringLoadBalancer  3m11s  service-controller  Ensuring load balancer
  Normal  EnsuredLoadBalancer   2m36s  service-controller  Ensured load balancer
```

The load-balancer service can send requests to all cluster servers.

3. Run `cypher-shell` from the local machine and connect to the `LoadBalancer Ingress` address, in the example `82.21.42.42`:

```
./cypher-shell -a neo4j://82.21.42.42 -u neo4j -p my-password
```

```
If you don't see a command prompt, try pressing enter.
Connected to Neo4j using Bolt protocol version 5 at neo4j://82.21.42.42:7687 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
```

```
Note that Cypher queries must end with a semicolon.
```

4. Run the Cypher command `SHOW DATABASES` to verify that all cluster members are online:

```
SHOW DATABASES;
```

```
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+
| name     | type       | aliases | access       | address                                  | role
| writer | requestedStatus | currentStatus | statusMessage | default | home  | constituents |
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+
| "neo4j"  | "standard" | []      | "read-write" | "server-3.neo4j.svc.cluster.local:7687" | "primary"
| TRUE   | "online"        | "online"      | ""           | TRUE    | TRUE  | []           |
| "neo4j"  | "standard" | []      | "read-write" | "server-2.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""           | TRUE    | TRUE  | []           |
| "neo4j"  | "standard" | []      | "read-write" | "server-1.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""           | TRUE    | TRUE  | []           |
| "system" | "system"   | []      | "read-write" | "server-3.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""           | FALSE   | FALSE | []           |
| "system" | "system"   | []      | "read-write" | "server-2.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""           | FALSE   | FALSE | []           |
| "system" | "system"   | []      | "read-write" | "server-1.neo4j.svc.cluster.local:7687" | "primary"
| TRUE   | "online"        | "online"      | ""           | FALSE   | FALSE | []           |
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+

6 rows
ready to start consuming query after 110 ms, results consumed after another 109 ms
```

5. Run the Cypher command `SHOW SERVERS` to verify that all cluster members are enabled:

```
SHOW SERVERS;
```

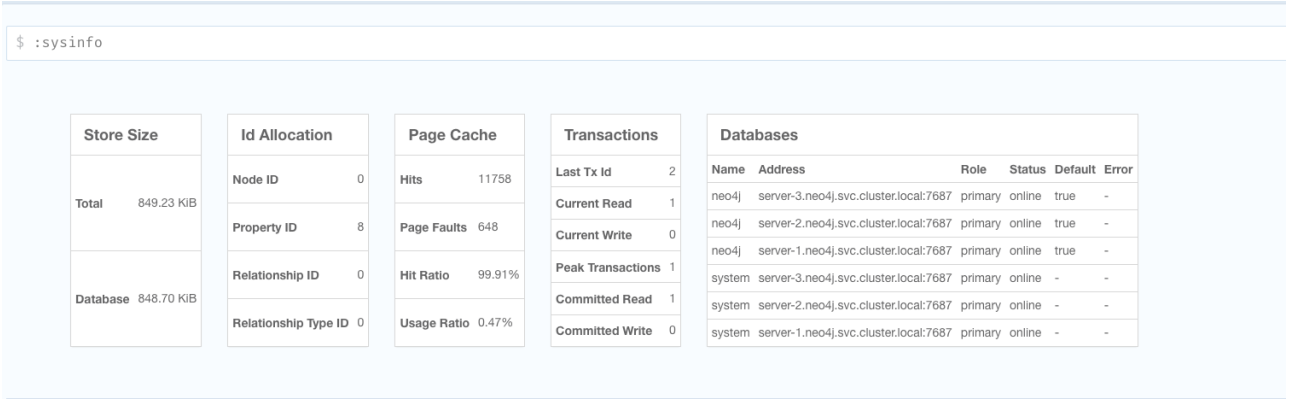
```
+-----------------------------------------------------------------------------------------------------
-----------------------------+
| name                                 | address                                | state     |
health      | hosting            |
+-----------------------------------------------------------------------------------------------------
-----------------------------+
| "ad5c3cf1-541a-44f8-a19b-28bc36030914" | "server-3.neo4j.svc.cluster.local:7687" | "Enabled" |
"Available" | ["system", "neo4j"] |
| "cbdebc59-64c2-4542-a041-24a1f051e64f" | "server-1.neo4j.svc.cluster.local:7687" | "Enabled" |
"Available" | ["system", "neo4j"] |
| "f37e98a7-15ec-4dc4-a6bf-df9e418a7488" | "server-2.neo4j.svc.cluster.local:7687" | "Enabled" |
"Available" | ["system", "neo4j"] |
+-----------------------------------------------------------------------------------------------------
-----------------------------+

3 rows
ready to start consuming query after 27 ms, results consumed after another 363 ms
```

You can see that the nodes are advertising their internal addresses, but since you connected without using internal addresses, you use server-side routing.

Access the Neo4j cluster using a load balancer and Neo4j Browser

1. Open a web browser and point it to the `LoadBalancer Ingress` address and port `7474`, in this example, *http://82.21.42.42:7474/browser*.
2. Once connected, verify that all databases are up and running using `:sysinfo` in the Browser Editor:

```
:sysinfo
```



Uninstall Neo4j cluster and clean up resources

Uninstall all Neo4j Helm deployments

1. Uninstall each of the cluster servers and the services using their Helm release names:

```
helm uninstall server-1 server-2 server-3 headless
```

```
release "server-1" uninstalled
release "server-2" uninstalled
release "server-3" uninstalled
release "headless" uninstalled
```

Fully remove all the data and resources

Uninstalling the Helm releases does not remove the created resources and data. Therefore, after uninstalling the helm deployments, you also have to delete all the data and resources.

1. Delete all persistent volume claims in the `neo4j` namespace:

```
kubectl delete pvc --all --namespace neo4j
```

2. Delete the entire Kubernetes cluster in your cloud provider:

```
gcloud container clusters delete my-neo4j-gke-cluster
```

```
eksctl delete cluster --name=my-neo4j-eks-cluster
```

```
az aks delete --name my-neo4j-aks-cluster --resource-group <MyResourceGroup>
```

Quickstart: Deploy a Neo4j cluster for analytic queries

The feature is available in the Neo4j Helm chart from version 5.14.

This quickstart shows how to configure and deploy a special Neo4j cluster that comprises one primary server and N secondary servers to support analytic queries. The primary server handles the transaction workloads, whereas the secondary servers are configured with the Neo4j Graph Data Science library (GDS) and are used only for the analytic workload.
Information on using GDS in a cluster can be found in the Neo4j Graph Data Science library documentation.

The cluster is deployed to a cloud or a local Kubernetes cluster using the Neo4j Helm chart.

Prerequisites

Before you can deploy a Neo4j cluster on Kubernetes, you need to have:

- A Kubernetes cluster running and the `kubectl` command-line tool installed and configured to communicate with your cluster. For more information, see Quickstart: Deploy a cluster → Prerequisites.
- A valid license for Neo4j Enterprise Edition. For more information, see Install GDS Enterprise Edition (EE) and Bloom plugins.
- The latest version of the Neo4j Helm chart repository.
- (Optional) A valid license for GDS Enterprise Edition. To install a licensed plugin, you must provide the license files in a Kubernetes secret. For more information, see Install GDS Enterprise Edition (EE) and Bloom plugins.

Create a value YAML file for each type of server

To set up a Neo4j cluster for analytic queries, you need to create a value YAML file for each type of server, primary and secondary. For example:

Create a value YAML file for the primary server, for example, *primary-value.yaml*:

```
neo4j:
  name: analytics-cluster
  acceptLicenseAgreement: "yes"
  edition: enterprise
  password: my-password
volumes:
  data:
    mode: defaultStorageClass

# Disable the Neo4j load balancer and enable the internal service so that the servers can access
each other:
```

```
services:
  neo4j:
    enabled: false
  internals:
    enabled: true

# Enable the analytics cluster and set the type to primary:
analytics:
  enabled: true
  type:
    name: primary
```

Create a value YAML file for the secondary servers, for example, *secondary-gds.yaml*. The password must be the same as for the primary server. If you are using GDS Enterprise Edition, you also need to create a secret with the license file and mount it as the */licenses* volume mount. For more information on how to create a secret, see Install GDS Enterprise Edition (EE) and Bloom plugins.

```
neo4j:
  name: analytics-cluster
  acceptLicenseAgreement: "yes"
  edition: enterprise
  password: my-password
volumes:
  data:
    mode: defaultStorageClass
  # Define the volume mount for the license file:
  licenses:
    disableSubPathExpr: true
    mode: volume
    volume:
      secret:
        secretName: gds-license
        items:
          - key: gds.license
            path: gds.license

# Set the environment variables to download the plugins:
env:
  NEO4J_PLUGINS: '["graph-data-science"]'

# Set the configuration for the plugins directory and the mount for the license file:
config:
  gds.enterprise.license_file: "/licenses/gds.license"
  server.directories.plugins: "plugins"

# Disable the Neo4j load balancer and enable the internal service so that the servers can access
each other:
services:
  neo4j:
    enabled: false
  internals:
    enabled: true

# Enable the analytics cluster and set the type to secondary:
analytics:
  enabled: true
  type:
    name: secondary
```

For all available options, see Customizing a Neo4j Helm chart.

Install the servers

1. Install a single Neo4j server using the *neo4j-primary.yaml* file, created in the previous section:

```
helm install primary neo4j/neo4j -f /path/to/neo4j-primary.yaml
```

2. Install the first secondary server using the *secondary-gds.yaml* file, created in the previous section:

```
helm install gds1 neo4j/neo4j -f /path/to/secondary-gds.yaml
```

3. Repeat step 2 to deploy a second secondary server. Use a different name, for example, *gds2*.

Verify that the GDS library is installed and licensed

1. Connect to each of the `gds` pods using the `kubectl exec` command:

```
kubectl exec -it gds1-0 -- bash
```

2. From the *bin* folder, connect to the `system` database of the `gds1` server using the `cypher-shell` command:

```
cypher-shell -u neo4j -p my-password -d system -a bolt://gds1-internals.default.svc.cluster.local:7687
```

3. Run the following Cypher function to verify that the GDS library is installed:

```
RETURN gds.version();
```

4. Call `gds.isLicensed()` to verify that the GDS library is licensed:

```
RETURN gds.isLicensed();
```

   The returned value must be `true`.

Verify the cluster formation

To verify that the cluster is deployed and running, you can install a load balancer and access Neo4j from the Neo4j Browser.

1. Deploy a Neo4j load balancer to the same namespace as the Neo4j cluster:

```
helm install lb neo4j/neo4j-load-balancer --set neo4j.name="analytics-cluster"
```

2. When deployed, copy the `EXTERNAL_IP` of the LoadBalancer service. For more information, see Access the Neo4j cluster from outside Kubernetes.
3. In a web browser, open the Neo4j Browser at *http://EXTERNAL_IP:7474/browser* and log in using the password you have configured in your values YAML files.
4. Verify that the cluster is deployed and running:

```
SHOW SERVERS;
```

```
+-----------------------------------------------------------------------------------------------------
-------------------------------------+
| name                                 | address                                 | state     |
health      | hosting                  |
+-----------------------------------------------------------------------------------------------------
-------------------------------------+
| "16cd6e9c-aa5a-4737-8ed5-e0df36ce52d3" | "gds2.default.svc.cluster.local:7687"    | "Free"    |
"Available" | ["system"]               |
| "bafbe254-a8a2-498d-9b60-6b3fd0124045" | "primary.default.svc.cluster.local:7687" | "Enabled" |
"Available" | ["neo4j", "system"]        |
| "f1478d5d-1718-4430-a9b6-26fe9695ca30" | "gds1.default.svc.cluster.local:7687"    | "Free"    |
"Available" | ["system"]               |
+-----------------------------------------------------------------------------------------------------
-------------------------------------+
```

The output shows that the secondary servers are in state `free` and host only the `system` database.

Enable the secondary servers to support analytic queries

To support analytic queries on the secondary servers, you need to enable them and change the `neo4j` database topology to include them.

1. In Neo4j Browser, enable the secondary servers to support analytic queries:

```
ENABLE SERVER "f1478d5d-1718-4430-a9b6-26fe9695ca30";
ENABLE SERVER "16cd6e9c-aa5a-4737-8ed5-e0df36ce52d3";
```

2. Alter the database topology to include the secondary servers:

```
ALTER DATABASE neo4j SET TOPOLOGY 1 PRIMARY 2 SECONDARY;
```

3. Check the status of the `neo4j` database to confirm the change:

```
SHOW DATABASE neo4j;
```

```
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+
| name    | type       | aliases | access       | address                                  | role
| writer | requestedStatus | currentStatus | statusMessage | default | home | constituents |
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+
| "neo4j" | "standard" | []      | "read-write" | "primary.default.svc.cluster.local:7687" | "primary"
| TRUE   | "online"        | "online"      | ""            | TRUE    | TRUE | []          |
| "neo4j" | "standard" | []      | "read-write" | "gds1.default.svc.cluster.local:7687"    |
"secondary" | FALSE  | "online"        | "online"      | ""            | TRUE    | TRUE | []
|
| "neo4j" | "standard" | []      | "read-write" | "gds2.default.svc.cluster.local:7687"    |
"secondary" | FALSE  | "online"        | "online"      | ""            | TRUE    | TRUE | []
|
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+
```

4. Check the status of all servers:

```
SHOW SERVERS;
```

```
+-----------------------------------------------------------------------------------------------------------
-------------------------------------+
| name                                 | address                                 | state     |
health      | hosting                    |
+-----------------------------------------------------------------------------------------------------------
-------------------------------------+
| "16cd6e9c-aa5a-4737-8ed5-e0df36ce52d3" | "gds2.default.svc.cluster.local:7687"    | "Enabled" |
"Available" | ["neo4j", "system"]        |
| "bafbe254-a8a2-498d-9b60-6b3fd0124045" | "primary.default.svc.cluster.local:7687" | "Enabled" |
"Available" | ["neo4j", "system"]        |
| "f1478d5d-1718-4430-a9b6-26fe9695ca30" | "gds1.default.svc.cluster.local:7687"    | "Enabled" |
"Available" | ["neo4j", "system"]        |
+-----------------------------------------------------------------------------------------------------------
-------------------------------------+
```

Volume mounts and persistent volumes

Neo4j Helm chart uses volume mounts and persistent volumes to manage the storage of data and other Neo4j files.

Volume mounts

A *volume mount* is part of a Kubernetes Pod spec that describes how and where a volume is mounted within a container.

The Neo4j Helm chart creates the following volume mounts:

- `backups` mounted at */backups*
- `data` mounted at */data*
- `import` mounted at */import*
- `licenses` mounted at */licenses*
- `logs` mounted at */logs*
- `metrics` mounted at */metrics* (Neo4j Community Edition does not generate `metrics`.)

It is also possible to specify a `plugins` volume mount (mounted at */plugins*), but this is not created by the default Helm chart. For more information, see Add plugins using a plugins volume.

Persistent volumes

`PersistentVolume` (PV) is a storage resource in the Kubernetes cluster that has a lifecycle independent of any individual pod that uses the PV.
`PersistentVolumeClaim` (PVC) is a request for a storage resource by a user. PVCs consume PV resources. For more information about what PVs are and how they work, see the Kubernetes official documentation.

The type of PV used and its configuration can have a significant effect on the performance of Neo4j. Some PV types are not suitable for use with Neo4j at all.

The volume type used for the `data` volume mount is particularly important. Neo4j supports the following

PV types for the `data` volume mount:

- `persistentVolumeClaim`
- `hostPath` when using Docker Desktop [1].

Neo4j `data` volume mount does not support `azureFile` and `nfs`.

> **Note:**
>
> `awsElasticBlockStore`, `azureDisk`, `gcePersistentDisk` are now deprecated volume types in Kubernetes and their use is no longer supported by the Neo4j Helm chart. If you currently use one of these volume types, consult your Kubernetes vendor's documentation on migrating to Container Storage Interface (CSI) driver-based storage.

For volume mounts other than the `data` volume mount, generally, all PV types are presumed to work.

> **Note:**
>
> `hostPath`, `local`, and `emptyDir` types are expected to perform well, provided suitable underlying storage, such as SSD, is used. However, these volume types have operational limitations and are not recommended.
>
> It is also not recommended to use an HDD or cloud storage, such as AWS S3 mounted as a drive.

Mapping volume mounts to persistent volumes

By default, the Neo4j Helm chart uses a single PV, named `data`, to support volume mounts.

The volume used for each volume mount can be changed by modifying the `volumes.<volume name>` object in the Helm chart values.

The Neo4j Helm chart `volumes` object supports different modes, such as `dynamic`, `share`, `defaultStorageClass`, `volume`, `selector`, and `volumeClaimTemplate`. From Neo4j 5.10, you can also set a label on creation for the volumes with mode `dynamic`, `defaultStorageClass`, `selector`, and `volumeClaimTemplate`, which can be used to filter the PVs that are used for the volume mount.

`mode: dynamic`

Recommended

Description

Dynamic volumes are recommended for most production workloads due to ease of management. The volume mount is backed by a PV that Kubernetes dynamically provisions using a dedicated `StorageClass`. The `StorageClass` is specified in the `storageClassName` field.

Example

The data volume uses a dedicated storage class:

storage-class-values.yaml

```
neo4j:
  name: standalone-with-storage-class
volumes:
  data:
    labels:
        data: "true"
    mode: dynamic
    dynamic:
      storageClassName: "neo4j-data"
      requests:
        storage: 10Gi
```

See Provision a PV using a dedicated `StorageClass` for more information.

`mode: share`

Description

The volume mount shares the underlying volume from one of the other volume objects.

Example

The `logs` volume mount uses the `data` volume (this is the default behavior).

```
volumes:
  logs:
    mode: "share"
    share:
      name: "data"
```

`mode: defaultStorageClass`

Description

The volume mount is backed by a PV that Kubernetes dynamically provisions using the default `StorageClass`.

Example

A dynamically provisioned `data` volume with a size of `10Gi`.

```
volumes:
  data:
    labels:
        data: "true"
    mode: "defaultStorageClass"
    defaultStorageClass:
      requests:
        storage: 10Gi
```

Note:

For the `data` volume, if `requests.storage` is not set, `defaultStorageClass` defaults to a `10Gi` volume. For all other volumes, `defaultStorageClass.requests.storage` must be set

explicitly when using `defaultStorageClass` mode.

`mode: volume`

Description

A complete Kubernetes `volume` object can be specified for the volume mount. Generally, volumes specified in this way have to be manually provisioned.

`volume` can be any valid Kubernetes volume type. This mode is typically used to mount a pre-existing Persistent Volume Claim (PVC).

For details on how to specify `volume` objects, see the Kubernetes documentation.

Set file permissions on mounted volumes

The Neo4j Helm chart supports an additional field not present in normal Kubernetes `volume` objects: `setOwnerAndGroupWritableFilePermissions: true|false`. If set to `true`, an `initContainer` will be run to modify the file permissions of the mounted volume, so that the contents can be written and read by the Neo4j process. This is to help with certain volume implementations that are not aware of the `SecurityContext` set on pods using them.

Example - reference an existing PersistentVolume

The `backups` volume mount is backed by the specified PVC. When this method is used, the `persistentVolumeClaim` object must already exist.

```
volumes:
  backups:
    mode: volume
    volume:
      persistentVolumeClaim:
        claimName: my-neo4j-pvc
```

`mode: selector`

Description

The volume to use is chosen from the existing PVs based on the provided `selector` object and a PVC that is dynamically generated.

If no matching PVs exist, the Neo4j pod will be unable to start. To match, a PV must have the specified `StorageClass`, match the label `selectorTemplate`, and have sufficient storage capacity to meet the requested storage amount.

Example

The `data` volume is chosen from the available volumes with the `neo4j` storage class and the label `developer: alice`.

```
volumes:
  import:
    labels:
        import: "true"
    mode: selector
    selector:
      storageClassName: "neo4j"
```

```
      requests:
        storage: 128Gi
      selectorTemplate:
        matchLabels:
          developer: "alice"
```

> **Note:**
>
> For the `data` volume, if `requests.storage` is not set, `selector` defaults to a `100Gi` volume. For all other volumes, `selector.requests.storage` must be set explicitly when using `selector` mode.

`mode: volumeClaimTemplate`

Description

A complete Kubernetes `volumeClaimTemplate` object is specified for the volume mount. Volumes specified in this way are dynamically provisioned.

Example - provision Neo4j storage using a volume claim template

The data volume uses a dynamically provisioned PVC from the `default` storage class.

```
volumes:
  data:
    labels:
        data: "true"
    mode: volumeClaimTemplate
    volumeClaimTemplate:
      storageClassName: "default"
      accessModes:
        - ReadWriteOnce
      resources:
        requests:
          storage: 10Gi
```

> **Note:**
>
> In all cases, do not forget to set the `mode` field when customizing the volumes object. If not set, the default `mode` is used, regardless of the other properties set on the `volume` object.

Provision persistent volumes with Neo4j Helm chart

Provision persistent volumes dynamically

With the Neo4j Helm chart, you can provision a PV dynamically using the default or a custom `StorageClass`. To see a list of available storage classes in your Kubernetes cluster, run the following command:

```
kubectl get storageclass
```

Provision a PV using a dedicated `StorageClass`

Recommended

For production workloads, it is recommended to create a dedicated storage class for Neo4j, which uses the `Retain` reclaim policy. This is to avoid data loss when disks are deleted after removing the persistent volume resource.

Example: Deploy Neo4j using a dedicated `StorageClass`

The following example shows how to deploy a Neo4j server with a dynamically provisioned PV that uses a dedicated `storageClass`.

1. Create a dedicated storage class that uses the `Retain` reclaim policy:

1. Create a storage class in GKE that uses the `Retain` reclaim policy and `pd-ssd` high-performance SSD disks:

```
cat <<EOF | kubectl apply -f -
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: neo4j-data
provisioner: pd.csi.storage.gke.io
parameters:
  type: pd-ssd
reclaimPolicy: Retain
volumeBindingMode: WaitForFirstConsumer
allowVolumeExpansion: true
EOF
```

2. Check the storage class is created:

```
kubectl get storageclass neo4j-data
```

```
NAME                 PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION   AGE
neo4j-data           pd.csi.storage.gke.io   Retain          WaitForFirstConsumer
true                   7s
```

1. Create a storage class in EKS that uses the `Retain` reclaim policy and `gp3` high-performance SSD disks:

> **Note:**
>
> The EBS CSI Driver addon is required to provision EBS disks in EKS clusters. See the AWS documentation for instructions on installing the driver.

```
cat <<EOF | kubectl apply -f -
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
```

```
  name: neo4j-data
provisioner: ebs.csi.aws.com
parameters:
  type: gp3
reclaimPolicy: Retain
allowVolumeExpansion: true
volumeBindingMode: WaitForFirstConsumer
EOF
```

2. Check the storage class is created:

```
kubectl get storageclass neo4j-data
```

```
NAME            PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION   AGE
neo4j-data      ebs.csi.aws.com         Retain          WaitForFirstConsumer   true
2m41s
```

1. Create a storage class in AKS that uses the `Retain` reclaim policy and `pd-ssd` high-performance SSD disks:

```
cat <<EOF | kubectl apply -f -
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: neo4j-data
provisioner: disk.csi.azure.com
parameters:
  skuName: Premium_LRS
reclaimPolicy: Retain
volumeBindingMode: WaitForFirstConsumer
allowVolumeExpansion: true
EOF
```

2. Check the storage class is created:

```
kubectl get storageclass neo4j-data
```

```
NAME                 PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION   AGE
neo4j-data           disk.csi.azure.com      Retain          WaitForFirstConsumer
true                   7s
```

2. Install a Neo4j server with a data volume that uses the new storage class:

   a. Create a file *storage-class-values.yaml* that configures the data volume to use the new storage class:

   storage-class-values.yaml

```
neo4j:
  name: standalone-with-storage-class
volumes:
  data:
    mode: dynamic
```

```
    dynamic:
      storageClassName: "neo4j-data"
      requests:
        storage: 10Gi
```

b. Install a single Neo4j server:

```
helm install standalone-with-storage-class neo4j -f storage-class-values.yaml
```

c. When the installation completes, verify that a PVC has been created:

```
kubectl get pvc
```

```
NAME                                     STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS   AGE
data-standalone-with-storage-class-0   Bound    pvc-5d400f06-f99f-43ac-bf37-6079d692eaac   10Gi
RWO            neo4j-data     23m
```

3. Clean up the resources:

   The storage class uses the `Retain` retention policy, meaning the disk will **not** be deleted after removing the PVC. To delete the disk, patch the PVC to use the `Delete` retention policy and delete the PVC:

```
export pv_name=$(kubectl get pvc data-standalone-with-storage-class-0 -o jsonpath
='{.spec.volumeName}')

kubectl patch pv $pv_name -p '{"spec":{"persistentVolumeReclaimPolicy": "Delete"}}'

kubectl delete pvc data-standalone-with-storage-class-0
```

> **Note:**
>
> For the `data` volume, if `requests.storage` is not set, `dynamic` defaults to a `100Gi` volume. For all other volumes, `dynamic.requests.storage` must be set explicitly when using `dynamic` mode.

Provision a PV using `defaultStorageClass`

Using the default `StorageClass` of the running Kubernetes cluster is the quickest way to spin up and run Neo4j for simple tests, handling small amounts of data. However, it is not recommended for large amounts of data, as it may lead to performance issues.

Example: Deploy Neo4j using `defaultStorageClass`

The following example shows how to deploy a Neo4j server with a dynamically provisioned PV that uses the default `StorageClass`.

1. Create a file *default-storage-class-values.yaml* that configures the data volume to use the default `StorageClass` and a storage size `100Gi`:

storage-class-values.yaml

```
volumes:
  data:
    mode: "defaultStorageClass"
    defaultStorageClass:
      requests:
        storage: 100Gi
```

2. Install a single Neo4j server:

```
helm install standalone-with-default-storage-class neo4j -f default-storage-class-values.yaml
```

Provision persistent volumes manually

Optionally, the Helm chart can use manually created disks for Neo4j storage. This installation option has more steps than using dynamic volumes, but it does provide more control over how disks are provisioned.

The instructions for the manual provisioning of PVs vary according to the type of PV being used and the underlying infrastructure. In general, there are two steps:

1. Create the disk/volume to be used for storage in the underlying infrastructure. For example:
   - If using a `csi` volume — create the Persistent Disk using the cloud provider CLI or console.
   - If using a `hostPath` volume — on the host node, create the path (directory).
2. Create a PV in Kubernetes that references the underlying resource created in step 1.
   a. Ensure that the created PV's `app` label matches the name of the Neo4j Helm release.
   b. Ensure that the created PV's `capacity.storage` matches the storage available on the underlying infrastructure.

If no suitable PV or PVC exists, the Neo4j pod will not start.

Provision a PV for Neo4j Storage using a PV selector

The Neo4j StatefulSet can select a persistent volume to use based on its labels. A Neo4j Helm release uses only manually provisioned PVs that have:

- `storageClassName` that uses the provisioner `kubernetes.io/no-provisioner`.
- An `app` label — set in their metadata, which matches the name of the `neo4j.name` value of the Helm installation.
- Sufficient storage capacity — the PV capacity must be greater than or equal to the value of `volumes.data.selector.requests.storage` set for the Neo4j Helm release (default is `100Gi`).

Note:

The neo4j/neo4j-persistent-volume Helm chart provides a convenient way to provision the persistent volume.

Example: Deploy Neo4j using a selector volume

The following example shows how to deploy Neo4j using a selector volume.

1. Create a file *persistent-volume-selector.yaml* that configures the data volume to use a selector:

   storage-class-values.yaml

   ```
   neo4j:
     name: volume-selector
   volumes:
     data:
       mode: selector
       selector:
         storageClassName: "manual"
         accessModes:
           - ReadWriteOnce
         requests:
           storage: 10Gi
   ```

2. Export environment variables to be used by the commands:

   ```
   export RELEASE_NAME=volume-selector
   export GCP_ZONE="$(gcloud config get compute/zone)"
   export GCP_PROJECT="$(gcloud config get project)"
   ```

3. Create the disks to be used by the persistent volume:

   ```
   gcloud compute disks create --size 10Gi --type pd-ssd "${RELEASE_NAME}"
   ```

4. Use the *neo4j/neo4j-persistent-volume* chart to configure the persistent volume. This command will create a persistent volume and a manual storage class that uses the `kubernetes.io/no-provisioner` provisioner.

   ```
   helm install "${RELEASE_NAME}"-disk neo4j/neo4j-persistent-volume \
          --set neo4j.name="${RELEASE_NAME}" \
          --set data.driver=pd.csi.storage.gke.io \
          --set data.storageClassName="manual" \
          --set data.reclaimPolicy="Delete" \
          --set data.createPvc=false \
          --set data.createStorageClass=true \
          --set data.volumeHandle="projects/${GCP_PROJECT}/zones/${GCP_ZONE}/disks/
   ${RELEASE_NAME}" \
          --set data.capacity.storage=10Gi
   ```

5. Now install Neo4j using the `persistent-volume-selector.yaml` created earlier:

   ```
   helm install "${RELEASE_NAME}" neo4j/neo4j -f persistent-volume-selector.yaml
   ```

6. Clean up the helm installation and disks created for the example:

   ```
   helm uninstall ${RELEASE_NAME} ${RELEASE_NAME}-disk
   kubectl delete pvc data-${RELEASE_NAME}-0
   gcloud compute disks delete ${RELEASE_NAME} --quiet
   ```

The EBS CSI Driver addon is required to provision EBS disks in EKS clusters. You can run the command `kubectl get daemonset ebs-csi-node -n kube-system` to check if it is installed See the AWS Documentation for instructions on installing the driver.

1. Create a file `persistent-volume-selector.yaml` that configures the data volume to use a selector:

   storage-class-values.yaml

   ```
   neo4j:
     name: volume-selector
   volumes:
     data:
       mode: selector
       selector:
         storageClassName: "manual"
         accessModes:
           - ReadWriteOnce
         requests:
           storage: 10Gi
   ```

2. Export environment variables to be used by the commands:

   ```
   readonly RELEASE_NAME=volume-selector
   readonly AWS_ZONE={availability zone of EKS cluster}
   ```

3. Create the disks to be used by the persistent volume:

   ```
   export volumeId=$(aws ec2 create-volume \
                       --availability-zone="${AWS_ZONE}" \
                       --size=10 \
                       --volume-type=gp3 \
                       --tag-specifications 'ResourceType=volume,Tags=[{Key=volume,Value='
   "${RELEASE_NAME}"'}]' \
                       --no-cli-pager \
                       --output text \
                       --query VolumeId)
   ```

4. Use the *neo4j/neo4j-persistent-volume* chart to configure the persistent volume. This command will create a persistent volume and a manual storage class that uses the `kubernetes.io/no-provisioner` provisioner.

   ```
   helm install "${RELEASE_NAME}"-disk neo4j-persistent-volume \
       --set neo4j.name="${RELEASE_NAME}" \
       --set data.driver=ebs.csi.aws.com \
       --set data.reclaimPolicy="Delete" \
       --set data.createPvc=false \
       --set data.createStorageClass=true \
       --set data.volumeHandle="${volumeId}" \
       --set data.capacity.storage=10Gi
   ```

5. Now install Neo4j using the `persistent-volume-selector.yaml` created earlier:

   ```
   helm install "${RELEASE_NAME}" neo4j/neo4j -f persistent-volume-selector.yaml
   ```

6. Clean up the helm installation and disks created for the example:

```
helm uninstall ${RELEASE_NAME} ${RELEASE_NAME}-disk
    kubectl delete pvc data-${RELEASE_NAME}-0
    aws ec2 delete-volume --volume-id ${volumeId}
```

1. Create a file `persistent-volume-selector.yaml` that configures the data volume to use a selector:

   storage-class-values.yaml

   ```
   neo4j:
     name: volume-selector
   volumes:
     data:
       mode: selector
       selector:
         storageClassName: "manual"
         accessModes:
           - ReadWriteOnce
         requests:
           storage: 10Gi
   ```

2. Export environment variables to be used by the commands:

   ```
   readonly AKS_CLUSTER_NAME={AKS Cluster name}
   readonly AZ_RESOURCE_GROUP={Resource group of cluster}
   readonly AZ_LOCATION={Location of cluster}
   ```

3. Create the disks to be used by the persistent volume:

   ```
   export node_resource_group=$(az aks show --resource-group "${AZ_RESOURCE_GROUP}" --name
   "${AKS_CLUSTER_NAME}" --query nodeResourceGroup -o tsv)
   export disk_id=$(az disk create --name "${RELEASE_NAME}" --size-gb "10" --max-shares 1
   --resource-group "${node_resource_group}" --location ${AZ_LOCATION} --output tsv --query
   id)
   ```

4. Use the *neo4j/neo4j-persistent-volume* chart to configure the persistent volume. This command will create a persistent volume and a manual storage class that uses the `kubernetes.io/no-provisioner` provisioner.

   ```
   helm install "${RELEASE_NAME}"-disk neo4j-persistent-volume \
       --set neo4j.name="${RELEASE_NAME}" \
       --set data.driver=disk.csi.azure.com \
       --set data.storageClassName="manual" \
       --set data.reclaimPolicy="Delete" \
       --set data.createPvc=false \
       --set data.createStorageClass=true \
       --set data.volumeHandle="${disk_id}" \
       --set data.capacity.storage=10Gi
   ```

5. Now install Neo4j using the `persistent-volume-selector.yaml` created earlier:

   ```
   helm install "${RELEASE_NAME}" neo4j/neo4j -f persistent-volume-selector.yaml
   ```

6. Clean up the helm installation and disks created for the example:

```
helm uninstall ${RELEASE_NAME} ${RELEASE_NAME}-disk
kubectl delete pvc data-${RELEASE_NAME}-0
az disk delete --name ${RELEASE_NAME} -y
```

Provision a PVC for Neo4j Storage

An alternative method for manual provisioning is to use a manually provisioned PVC. This is supported by the Neo4j Helm chart using the `volume` mode.

The *neo4j/neo4j-persistent-volume* Helm chart can be used to create a PV and PVC for a manually provisioned disk. A full example can be found in the Neo4j GitHub repository. For example, to use a pre-existing PVC called `my-neo4j-pvc` set these values:

```
volumes:
  data:
    mode: "volume"
    volume:
      persistentVolumeClaim:
        claimName: my-neo4j-pvc
```

Reuse a persistent volume

After uninstalling the Neo4j Helm chart, both the PVC and the PV remain and can be reused by a new install of the Helm chart. If you delete the PVC, the PV moves into a `Released` status and will not be reusable.

To be able to reuse the PV by a new install of the Neo4j Helm chart, remove its connection to the previous PVC:

1. Edit the PV by running the following command:

   ```
   kubectl edit pv <pv-name>
   ```

2. Remove the section `spec.claimRef`.
   The PV goes back to the `Available` status and can be reused by a new install of the Neo4j Helm chart.

Note:

The performance of Neo4j is very dependent on the latency, IOPS capacity, and throughput of the storage it is using. For the best performance of Neo4j, use the best available disks (e.g., SSD) and set IOPS throttling/quotas to high values. For some cloud providers, IOPS throttling is proportional to the size of the volume. In these cases, the best performance is achieved by setting the size of the volume based on the desired IOPS rather than the amount required for data storage.

Customizing a Neo4j Helm chart

Helm is different from “package managers”, such as `apt`, `yum`, and `npm`, because, in addition to installing

applications, Helm allows rich configuration of applications. The customized configuration should be expressed declaratively in a YAML formatted file, and then passed during installation.

Tip:

For more information, see Helm official documentation.

Important configuration parameters

The following is a list of the most important configuration parameters for the Neo4j Helm chart. For a full list of configuration parameters, run `helm show values`. See Create a custom *values.yaml* file.

`neo4j.name`

Starting from Neo4j 5.0.0, standalone servers and cluster servers have no distinction. This means a standalone server can be upgraded to a cluster by adding more servers. Therefore, the `neo4j.name` parameter, which value links together servers in a cluster, is mandatory, and the installation will fail if it is not specified. `neo4j.name` must be unique within a namespace.

`neo4j.resources`

The resources (CPU, memory) for the Neo4j container are configured by setting `neo4j.resources` object in the *values.yaml* file. In the resource *requests*, you can specify how much CPU and memory the Neo4j container needs, while in the resource *limits*, you can set a limit on these resources in case the container tries to use more resources than its *requests* allow.

```
neo4j:
  resources:
    requests:
     cpu: "1000m"
     memory: "2Gi"
    limits:
     cpu: "2000m"
     memory: "4Gi"
```

If no resource *requests* and resource *limits* are specified, the values set in the `resources` object are used for both the Neo4j container's resource *requests* and resource *limits*.

```
neo4j:
  resources:
    cpu: "2"
    memory: "5Gi"
```

Note:

The minimum for a Neo4j instance is `0.5` CPU and `2GB` memory.
If invalid or less than the minimum values are provided, Helm will throw an error, for example:

```
Error: template: neo4j-standalone/templates/_helpers.tpl:157:11: executing
"neo4j.resources.evaluateCPU" at <fail (printf "Provided cpu value %s is less than minimum.
```

```
\n %s" (.Values.neo4j.resources.cpu) (include "neo4j.resources.invalidCPUMessage" .))>:
error calling fail: Provided cpu value 0.25 is less than minimum.
 cpu value cannot be less than 0.5 or 500m
```

JVM heap and page cache

You configure Neo4j to use the memory provided to the container by setting the parameters `server.memory.heap.initial_size` and `server.memory.pagecache.size`. Combined, they must not exceed the memory configuration of the Neo4j container.
In Kubernetes, running processes in the Neo4j container, which exceed the configured memory limit are killed by the underlying operating system. Therefore, it is recommended to allow an additional 1GB of memory headroom so that `heap + pagecache + 1GB < available memory`.

For example, a 5GB container could be configured like this:

```
neo4j:
  resources:
    cpu: "2"
    memory: "5Gi"

# Neo4j configuration (yaml format)
config:
  server.memory.heap.initial_size: "3G"
  server.memory.heap.max_size: "3G"
  server.memory.pagecache.size: "1G"
```

`server.memory.pagecache.size` and `server.memory.heap.initial_size` are not the only settings available in Neo4j to manage memory usage. For full details of how to configure memory usage in Neo4j, see Performance - Memory Configuration.

`neo4j.password`

The password for the `neo4j` user.
You can set an initial password for accessing Neo4j in the *values.yaml* file.

Note:

You cannot use `neo4j` as the initial password as this is the default password.

If no initial password is set, the Neo4j helm chart will automatically generate one. (Make a note of it.) In cluster deployments, the same password must be set for all cluster members.

```
neo4j:
 # If not set or empty a random password will be generated
 password: ""
```

The password will be printed out in the Helm install output, unless `--set logInitialPassword=false` is used.

The initial Neo4j password is stored in a *Kubernetes Secret*. The password can be extracted from the Secret using this command:

```
kubectl get secret <release-name>-auth -oyaml | yq -r '.data.NEO4J_AUTH' | base64 -d
```

> **Tip:**
>
> To change the initial password, follow the steps in Reset the neo4j user password.
>
> Once you change the password in Neo4j, the password stored in *Kubernetes Secrets* will still exist but will no longer be valid.

`neo4j.edition` and `neo4j.acceptLicenseAgreement`

By default, the Helm chart installs Neo4j Community Edition.
If you want to install Neo4j Enterprise Edition, set the configuration parameters `edition: "enterprise"` and acknowledge license compliance by setting `neo4j.acceptLicenseAgreement` to `"yes"` if you have a valid license or to `"eval"` if you want to accept the Neo4j evaluation license.
For more information on how to obtain a valid license for Neo4j Enterprise Edition, see https://neo4j.com/licensing/ and link:https://neo4j.com/terms/licensing/, or use the form Contact Neo4j.

`volumes.data`

The `volumes.data` parameter maps the `data` volume mount of your Neo4j to the persistent volume that you have for that instance. For more information, see Volume mounts and persistent volumes.

`neo4j.minimumClusterSize`

By default, servers in a cluster can host primary and secondary databases. See the clustering documentation for more details.
`neo4j.minimumClusterSize` is set to 1 by default, which means the server starts without waiting for the other servers. When installing a cluster, you should set `neo4j.minimumClusterSize` to the number of desired members in the cluster. If you later decide to add an extra cluster server in excess of `neo4j.minimumClusterSize`, you need to manually enable it using the Cypher command `ENABLE SERVER`. For more information on enabling new servers, see Add a server to the cluster.

`config`

The Neo4j Helm chart does not use a `neo4j.conf` file. Instead, the Neo4j configuration is set in the Helm deployment's *values.yaml* file under the `config` object.

The `config` object should contain a string map of *neo4j.conf* setting name to value. For example, this `config` object configures the Neo4j metrics:

```
# Neo4j configuration (yaml format)
config:
  server.metrics.enabled: "true"
  server.metrics.csv.interval: "10s"
  server.metrics.csv.rotation.keep_number: "2"
  server.metrics.csv.rotation.compression: "NONE"
```

> **Note:**

> All Neo4j `config` values must be YAML strings. It is important to put quotes around the values, such as `"true"`, `"false"`, and `"2"`, so that they are handled correctly as strings.

All *neo4j.conf* settings are supported except for `server.jvm.additional`. Additional JVM settings can be set on the `jvm` object in the Helm deployment *values.yaml* file, as shown in the example:

```
# Jvm configuration for Neo4j
jvm:
  additionalJvmArguments:
  - "-XX:+HeapDumpOnOutOfMemoryError"
  - "-XX:HeapDumpPath=/logs/neo4j.hprof"
```

To find out more about configuring Neo4j and the *neo4j.conf* file, see Configuration and The neo4j.conf file.

`image.customImage`

The helm chart uses the official Neo4j Docker image that matches the version of the Helm chart. To configure the helm chart to use a different container image, set the `image.customImage` property in the *values.yaml* file.

This can be necessary when public container repositories are not accessible for security reasons. For example, this *values.yaml* file configures Neo4j to use `my-container-repository.io` as the container repository:

```
# neo4j-values.yaml
neo4j:
  password: "my-password"

image:
  customImage: "my-container-repository.io/neo4j:5-enterprise"
```

Other configuration options

Some examples of possible Neo4j configurations

- All Neo4j configuration (*neo4j.conf*) settings can be set directly on the `config` object in the *values.yaml* file.
- Neo4j can be configured to use SSL certificates contained in Kubernetes Secrets by modifying the `ssl` object in the values file. For more information, see Configure SSL.

Some examples of possible K8s configurations

- Configure (or disable completely) the Kubernetes LoadBalancer that exposes Neo4j outside the Kubernetes cluster by modifying the `services.neo4j.spec.loadBalancerIP` object in the *values.yml* file.
- Set the `securityContext` used by Neo4j Pods by modifying the `securityContext` object in the *values.yml* file.
- Configure manual persistent volume provisioning or set the `StorageClass` to be used as the Neo4j persistent storage.

Create a custom *values.yaml* file

1. Ensure your Neo4j Helm chart repository is up to date and get the latest charts. For more information, see Configure the Neo4j Helm chart repository.
2. To see what options are configurable on the Neo4j helm chart that you want to deploy, use `helm show values` and the Helm chart *neo4j/neo4j*. For example:

```
helm show values neo4j/neo4j
```

```
# Default values for Neo4j.
# This is a YAML-formatted file.

## @param nameOverride String to partially override common.names.fullname
nameOverride: ""
## @param fullnameOverride String to fully override common.names.fullname
fullnameOverride: ""
# disableLookups will disable all the lookups done in the helm charts
# This should be set to true when using ArgoCD since ArgoCD uses helm template and the helm lookups
will fail
# You can enable this when executing helm commands with --dry-run command
disableLookups: false

neo4j:
  # Name of your cluster
  name: ""
  # If password is not set or empty a random password will be generated during installation.
  # Ignored if `neo4j.passwordFromSecret` is provided
  password: ""

  # Existing secret to use for initial database password
  passwordFromSecret: ""

  # Neo4j Edition to use (community|enterprise)
  edition: "community"

  # Minimum number of machines initially required to form a clustered database. The StatefulSet will
not reach the ready state
  # until at least this many members have discovered each other. The default is 1 (standalone)
#  minimumClusterSize: 1
#

  # (Clustering only feature)
  # Neo4j operations allows you to enable servers (part of cluster) which are added outside the
minimumClusterSize
  # When the enableServer flag is set to true , an operations pod is created which performs the
following functions
    # fetch neo4j creds from the k8s secret (provided by user or created via helm chart)
    # Use the cluster ip created as part of the respective release to connect to Neo4j via Go Driver
    # Execute the ENABLE SERVER query and enable the server
  # The operations pod ends successfully if the server is enabled, or it was already enabled
  operations:
      enableServer: false
      image: "neo4j/helm-charts-operations:5.26.9"
      # protocol can be "neo4j or "neo4j+ssc" or "neo4j+s". Default set to neo4j
      # Note: Do not specify bolt protocol here...it will FAIL.
      protocol: "neo4j"
      labels: {}

  # set edition: "enterprise" to use Neo4j Enterprise Edition
  #
  # To use Neo4j Enterprise Edition you must have a Neo4j license agreement.
  #
  # More information is also available at: https://neo4j.com/licensing/
  # Email inquiries can be directed to: licensing@neo4j.com
  #
  # Set acceptLicenseAgreement: "yes" to confirm that you have a Neo4j license agreement.
  # Set acceptLicenseAgreement: "eval" to use Neo4j Enterprise Edition for evaluation purposes.
  acceptLicenseAgreement: "no"
  #
  # set offlineMaintenanceModeEnabled: true to restart the StatefulSet without the Neo4j process
running
  # this can be used to perform tasks that cannot be performed when Neo4j is running such as `neo4j-
admin dump`
```

```
  offlineMaintenanceModeEnabled: false
  #
  # set resources for the Neo4j Container. The values set will be used for both "requests" and
"limit".
  resources:
    cpu: "1000m"
    memory: "2Gi"

  #add labels if required
  labels:

# Volumes for Neo4j
volumes:
  data:

    #labels for data pvc on creation (Valid only when mode set to selector | defaultStorageClass |
dynamic | volumeClaimTemplate)
    labels: {}

    # Set it to true when you do not want to use the subPathExpr
    disableSubPathExpr: false

    # REQUIRED: specify a volume mode to use for data
    # Valid values are share|selector|defaultStorageClass|volume|volumeClaimTemplate|dynamic
    # To get up-and-running quickly, for development or testing, use "defaultStorageClass" for a
dynamically provisioned volume of the default storage class.
    mode: ""

    # Only used if mode is set to "selector"
    # Will attach to existing volumes that match the selector
    selector:
      storageClassName: "manual"
      accessModes:
        - ReadWriteOnce
      requests:
        storage: 100Gi
      # A helm template to generate a label selector to match existing volumes n.b. both
storageClassName and label selector must match existing volumes
      selectorTemplate:
        matchLabels:
          app: "{{ .Values.neo4j.name }}"
          helm.neo4j.com/volume-role: "data"

    # Only used if mode is set to "defaultStorageClass"
    # Dynamic provisioning using the default storageClass
    defaultStorageClass:
      accessModes:
        - ReadWriteOnce
      requests:
        storage: 10Gi

    # Only used if mode is set to "dynamic"
    # Dynamic provisioning using the provided storageClass
    dynamic:
      storageClassName: "neo4j"
      accessModes:
        - ReadWriteOnce
      requests:
        storage: 100Gi

    # Only used if mode is set to "volume"
    # Provide an explicit volume to use
    volume:
      # If set an init container (running as root) will be added that runs:
      #   `chown -R <securityContext.fsUser>:<securityContext.fsGroup>` AND `chmod -R g+rwx`
      # on the volume. This is useful for some filesystems (e.g. NFS) where Kubernetes fsUser or
fsGroup settings are not respected
      setOwnerAndGroupWritableFilePermissions: false

      # Example (using a specific Persistent Volume Claim)
      # persistentVolumeClaim:
      #   claimName: my-neo4j-pvc

    # Only used if mode is set to "volumeClaimTemplate"
    # Provide an explicit volumeClaimTemplate to use
    volumeClaimTemplate: {}
```

```
  # provide a volume to use for backups
  # n.b. backups will be written to /backups on the volume
  # any of the volume modes shown above for data can be used for backups
  backups:
    #labels for backups pvc on creation (Valid only when mode set to selector | defaultStorageClass |
dynamic | volumeClaimTemplate)
    labels: {}
    disableSubPathExpr: false
    mode: "share" # share an existing volume (e.g. the data volume)
    share:
      name: "data"

  # provide a volume to use for logs
  # n.b. logs will be written to /logs/$(POD_NAME) on the volume
  # any of the volume modes shown above for data can be used for logs
  logs:
    #labels for logs pvc on creation (Valid only when mode set to selector | defaultStorageClass |
dynamic | volumeClaimTemplate)
    labels: {}
    disableSubPathExpr: false
    mode: "share" # share an existing volume (e.g. the data volume)
    share:
      name: "data"

  # provide a volume to use for csv metrics (csv metrics are only available in Neo4j Enterprise
Edition)
  # n.b. metrics will be written to /metrics/$(POD_NAME) on the volume
  # any of the volume modes shown above for data can be used for metrics
  metrics:
    #labels for metrics pvc on creation (Valid only when mode set to selector | defaultStorageClass |
dynamic | volumeClaimTemplate)
    labels: {}
    disableSubPathExpr: false
    mode: "share" # share an existing volume (e.g. the data volume)
    share:
      name: "data"

  # provide a volume to use for import storage
  # n.b. import will be mounted to /import on the underlying volume
  # any of the volume modes shown above for data can be used for import
  import:
    #labels for import pvc on creation (Valid only when mode set to selector | defaultStorageClass |
dynamic | volumeClaimTemplate)
    labels: {}
    disableSubPathExpr: false
    mode: "share" # share an existing volume (e.g. the data volume)
    share:
      name: "data"

  # provide a volume to use for licenses
  # n.b. licenses will be mounted to /licenses on the underlying volume
  # any of the volume modes shown above for data can be used for licenses
  licenses:
    #labels for licenses pvc on creation (Valid only when mode set to selector | defaultStorageClass |
dynamic | volumeClaimTemplate)
    labels: {}
    disableSubPathExpr: false
    mode: "share" # share an existing volume (e.g. the data volume)
    share:
      name: "data"

# add additional volumes and their respective mounts
additionalVolumes: []
#  - name: neo4j1-conf
#    emptyDir: {}
additionalVolumeMounts: []
#  - mountPath: "/config/neo4j1.conf"
#    name: neo4j1-conf

# ldapPasswordFromSecret defines the secret which holds the password for ldap system account
# Secret key name must be LDAP_PASS
# This secret is accessible by Neo4j at the path defined in ldapPasswordMountPath
ldapPasswordFromSecret: ""

# The above secret gets mounted to the path mentioned here
ldapPasswordMountPath: ""
```

```
# nodeSelector labels
# please ensure the respective labels are present on one of the cluster nodes or else helm charts will
throw an error
nodeSelector: {}
#  label1: "value1"
#  label2: "value2"

# Services for Neo4j
services:
  # A ClusterIP service with the same name as the Helm Release name should be used for Neo4j Driver
connections originating inside the
  # Kubernetes cluster.
  default:
    # Annotations for the K8s Service object
    annotations: { }

  # A LoadBalancer Service for external Neo4j driver applications and Neo4j Browser
  neo4j:
    enabled: true

    # Annotations for the K8s Service object
    annotations: {}

    spec:
      # Type of service.
      type: LoadBalancer

      # in most cloud environments LoadBalancer type will receive an ephemeral public IP address
automatically. If you need to specify a static ip here use:
      # loadBalancerIP: ...

    # ports to include in neo4j service
    ports:
      http:
        enabled: true # Set this to false to remove HTTP from this service (this does not affect
whether http is enabled for the neo4j process)
        # uncomment to publish http on port 80 (neo4j default is 7474)
        #port: 80
        #targetPort: 7474
        #name: http
        #nodePort: <your-nodeport>, enabled only when type set to NodePort
      https:
        enabled: true # Set this to false to remove HTTPS from this service (this does not affect
whether https is enabled for the neo4j process)
        # uncomment to publish http on port 443 (neo4j default is 7473)
        #port: 443
        #targetPort: 7473
        #name: https
        #nodePort: <your-nodeport>, enabled only when type set to NodePort
      bolt:
        enabled: true # Set this to false to remove BOLT from this service (this does not affect
whether https is enabled for the neo4j process)
        # Uncomment to explicitly specify the port to publish Neo4j Bolt (7687 is the default)
        #port: 7687
        #targetPort: 7687
        #name: tcp-bolt
        #nodePort: <your-nodeport>, enabled only when type set to NodePort
      backup:
        enabled: false # Set this to true to expose backup port externally (n.b. this could have
security implications. Backup is not authenticated by default)
        # Uncomment to explicitly specify the port to publish Neo4j Backup (6362 is the default)
        #port: 6362
        #targetPort: 6362
        #name: tcp-backup
        #nodePort: <your-nodeport>, enabled only when type set to NodePort

    selector:
      "helm.neo4j.com/neo4j.loadbalancer": "include"
      # By default the load balancer will match all Neo4j instance types.
      # When Neo4j drivers connect from outside K8s using the load balancer they will not fetch a
routing table.
      # In this case drivers can only use instances included in the load balancer.
      # To only include Neo4j Core instances uncomment the setting below.
      # To only route to Neo4j Read Replicas uncomment the setting and change the value to
"READ_REPLICA"
      # "helm.neo4j.com/clustering": "false"
```

```
    # This flag allows you to open internal neo4j ports necessary in multi zone /region neo4j cluster
scenario
    multiCluster: false

    # The neo4j LoadBalancer service is shared between all servers in the cluster. Because of this,
the `helm.sh/resource-policy: keep`
    # annotation is used to avoid helm ownership conflicts when another release attempts to update the
service.
    # To prevent the service being orphaned when uninstalling a release, a pre-delete helm hook is
provided by the template `delete-loadbalancer-hook.yaml`
    # This is enabled by default, and will create a Job, Service Account, Role and Role Binding that
will run a kubectl image and delete the service
    # If enabled: is set to false, the LoadBalancer will be orphaned and will have to manually deleted
post uninstall and the hook job will not be created
    cleanup:
      enabled: true
      # Pod annotations for the cleanup job
      podAnnotations:
        sidecar.istio.io/inject: "false"
      image:
        registry: docker.io
        repository: bitnami/kubectl
        # Will default to use the Kubernetes server version where the chart is deployed, eg 1.22
        tag: ""
        digest: ""
        imagePullPolicy: IfNotPresent

  # A service for admin/ops tasks including taking backups
  # This service is available even if the deployment is not "ready"
  admin:
    enabled: true
    # Annotations for the admin service
    annotations: { }
    spec:
      type: ClusterIP
    # n.b. there is no ports object for this service. Ports are autogenerated based on the neo4j
configuration

  # A ClusterIP service for admin/ops and Neo4j cluster-internal communications
  # This is no longer a headless service as headless service have been seen to introduce latency
whenever a cluster member restarts
  # This service is available even if the deployment is not "ready"
  internals:
    enabled: false

    # Annotations for the internals service
    annotations: { }
    spec:
      type: ClusterIP
    # n.b. there is no ports object for this service. Ports are autogenerated based on the neo4j
configuration


# Neo4j Configuration (yaml format)
config:
  server.config.strict_validation.enabled: "false"
  #dbms.cluster.minimum_initial_system_primaries_count: "3"
  # The amount of memory to use for mapping the store files.
  # The default page cache memory assumes the machine is dedicated to running
  # Neo4j, and is heuristically set to 50% of RAM minus the Java heap size.
  # server.memory.pagecache.size: "74m"

  #The number of Cypher query execution plans that are cached.
  # server.db.query_cache_size: "10"

  # Java Heap Size: by default the Java heap size is dynamically calculated based
  # on available system resources. Uncomment these lines to set specific initial
  # and maximum heap size.
  # server.memory.heap.initial_size: "317m"
  # server.memory.heap.max_size: "317m"

apoc_config: {}
#  apoc.trigger.enabled: "true"
#  apoc.import.file.enabled: "true"

#apoc_credentials allow you to set configs like apoc.jdbc.<aliasname>.url and apoc.es.<aliasname>.url
via a kubernetes secret mounted on the provided path
```

```
#ensure the secret exists beforehand or else an error will be thrown by the helm chart
#aliasName , secretName and secretMountPath are compulsory fields. Empty fields will result in error
#please ensure you are using the compatible apoc-extended plugin jar while using apoc_credentials
#please ensure the secret is created with the key named as "URL"
  #Ex: kubectl create secret generic jdbcsecret --from
-literal=URL="jdbc:mysql://30.0.0.0:3306/Northwind?user=root&password=password"
apoc_credentials: {}
#   jdbc:
#    aliasName: "jdbc"
#    secretName: "jdbcsecret"
#    secretMountPath: "/secret/jdbcCred"
#
#   elasticsearch:
#     aliasName: "es"
#     secretName: "essecret"
#     secretMountPath: "/secret/esCred"



# securityContext defines privilege and access control settings for a Pod. Making sure that we dont
run Neo4j as root user.
securityContext:
  runAsNonRoot: true
  runAsUser: 7474
  runAsGroup: 7474
  fsGroup: 7474
  fsGroupChangePolicy: "Always"

# securityContext defines privilege and access control settings for a Container. Making sure that we
dont run Neo4j as root user.
containerSecurityContext:
  runAsNonRoot: true
  runAsUser: 7474
  runAsGroup: 7474
  capabilities:
    drop: [ "ALL" ]

# Readiness probes are set to know when a container is ready to be used.
# Because Neo4j uses Java these values are large to distinguish between long Garbage Collection pauses
(which don't require a restart) and an actual failure.
# These values should mark Neo4j as not ready after at most 5 minutes of problems (20 attempts * max
15 seconds between probes)
readinessProbe:
  tcpSocket:
    port: 7687
  failureThreshold: 20
  timeoutSeconds: 10
  periodSeconds: 5

# Liveness probes are set to know when to restart a container.
# Because Neo4j uses Java these values are large to distinguish between long Garbage Collection pauses
(which don't require a restart) and an actual failure.
# These values should trigger a restart after at most 10 minutes of problems (40 attempts * max 15
seconds between probes)
livenessProbe:
  tcpSocket:
    port: 7687
  failureThreshold: 40
  timeoutSeconds: 10
  periodSeconds: 5

# Startup probes are used to know when a container application has started.
# If such a probe is configured, it disables liveness and readiness checks until it succeeds
# When restoring Neo4j from a backup it's important that startup probe gives time for Neo4j to recover
and/or upgrade store files
# When using Neo4j clusters it's important that startup probe give the Neo4j cluster time to form
startupProbe:
  tcpSocket:
    port: 7687
  failureThreshold: 1000
  periodSeconds: 5

# top level setting called ssl to match the "ssl" from "dbms.ssl.policy"
ssl:
  # setting per "connector" matching neo4j config
  bolt:
    privateKey:
```

```
      secretName:  # we set up the template to grab `private.key` from this secret
      subPath:  # we specify the privateKey value name to get from the secret
    publicCertificate:
      secretName:  # we set up the template to grab `public.crt` from this secret
      subPath:  # we specify the publicCertificate value name to get from the secret
    trustedCerts:
      sources: [ ] # a sources array for a projected volume - this allows someone to (relatively)
easily mount multiple public certs from multiple secrets for example.
    revokedCerts:
      sources: [ ]  # a sources array for a projected volume
  https:
    privateKey:
      secretName:
      subPath:
    publicCertificate:
      secretName:
      subPath:
    trustedCerts:
      sources: [ ]
    revokedCerts:
      sources: [ ]
  cluster:
    privateKey:
      secretName:
      subPath:
    publicCertificate:
      secretName:
      subPath:
    trustedCerts:
      sources: [ ]
    revokedCerts:
      sources: [ ]

# Kubernetes cluster domain suffix
clusterDomain: "cluster.local"

# Discovery version, possible values are V1_ONLY, V1_OVER_V2, V2_OVER_V1, V2_ONLY
discoveryVersion: "V1_ONLY"


# Override image settings in Neo4j pod
image:
  imagePullPolicy: IfNotPresent
  # set a customImage if you want to use your own docker image
#  customImage: eu.gcr.io/neo4j-helm/neo4j:v5

  #imagePullSecrets list
#  imagePullSecrets:
#    - "demo"

  #imageCredentials list for which secret of type docker-registry will be created automatically using
the details provided
  # password, name are compulsory fields for an imageCredential , without these fields helm chart will
throw an error
  # registry ,username and email are optional fields
  # imageCredential name should be part of the imagePullSecrets list or else the respective
imageCredential will be ignored and no secret creation will be done
  # In case of a secret already pre-existing you don't need to mention the imageCredential , just add
the pre-existing secretName to the imagePullSecret list
  # and that will be used as an imagePullSecret
#  imageCredentials:
#    - registry: ""
#      username: ""
#      password: ""
#      email: ""
#      name: ""

statefulset:
  metadata:
    #Annotations for Neo4j StatefulSet
    annotations:
#      imageregistry: "https://hub.docker.com/"
#      demo: alpha

# additional environment variables for the Neo4j Container
env: {}
```

```
# Other K8s configuration to apply to the Neo4j pod
podSpec:

  #Annotations for Neo4j pod
  annotations: {}
#   imageregistry: "https://hub.docker.com/"
#   demo: alpha

  nodeAffinity: {}
#    requiredDuringSchedulingIgnoredDuringExecution:
#      nodeSelectorTerms:
#        - matchExpressions:
#            - key: topology.kubernetes.io/zone
#              operator: In
#              values:
#                - antarctica-east1
#                - antarctica-west1
#    preferredDuringSchedulingIgnoredDuringExecution:
#      - weight: 1
#        preference:
#          matchExpressions:
#            - key: another-node-label-key
#              operator: In
#              values:
#                - another-node-label-value

  # Anti Affinity
  # If set to true then an anti-affinity rule is applied to prevent database pods with the same
`neo4j.name` running on a single Kubernetes node.
  # If set to false then no anti-affinity rules are applied
  # If set to an object then that object is used for the Neo4j podAntiAffinity
  podAntiAffinity: true
#    requiredDuringSchedulingIgnoredDuringExecution:
#      - labelSelector:
#          matchLabels:
#            app: "demo"
#            helm.neo4j.com/pod_category: "neo4j-instance"
#        topologyKey: kubernetes.io/hostname

  #Add tolerations to the Neo4j pod
  tolerations: []
#    - key: "key1"
#      operator: "Equal"
#      value: "value1"
#      effect: "NoSchedule"
#    - key: "key2"
#      operator: "Equal"
#      value: "value2"
#      effect: "NoSchedule"

  # topologySpreadConstraints fields for Neo4j Pod
  # https://kubernetes.io/docs/concepts/scheduling-eviction/topology-spread-constraints/
  topologySpreadConstraints: []
#    - maxSkew: 1
#      topologyKey: kubernetes.io/hostname
#      whenUnsatisfiable: DoNotSchedule
#      labelSelector:
#        matchLabels:
#          app: foo
#      matchLabelKeys:
#        - pod-template-hash

  #Priority indicates the importance of a Pod relative to other Pods.
  # More Information : https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-
preemption/
  priorityClassName: ""

  #This indicates that the neo4j instance be included to the loadbalancer. Can be set to exclude to
not add the stateful set to loadbalancer
  loadbalancer: "include"

  # set pod's dns policy. ClusterFirst by default
  # https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/#pod-s-dns-policy
  dnsPolicy: "ClusterFirst"

  # Name of service account to use for the Neo4j Pod (optional)
  # this is useful if you want to use Workload Identity to grant permissions to access cloud resources
```

```
e.g. cloud object storage (AWS S3 etc.)
  # For clusters, please ensure that it has the appropriate roles and role-bindings to be able to
query kubernetes services
  serviceAccountName: ""

  # How long the Neo4j pod is permitted to keep running after it has been signalled by Kubernetes to
stop. Once this timeout elapses the Neo4j process is forcibly terminated.
  # A large value is used because Neo4j takes time to flush in-memory data to disk on shutdown.
  terminationGracePeriodSeconds: 3600

  # initContainers for the Neo4j pod
  initContainers: [ ]

  # additional runtime containers for the Neo4j pod
  containers: [ ]

# print the neo4j user password set during install to the `helm install` log
logInitialPassword: true

# Jvm configuration for Neo4j
jvm:
  # If true any additional arguments are added after the Neo4j default jvm arguments.
  # If false Neo4j default jvm arguments are not used.
  useNeo4jDefaultJvmArguments: true
  # additionalJvmArguments is a list of strings. Each jvm argument should be a separate element:
  additionalJvmArguments: []
#   - "-XX:+HeapDumpOnOutOfMemoryError"
#   - "-XX:HeapDumpPath=/logs/neo4j.hprof"
#   - "-XX:MaxMetaspaceSize=180m"
#   - "-XX:ReservedCodeCacheSize=40m"

logging:
  serverLogsXml: |-
#    <?xml version="1.0" encoding="UTF-8"?>
#    <!-- Example JSON logging configuration -->
#    <Configuration status="ERROR" monitorInterval="30" packages="org.neo4j.logging.log4j">
#        <Appenders>
#            <!-- Default debug.log, please keep -->
#            <RollingRandomAccessFile name="DebugLog"
fileName="${config:server.directories.logs}/debug.log"
#                                     filePattern="$${config:server.directories.logs}/debug.log.%02i">
#                <JsonTemplateLayout
eventTemplateUri="classpath:org/neo4j/logging/StructuredLayoutWithMessage.json"/>
#                <Policies>
#                    <SizeBasedTriggeringPolicy size="20 MB"/>
#                </Policies>
#                <DefaultRolloverStrategy fileIndex="min" max="7"/>
#            </RollingRandomAccessFile>
#
#            <RollingRandomAccessFile name="HttpLog"
fileName="${config:server.directories.logs}/http.log"
#                                     filePattern="$${config:server.directories.logs}/http.log.%02i">
#                <JsonTemplateLayout
eventTemplateUri="classpath:org/neo4j/logging/StructuredLayoutWithMessage.json"/>
#                <Policies>
#                    <SizeBasedTriggeringPolicy size="20 MB"/>
#                </Policies>
#                <DefaultRolloverStrategy fileIndex="min" max="5"/>
#            </RollingRandomAccessFile>
#
#            <RollingRandomAccessFile name="QueryLog"
fileName="${config:server.directories.logs}/query.log"
#                                     filePattern="$${config:server.directories.logs}/query.log.%02i">
#                <JsonTemplateLayout
eventTemplateUri="classpath:org/neo4j/logging/QueryLogJsonLayout.json"/>
#                <Policies>
#                    <SizeBasedTriggeringPolicy size="20 MB"/>
#                </Policies>
#                <DefaultRolloverStrategy fileIndex="min" max="7"/>
#            </RollingRandomAccessFile>
#
#            <RollingRandomAccessFile name="SecurityLog"
fileName="${config:server.directories.logs}/security.log"
#
filePattern="$${config:server.directories.logs}/security.log.%02i">
#                <JsonTemplateLayout
eventTemplateUri="classpath:org/neo4j/logging/StructuredLayoutWithMessage.json"/>
```

```
#               <Policies>
#                   <SizeBasedTriggeringPolicy size="20 MB"/>
#               </Policies>
#               <DefaultRolloverStrategy fileIndex="min" max="7"/>
#           </RollingRandomAccessFile>
#       </Appenders>
#
#       <Loggers>
#           <!-- Log levels. One of DEBUG, INFO, WARN, ERROR or OFF -->
#
#           <!-- The debug log is used as the root logger to catch everything -->
#           <Root level="INFO">
#               <AppenderRef ref="DebugLog"/> <!-- Keep this -->
#           </Root>
#           <!-- The query log, must be named "QueryLogger" -->
#           <Logger name="QueryLogger" level="INFO" additivity="false">
#               <AppenderRef ref="QueryLog"/>
#           </Logger>
#           <!-- The http request log, must be named "HttpLogger" -->
#           <Logger name="HttpLogger" level="INFO" additivity="false">
#               <AppenderRef ref="HttpLog"/>
#           </Logger>
#           <!-- The security log, must be named "SecurityLogger" -->
#           <Logger name="SecurityLogger" level="INFO" additivity="false">
#               <AppenderRef ref="SecurityLog"/>
#           </Logger>
#       </Loggers>
#   </Configuration>
  userLogsXml: |-
#   <?xml version="1.0" encoding="UTF-8"?>
#   <!-- Example JSON logging configuration -->
#   <Configuration status="ERROR" monitorInterval="30" packages="org.neo4j.logging.log4j">
#   <Appenders>
#       <RollingRandomAccessFile name="Neo4jLog"
fileName="${config:server.directories.logs}/neo4j.log"
#                                filePattern="$${config:server.directories.logs}/neo4j.log.%02i">
#           <JsonTemplateLayout
eventTemplateUri="classpath:org/neo4j/logging/StructuredLayoutWithMessage.json"/>
#           <Policies>
#               <SizeBasedTriggeringPolicy size="20 MB"/>
#           </Policies>
#           <DefaultRolloverStrategy fileIndex="min" max="7"/>
#       </RollingRandomAccessFile>
#       <!-- Only used by "neo4j console", will be ignored otherwise -->
#       <Console name="ConsoleAppender" target="SYSTEM_OUT">
#           <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSSZ}{GMT+0} %-5p %m%n"/>
#       </Console>
#   </Appenders>
#   <Loggers>
#       <!-- Log level for the neo4j log. One of DEBUG, INFO, WARN, ERROR or OFF -->
#       <Root level="INFO">
#           <AppenderRef ref="Neo4jLog"/>
#           <AppenderRef ref="ConsoleAppender"/>
#       </Root>
#   </Loggers>
#   </Configuration>

# define your podDisruptionBudget details here
podDisruptionBudget:
  enabled: false
  matchLabels: {}
#    "demo": "neo4j"
  matchExpressions: []
#    - key: "demo"
#      operator: "Equals"
#      value: "neo4j"
  labels: {}
#    "name": "neo4j"
  minAvailable: ""
  maxUnavailable: ""

# Service Monitor for prometheus
# Please ensure prometheus operator or the service monitor CRD is present in your cluster before using
service monitor config
serviceMonitor:
  enabled: false
  labels: {}
```

```
#    "demo": "value"
  jobLabel: ""
  interval: ""
  port: ""
  path: ""
  namespaceSelector: {}
#    any: false
#    matchNames:
#      - default
  targetLabels: []
#    - "demo"
#    - "value"
  selector: {}
#    matchLabels:
#      helm.neo4j.com/service: "admin"


# this section is to be used only when setting up (1 primary + n secondary neo4j instances scenario)
# Disabled by default.
analytics:
  # This flag will enable the internal ports and certain configs necessary to allow 1 primary + n
secondary neo4j instances scenario
  enabled: false
  type:
    # values can be primary or secondary
    # this field denotes the neo4j instance type either primary or secondary
    name: primary
```

3. Pass the *neo4j-values.yaml* file during installation. The `neo4j.name` parameter is mandatory and can be supplied either in `neo4j-values.yaml` or by using the `--set` argument.

```
helm install <release-name> neo4j/neo4j --set "neo4j.name=my-neo4j-db" -f neo4j-values.yaml
```

Tip:

To see the values that have been set for a given release, use `helm get values <release-name>`.

Configuring SSL

The Neo4j SSL Framework can be used with Neo4j Helm charts. You can specify SSL policy objects for `bolt`, `https`, `cluster`, and `backup`. SSL public certificates and private keys to use with a Neo4j Helm deployment must be stored in *Kubernetes Secrets*.

To enable Neo4j SSL policies, configure the `ssl.<policy name>` object in the Neo4j Helm deployment's *values.yaml* file to reference the *Kubernetes Secrets* containing the SSL certificates and keys to use. This example shows how to configure the `bolt` SSL policy:

```
ssl:
 bolt:
   privateKey:
     secretName: bolt-cert
     subPath: private.key
   publicCertificate:
     secretName: bolt-cert
     subPath: public.crt
```

When a private key is specified in the *values.yaml* file, the Neo4j `ssl` policy is enabled automatically. To disable a policy, add `dbms.ssl.policy.{{ $name }}.enabled: "false"` to the `config` object.

> **Note:**
>
> Unencrypted `http` is not disabled automatically when `https` is enabled. If `https` is enabled, add `server.http.enabled: "false"` to the `config` object to disable `http`.

For more information on configuring SSL policies, see SSL configuration.

The following examples show how to deploy a Neo4j cluster with configured SSL policies.

Create a self-signed certificate

If you do not have a self-signed certificate to use, follow the steps to create one:

1. Create a new folder for the self-signed certificate. This example uses the */neo4j-ssl* folder.

```
mkdir neo4j-ssl
cd neo4j-ssl
```

2. Create the `private.key` and `public.crt` for the self-signed certificate by using the `openssl` command and passing all the values in the `subj` argument:

```
openssl req -newkey rsa:2048 -nodes -keyout private.key -x509 -days 365 -out public.crt -subj
"/C=GB/ST=London/L=London/O=Neo4j/OU=IT Department"
```

3. Verify that the *private.key* and *public.crt* files are created:

```
ls -lst
```

Example output

```
-rw-r--r--  1 user  staff  1679  28 Dec 15:00 private.key
-rw-r--r--  1 user  staff  1679  28 Dec 15:00 public.crt
```

Create a `neo4j` namespace and configure it to be used in the current context

```
kubectl create namespace neo4j
kubectl config set-context --current --namespace=neo4j
```

Configure an SSL policy using a `tls` Kubernetes secret

This example shows how to configure an SSL policy for intra-cluster communication using a self-signed certificate stored in a `tls` Kubernetes secret.

1. Create a Kubernetes TSL secret using the *public.crt* and *private.key* files:

> You must have a Kubernetes cluster running and the `kubectl` command installed. For

more information, see Prerequisites.

a. To create a TLS secret, use the `tls` option and a secret name, e.g., `neo4j-tls`:

```
kubectl create secret tls neo4j-tls --cert=/path/to/neo4j-ssl/public.crt --key=/path/to/neo4j-ssl/private.key
```

b. Verify that the secret is created:

```
kubectl get secret
```

Example output

```
NAME                  TYPE                                  DATA   AGE
neo4j-tls             kubernetes.io/tls                     2      4s
```

c. Verify that the secret contains the *public.crt* and *private.key* files:

```
kubectl get secret neo4j-tls -o yaml
```

Example output

```
apiVersion: v1
data:
  tls.crt:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURLakNDQWhJQ0NRRGRYYVg1Y29mczdEQU5CZ2txaGtpRzl3MEJBUXNGQU
RCWE1Rc3dDUVlEVlFRR0V3SkgKUWpFUE1BMEdBMVVFQ0F3R1RHOXVaRzl1TVE4d0RRWURWUVFIREFaTWIyNWtiMjR4RGpBTUJn
TlZCQW9NQlU1bApielJxTVJZd0ZBWURWUVFMREExSlZDQkVaWEJoY25SdFpXNTBNQjRYRFRJeU1USXlPREl4TURjeU5sb1hEVE
l6Ck1USXlPREl4TURjeU5sb3dWekVMTUFrR0ExVUVCaE1DUjBJeER6QU5CZ05WQkFnTUJreHZibVJ2YmpFUE1BMEcKQTFVRUJ3
d0dURzl1Wkc5dU1RNHdEQVlEVlFRS0RBVk9aVzgwYWpFV01CUUdBMVVFQ3d3TlNWUWdSR1Z3WVhKMApiV1Z1ZERDQ0FTSXdEUV
lKS29aSWh2Y05BUUVCQlFBRGdnRVBBRENDQVFvQ2dnRUJBTDBRc0c2Ukwrd3hxZSt3CjJGSWljZldVaUFtdmNqeVdlS0lKaThu
T2tBSGIvSTYzUUU2L3ZpR3RNeEI3S28xdUJLNlVPZXBaeU91UzE2bUMKaitpMDAwbmFnWkR3RGNyRXd3UUE1cTBGMC90VXB5UH
BaL1p3clhEaGFDOXhzVnFnVms0TXl5aUtTNzRIOUc2UgprUUV4dHBaNFArcTlaRHVFVk1KVGVaL2pQNGZoTkg2MUpSTVdORTJ3
NjNUWkx2ZGMyUitXL2U5N3h2TGQ5Y0FnCjlqTm9FMHo5UHRmczB2L2lyUGhuUHpzWHQ5bzE0MWlnOVFZNjNtMzBxQ0NaYnpMRl
R6WFgvdTUvTSsycFB3WXoKcUNOTUZYYW1ITlAxdlRPWFlRTG1iYW1JdVplYnVPNEVlUHZ6WUVXSmEyUi9oTmhtUDNvM2tRVFAz
dmF1UEFjZQpSQlJZS09NQ0F3RUFBVEFOQmdrcWhraUc5dzBCQVFzRkFBT0NBUUVBWVh6SkIzNU55cExDUEdvQXVJdGt5dHBFCk
1ZeSs5YnlrV3BEVnhRaXZLUHQ3d1NUaWZjNU1QdW5NUy9xYmxnaWprWm5IaWVLeEdoU3lQL283QndtMzJDSnAKQUFsSjJ3RjhI
VXVSSGpYcUU5dkNQeFdtVlVJS2ExOWN5V0tUYWhySWU1eWZkQWNkbUJmRzJNWnY0dEdFeWxsUgo0Vk81STdRNjVWZDlGQnB0U3
JjS3R1WUtBUzg2RTBHZmlmMWxCakdUZTFZbkhvK1RZTVpoVEUvN3RlNHZ1M251CjA4Y1BmbS9RYThSNFBXZDZNbXVDaTJYcDdu
WVlEMmp3WklCSENtMUU3U1RrdS9JRk5kOWFWRW91VG5KR1pCWFcKeWVzWG9OMXhOb3kvMXZFdElhV2xXZW1GcGo4clJ6VGJQek
Q1TEpiNDBSRFVOTXN3NytLUXczV3BBMjVKUHc9PQotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
  tls.key:
LS0tLS1CRUdJTiBQUklWQVRFIEtFWS0tLS0tCk1JSUV2QUlCQURBTkJna3Foa2lHOXcwQkFRRUZBQVNDQktZd2dnU2lBZ0VBQW
9JQkFRQzlFTEJ1a1Mvc01hbnYKc05oU0luSDFsSWdKcjNJOGxuaWlDWXZKenBBQjIveU90MEJPdjc0aHJUTVFleXFOYmdTdWxE
bnFXY2pya3RlcApnby9vdE5OSjJvR1E4QTNLeE1NRUFPYXRCZFA3VktjajZXZjJjSzF3NFdndmNiRmFvRlpPRE1zb2lrdStCL1
J1CmtaRUJNYmFXZUQvcXZXUTdoRlRDVTNtZjR6K0g0VFIrdFNVVEZqUk5zT3QwMlM3M1hOa2ZsdjN2ZThieTNmWEEKSVBZemFC
Tk0vVDdYN05MLzRxejRaejg3RjdmYU5lTllvUFVHT3Q1dDlLZ2dtVzh5eFU4MTEvN3VmelB0cVQ4RwpNNmdqVEJWMnBoelQ5Yj
B6bDJFQzVtMnBpTG1YbTdqdUJIajc4MkJGaVd0a2Y0VFlaajk2TjVFRXo5NzJyandICkhrUVVXQ2pqQWdNQkFBRUNnZ0VBTmF6
OVNnYXlJazVmUG90b2Zya0V2WUh6dFR3NEpIZGJ2RFVWbUsrcU5yenIKME9DNXd5R3dxd0x2RW1qRlJlM01Lbnd1alJmOGNOVD
VvVWhON3ZVWFgwcEhxb3hjZmdxcWl3SnVld1RDa0FJUwppYUdFUUhUdzZMRTEwUEpvTmFCN29DRUZ0SGErMWk2UCtLd2ZETVcr
WHEyNUI3M0pMUlIrczhUYkxNZHBpL3VvCjRmTFNJV0xDV09MZThUTlU0ck5vVDQ0enY0eUhUOXAyV3liSUNrL3F5bVV3bTlhRH
FnYzRJRzk4YXVVNG5JYVQKenk2T3NBODdONW9FME0rcHlUdEFJcmxRZFBXUzBBZ28xZUJCcWplL1I3MTI4TmdHVzhOTzVMWDBt
Nit2YzhyVgpaTHh0N1d0NThucXR2WlI3QTF5SU9lbWtocHl2Q3hrNVRxSmZQRlJxRVFLQmdRRGlOL3NBZncrZ1dvTFpLbTNyCm
50WVkyRW9TOTBkQ0wwd293SGFGa0lsL2hIWXduQi9qaWlnMU5ZbEhDNzNPVDdDc2UycS9tS0xhMzZBVHlpTHcKZjN1T0J3NmNF
Z2RJZlU5aDBtZjJCNFZXdEVEeDJMSU94METZU2VrYldTTVZQZ2w2SkhNb3hLdjNMbEx5R1RiMApZQmtKVmpRdkVLS1dpa1FLMU
dPYnZtdzFWUUtCZ1FEVjlJLzc5WFJuN1EzZ3M4Z2JqZWhDejRqNHdtNWdpNFM2CkVsVzVJWkFidDh3QWZPdVIzUm4wQW41NFl0
ZW1HUk1seDF0L1ZUM1IzK0J5bmVTZEgzbUJ6eVJEQysvRGhBTlYKNVZPckk5SFhnVTRMSElVMmNwVVZxYVo3N1J1b2JnTmlDen
BmOVZPVkNadzdmQzRPYkFqcTMzQ3RtT2taR0hRbAo2dkJtNm1ubFZ3S0JnQ2FnOW95TUplZjA3TGtXcExTQ1ovN1FHRDRLMmJF
MGtHVzVEOFFZL1ZHNEZkS1JKbVRkCmQ2WTJZUjJ2cEpheFJ2TDlGQ3BwYncyKzkvL0pHWlJGd0p4dEdoS09oWTNjVUF6ZE9BRn
NJVm0vNkFNa1JLdC8KWFNEU0ppc1VXb2hMRXFVM3lpNWcveGh6WVppVHM2MmhKMFZQNGhOVFhPQWw5aDUvVEE4UlFqc05Bb0dB
Tm84Twp5R2xuTGJrOWVMZGZwK2NmK3ltQS9DNVloellNdW9aQ1pkc3hMR0JLSFRXOXZJeHRPZFFJL0JuNGM5cWhEMWt1CjgrR0
F5aXdVeUNXTFRxWGdEa0lNTlN5dUQyVnlsRXpPY1MzSkxQTkVPNEVpVnlnUTdGMCtud3R2cWh1anNUUzcKeGd5Qks5Z3ZodHU3
d3VHNXhHc0dDTDZkY2xEU0RYbERwSHJTVmpFQ2dZQWx0STNjMzJxaG5KU2xHSGhjdW1wRwpReGpvYnJBUUxUa3dyOWk2TkNuS0
EyNVR1SVFXa1NiN2JUWWtuUi80WDhOT2w2U2EvYm9QK2dncWNJM0haSk05CkxJRnpPUTFWT1luQ2ZYZVd0SmlHQklwUExadFdo
bnA3NGVhdmJKYW9udlhVVGNZcm5qcytIWGhpaFhjOUhENWsKeEJEaWJKYUlEbXg2T1FpVWI2RndJZz09Ci0tLS0tRU5EIFBSSV
```

```
ZBVEUgS0VZLS0tLS0K
kind: Secret
metadata:
  creationTimestamp: "2023-01-04T13:53:14Z"
  managedFields:
  - apiVersion: v1
    fieldsType: FieldsV1
    fieldsV1:
      f:data:
        .: {}
        f:tls.crt: {}
        f:tls.key: {}
      f:type: {}
    manager: kubectl
    operation: Update
    time: "2023-01-04T13:53:14Z"
  name: neo4j-tls
  namespace: neo4j
  resourceVersion: "212009"
  uid: b1be45dd-4cbe-41c9-a6e5-c814c5e39c25
type: kubernetes.io/tls
```

2. Configure `ssl` object in the *ssl-values.yaml* file using the created secret:

```
ssl:
# setting per "connector" matching neo4j config
  bolt:
    privateKey:
      secretName: neo4j-tls
      subPath: tls.key
    publicCertificate:
      secretName: neo4j-tls
      subPath: tls.crt
  https:
    privateKey:
      secretName: neo4j-tls
      subPath: tls.key
    publicCertificate:
      secretName: neo4j-tls
      subPath: tls.crt
    trustedCerts:
      sources:
      - secret:
          name: neo4j-tls
          items:
          - key: tls.crt
            path: public.crt
  cluster:
    privateKey:
      secretName: neo4j-tls
      subPath: tls.key
    publicCertificate:
      secretName: neo4j-tls
      subPath: tls.crt
    trustedCerts:
      sources:
      - secret:
          name: neo4j-tls
          items:
          - key: tls.crt
            path: public.crt
    revokedCerts:
      sources: [ ]
```

Now you are ready to deploy the Neo4j cluster using the configured *ssl-values.yaml* file and the Neo4j Helm charts.

Configure an SSL policy using a `generic` Kubernetes secret

This example shows how to configure an SSL policy for intra-cluster communication using a self-signed certificate stored in a `generic` Kubernetes secret.

1. Create a Kubernetes `generic` secret using the *public.crt* and *private.key* files:

   > You must have a Kubernetes cluster running and the `kubectl` command installed. For more information, see Prerequisites.

   a. Get the Base64-encoded value of your *public.crt* and *private.key*:

   ```
   cat public.crt| base64
   ```

   ```
   cat private.key| base64
   ```

   b. Using the Base64-encoded values of your *public.crt* and *private.key*, create a *secret.yaml* file:


```
apiVersion: v1
data:
  public.crt:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURLakNDQWhJQ0NRRGRYYVg1Y29mczdEQU5CZ2txaGtpRzl3MEJBUXNGQU
RCWE1Rc3dDUVlEVlFRR0V3SkgKUWpFUE1BMEdBMVVFQ0F3R1RHOXVaRzl1TVE4d0RRWURWUVFIREFaTWIyNWtiMjR4RGpBTUJn
TlZCQW9NQlU1bApielJxTVJZd0ZBWURWUVFMREExSlZDQkVaWEJoY25SdFpXNTBNQjRYRFRJeU1USXlPREl4TURjeU5sb1hEVE
l6Ck1USXlPREl4TURjeU5sb3dWekVMTUFrR0ExVUVCaE1DUjBJeER6QU5CZ05WQkFnTUJreHZibVJ2YmpFUE1BMEcKQTFVRUJ3
d0dURzl1Wkc5dU1RNHdEQVlEVlFRS0RBVk9aVzgwYWpFV01CUUdBMVVFQ3d3TlNWUWdSR1Z3WVhKMApiV1Z1ZERDQ0FTSXdEUV
lKS29aSWh2Y05BUUVCQlFBRGdnRVBBRENDQVFvQ2dnRUJBTDBRc0c2Ukwrd3hxZSt3CjJGSWljZldVaUFtdmNqeVdlS0lKaThu
T2tBSGIvSTYzUUU2L3ZpR3RNeEI3S28xdUJLNlVPZXBaeU91UzE2bUMKaitpMDAwbmFnWkR3RGNyRXd3UUE1cTBGMC90VXB5UH
BaL1p3clhEaGFDOXhzVnFnVms0TXl5aUtTNzRIOUc2UgprUUV4dHBaNFArTlaRHVFVk1KVGVaL2pQNGZoTkg2MUpSTVdORTJ3
NjNUWkx2ZGMyUitXL2U5N3h2TGQ5Y0FnCjlqTm9FMHo5UHRmczB2L2lyUGhuUHpzWHQ5bzE0MWlnOVFZNjNtMzBxQ0NaYnpMRl
R6WFgvdTUvTSsycFB3WXoKcUNOTUZYYW1ITlAxdlRPWFlRTG1iYW1JdVplYnVPNEVlUHZ6WUVXSmEyUi9oTmhtUDNvM2tRVFAz
dmF1UEFjZQpSQlJZS09NQ0F3RUFBVEFOQmdrcWhraUc5dzBCQVFzRkFBT0NBUUVBWVh6SkIzNU55cExDUEdvQXVJdGt5dHBFCk
1ZeSs5YnlrV3BEVnhRaXZLUHQ3d1NUaWZjNU1QdW5NUy9xYmxnaWprWm5IaWVLeEdoU3lQL283QndtMzJDSnAKQUFsSjJ3RjhI
VXVSSGpYcUU5dkNQeFdtVlVJS2ExOWN5V0tUYWhySWU1eWZkQWNkbUJmRzJNWnY0dEdFeWxsUgo0Vk81STdRNjVWZDlGQnB0U3
JjS3R1WUtBUzg2RTBHZmlmMWxCakdUZTFZbkhvK1RZTVpoVEUvN3RlNHZ1M251CjA4Y1BmbS9RYThSNFBXZDZNbXVDaTJYcDdu
WVlEMmp3WklCSENtMUU3U1RrdS9JRk5kOWFWRW91VG5KR1pCWFcKeWVzWG9OMXhOb3kvMXZFdElhV2xXZW1GcGo4clJ6VGJQek
Q1TEpiNDBSRFVOTXN3NytLUXczV3BBMjVKUHc9PQotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
  private.key:
LS0tLS1CRUdJTiBQUklWQVRFIEtFWS0tLS0tCk1JSUV2QUlCQURBTkJna3Foa2lHOXcwQkFRRUZBQVNDQktZd2dnU2lBZ0VBQW
9JQkFRQzlFTEJ1a1Mvc01hbnYKc05oU0luSDFsSWdKcjNJOGxuaWlDWXZKenBBQjIveU90MEJPdjc0aHJUTVFleXFOYmdTdWxE
bnFXY2pya3RlcApnby9vdE5OSjJvR1E4QTNLeE1NRUFPYXRCZFA3VktjajZXZjJjSzF3NFdndmNiRmFvRlpPRE1zb2lrdStCL1
J1CmtaRUJNYmFXZUQvcXZXUTdoRlRDVTNtZjR6K0g0VFIrdFNVVEZqUk5zT3QwMlM3M1hOa2ZsdjN2ZThieTNmWEEKSVBZemFC
Tk0vVDdYN05MLzRxejRaejg3RjdmYU5lTllvUFVHT3Q1dDlLZ2dtVzh5eFU4MTEvN3VmelB0cVQ4RwpNNmdqVEJWMnBoelQ5Yj
B6bDJFQzVtMnBpTG1YbTdqdUJIajc4MkJGaVd0a2Y0VFlaajk2TjVFRXo5NzJyandICkhrUVVXQ2pqQWdNQkFBRUNnZ0VBTmF6
OVNnYXlJazVmUG90b2Zya0V2WUh6dFR3NEpIZGJ2RFVWbUsrcU5yenIKME9DNXd5R3dxd0x2RW1qRlJlM01Lbnd1alJmOGNOVD
VvVWhON3ZVWFgwcEhxb3hjZmdxcWl3SnVld1RDa0FJUwppYUdFUUhUdzZMRTEwUEpvTmFCN29DRUZ0SGErMWk2UCtLd2ZETVcr
WHEyNUI3M0pMUlIrczhUYkxNZHBpL3VvCjRmTFNJV0xDV09MZThUTlU0ck5vVDQ0enY0eUhUOXAyV3liSUNrL3F5bVV3bTlhRH
FnYzRJRzk4YXVVNG5JYVQKenk2T3NBODdONW9FME0rcHlUdEFJcmxRZFBXUzBBZ28xZUJCcWplL1I3MTI4TmdHVzhOTzVMWDBt
Nit2YzhyVgpaTHh0N1d0NThucXR2WlI3QTF5SU9lbWtocHl2Q3hrNVRxSmZQRlJxRVFLQmdRRGlOL3NBZncrZ1dvTFpLbTNyCm
50WVkyRW9TOTBkQ0wwd293SGFGa0lsL2hIWXduQi9qaWlnMU5ZbEhDNzNPVDdDc2UycS9tS0xhMzZBVHlpTHcKZjN1T0J3NmNF
Z2RJZlU5aDBtZjJCNFZXdEVEeDJMSU94MEtZU2VrYldTTVZQZ2w2SkhNb3hLdjNMbEx5R1RiMApZQmtKVmpRdkVLS1dpa1FLMU
dPYnZtdzFWUUtCZ1FEVjlJLzc5WFJuN1EzZ3M4Z2JqZWhDejRqNHdtNWdpNFM2CkVsVzVJWkFidDh3QWZPdVIzUm4wQW41NFl0
ZW1HUk1seDF0L1ZUM1IzK0J5bmVTZEgzbUJ6eVJEQysvRGhBTlYKNVZPckk5SFhnVTRMSElVMmNwVVZxYVo3N1J1b2JnTmlDen
BmOVZPVkNadzdmQzRPYkFqcTMzQ3RtT2taR0hRbAo2dkJtNm1ubFZ3S0JnQ2FnOW95TUplZjA3TGtXcExTQ1ovN1FHRDRLMmJF
MGtHVzVEOFFZL1ZHNEZkS1JKbVRkCmQ2WTJZUjJ2cEpheFJ2TDlGQ3BwYncyKzkvL0pHWlJGd0p4dEdoS09oWTNjVUF6ZE9BRn
NJVm0vNkFNa1JLdC8KWFNEU0ppc1VXb2hMRXFVM3lpNWcveGh6WVppVHM2MmhKMFZQNGhOVFhPQWw5aDUvVEE4UlFqc05Bb0dB
Tm84Twp5R2xuTGJrOWVMZGZwK2NmK3ltQS9DNVloellNdW9aQ1pkc3hMR0JLSFRXOXZJeHRPZFFJL0JuNGM5cWhEMWt1CjgrR0
F5aXdVeUNXTFRxWGdEa0lNTlN5dUQyVnlsRXpPY1MzSkxQTkVPNEVpVnlnUTdGMCtud3R2cWh1anNUUzcKeGd5Qks5Z3ZodHU3
d3VHNXhHc0dDTDZkY2xEU0RYbERwSHJTVmpFQ2dZQWx0STNjMzJxaG5KU2xHSGhjdW1wRwpReGpvYnJBUUxUa3dyOWk2TkNuS0
EyNVR1SVFXa1NiN2JUWWtuUi80WDhOT2w2U2EvYm9QK2dncWNJM0haSk05CkxJRnpPUTFWT1luQ2ZYZVd0SmlHQklwUExadFdo
bnA3NGVhdmJKYW9udlhVVGNZcm5qcytIWGhpaFhjOUhENWsKeEJEaWJKYUlEbXg2T1FpVWI2RndJZz09Ci0tLS0tRU5EIFBSSV
ZBVEUgS0VZLS0tLS0K
kind: Secret
metadata:
  name: neo4j-tls
  namespace: neo4j
type: Opaque
```

c. Create the generic secret using the `kubectl create` command and the *secret.yaml* file:

```
kubectl create -f /path/to/secret.yaml
```

Example output

```
secret/neo4j-tls created
```

d. Verify that the secret is created:

```
kubectl get secret
```

Example output

```
NAME        TYPE     DATA   AGE
neo4j-tls   Opaque   2      85s
```

2. Configure the `ssl` object in the *ssl-values.yaml* file using the created secret:

```
ssl:
# setting per "connector" matching neo4j config
  bolt:
    privateKey:
      secretName: neo4j-tls
      subPath: private.key
    publicCertificate:
      secretName: neo4j-tls
      subPath: public.crt
  https:
    privateKey:
      secretName: neo4j-tls
      subPath: private.key
    publicCertificate:
      secretName: neo4j-tls
      subPath: public.crt
    trustedCerts:
      sources:
      - secret:
          name: neo4j-tls
          items:
          - key: public.crt
            path: public.crt
  cluster:
    privateKey:
      secretName: neo4j-tls
      subPath: private.key
    publicCertificate:
      secretName: neo4j-tls
      subPath: public.crt
    trustedCerts:
      sources:
      - secret:
          name: neo4j-tls
          items:
          - key: public.crt
            path: public.crt
    revokedCerts:
      sources: [ ]
```

Now you are ready to deploy the Neo4j cluster using the *ssl-values.yaml* file and the Neo4j Helm charts.

Deploy a Neo4j cluster with SSL certificates

Deploy a Neo4j cluster using the Neo4j Helm chart and the *ssl-values.yaml* file.

1. Install server-1:

```
helm install server-1 neo4j/neo4j --namespace neo4j --set neo4j.acceptLicenseAgreement=yes --set
neo4j.password=my-password --set neo4j.name="my-cluster" --set neo4j.minimumClusterSize=3 --set
neo4j.edition="enterprise" --set volumes.data.mode=defaultStorageClass -f ~/Documents/neo4j-ssl/ssl-
values.yaml
```

2. Repeat the command from the previous step for `server-2` and `server-3`.
3. Verify that the Neo4j cluster is running:

```
kubectl get pods
```

Example output

```
NAME                        READY   STATUS    RESTARTS   AGE
server-1-0                    1/1     Running   0          2m
server-2-0                    1/1     Running   0          2m
server-3-0                    1/1     Running   0          2m
```

4. Connect to one of the servers and verify that the */certificates/cluster* directory contains the certificates:

```
kubectl exec -it server-1-0 -- bash
```

```
neo4j@server-1-0:~$ cd certificates/
neo4j@server-1-0:~/certificates$ ls -lst
```

Example output

```
total 12
4 drwxr-xr-x 2 root root 4096 Jan  4 13:55 bolt
4 drwxr-xr-x 3 root root 4096 Jan  4 13:55 cluster
4 drwxr-xr-x 3 root root 4096 Jan  4 13:55 https
```

```
neo4j@server-1-0:~/certificates$ cd cluster/
neo4j@server-1-0:~/certificates/cluster$ ls -lst
```

Example output

```
total 8
0 drwxrwsrwt 3 root neo4j  100 Jan  4 13:56 trusted
4 -rw-r--r-- 1 root neo4j 1704 Jan  4 13:56 private.key
4 -rw-r--r-- 1 root neo4j 1159 Jan  4 13:56 public.crt
```

```
neo4j@server-1-0:~/certificates/cluster$ cd trusted/
neo4j@server-1-0:~/certificates/cluster/trusted$ ls -lst
```

Example output

```
total 0
0 lrwxrwxrwx 1 root neo4j 17 Jan  4 13:56 public.crt -> ..data/public.crt
```

5. Exit the pod:

```
exit
```

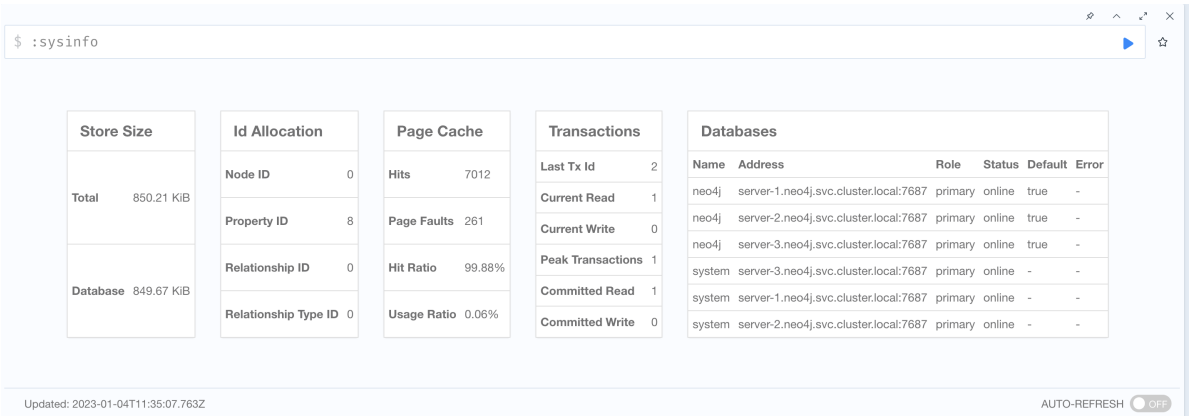
6. Check that the LoadBalancer service is available using the `neo4j.name` used for the installation:

```
export NEO4J_NAME=my-cluster
kubectl get service ${NEO4J_NAME}-lb-neo4j
```

Example output

```
NAME                  TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)
AGE
my-cluster-lb-neo4j   LoadBalancer   10.0.134.210   20.237.50.207
7474:31168/TCP,7473:31045/TCP,7687:32708/TCP   3m30s
```

7. Connect to the Neo4j cluster using one of the following options:
   - Neo4j Browser:
     a. Open a web browser and type *https://lb-EXTERNAL_IP:7473* (in this example, *https://20.237.50.207:7473/browser/*). You should see the Neo4j browser.
     b. Authenticate using the user `neo4j` and the password you set when deploying the cores, in this example, `my-password`.
     c. Verify that the cluster is online by running `:sysinfo` or `SHOW SERVERS`:



     d. Run `SHOW SETTINGS YIELD name, value WHERE name CONTAINS 'ssl'` to verify that the configuration is deployed as expected.
   - Cypher Shell:
     a. Open a terminal and connect to one of the cluster pods:

```
kubectl exec -it server-1-0 -- bash
```

     b. Navigate to the *bin* directory and connect to server-1 using `cypher-shell`:

```
neo4j@server-1-0:~$ cd bin
neo4j@server-1-0:~/bin$ ./cypher-shell -u neo4j -p my-password -a neo4j+ssc://server-
1.neo4j.svc.cluster.local:7687
```

Example output

```
Connected to Neo4j using Bolt protocol version 5.3 at neo4j+ssc://server-
1.neo4j.svc.cluster.local:7687 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
neo4j@neo4j>
```

c. Verify that the cluster is online by running `SHOW SERVERS`:

```
neo4j@server-1-0:~/bin$ SHOW SERVERS;
```

Example output

```
+----------------------------------------------------------------------------------------------------
-------------------------------------+
| name                                   | address                                 | state
| health      | hosting             |
+----------------------------------------------------------------------------------------------------
-------------------------------------+
| "1c5946b1-0eb5-43b9-a549-5601087c57f2" | "server-3.neo4j.svc.cluster.local:7687" | "Enabled"
| "Available" | ["neo4j", "system"]  |
| "ba63cd32-3e7d-4042-9935-c8eba925a98f" | "server-1.neo4j.svc.cluster.local:7687"  |
"Enabled" | "Available" | ["neo4j", "system"] |
| "cbad7ed6-0c13-4ba7-b6a1-f20c5552dfcd" | "server-2.neo4j.svc.cluster.local:7687" | "Enabled"
| "Available" | ["neo4j", "system"]  |
+----------------------------------------------------------------------------------------------------
-------------------------------------+
```

d. Run `SHOW SETTINGS YIELD name, value WHERE name CONTAINS 'ssl'` to verify that the configuration is deployed as expected.

Example output

```
+----------------------------------------------------------------------------------------------------
-------+
| name                                          | value
|
+----------------------------------------------------------------------------------------------------
-------+
| "dbms.netty.ssl.provider"                     | "JDK"
|
| "dbms.ssl.policy.bolt.base_directory"         | "/var/lib/neo4j/certificates/bolt"
|
| "dbms.ssl.policy.bolt.ciphers"                | "No Value"
|
| "dbms.ssl.policy.bolt.client_auth"            | "NONE"
|
| "dbms.ssl.policy.bolt.enabled"                | "true"
|
| "dbms.ssl.policy.bolt.private_key"            |
"/var/lib/neo4j/certificates/bolt/private.key"    |
| "dbms.ssl.policy.bolt.private_key_password"   | "No Value"
|
| "dbms.ssl.policy.bolt.public_certificate"     |
"/var/lib/neo4j/certificates/bolt/public.crt"     |
| "dbms.ssl.policy.bolt.revoked_dir"            | "/var/lib/neo4j/certificates/bolt/revoked"
|
| "dbms.ssl.policy.bolt.tls_versions"           | "TLSv1.2"
|
| "dbms.ssl.policy.bolt.trust_all"              | "false"
|
| "dbms.ssl.policy.bolt.trusted_dir"            | "/var/lib/neo4j/certificates/bolt/trusted"
|
| "dbms.ssl.policy.bolt.verify_hostname"        | "false"
|
| "dbms.ssl.policy.cluster.base_directory"      | "/var/lib/neo4j/certificates/cluster"
|
```

```
| "dbms.ssl.policy.cluster.ciphers"              | "No Value"
|
| "dbms.ssl.policy.cluster.client_auth"          | "REQUIRE"
|
| "dbms.ssl.policy.cluster.enabled"              | "true"
|
| "dbms.ssl.policy.cluster.private_key"          |
"/var/lib/neo4j/certificates/cluster/private.key" |
| "dbms.ssl.policy.cluster.private_key_password" | "No Value"
|
| "dbms.ssl.policy.cluster.public_certificate"   |
"/var/lib/neo4j/certificates/cluster/public.crt"  |
| "dbms.ssl.policy.cluster.revoked_dir"          |
"/var/lib/neo4j/certificates/cluster/revoked"     |
| "dbms.ssl.policy.cluster.tls_versions"         | "TLSv1.2"
|
| "dbms.ssl.policy.cluster.trust_all"            | "false"
|
| "dbms.ssl.policy.cluster.trusted_dir"          |
"/var/lib/neo4j/certificates/cluster/trusted"     |
| "dbms.ssl.policy.cluster.verify_hostname"      | "false"
|
| "dbms.ssl.policy.https.base_directory"         | "/var/lib/neo4j/certificates/https"
|
| "dbms.ssl.policy.https.ciphers"                | "No Value"
|
| "dbms.ssl.policy.https.client_auth"            | "NONE"
|
| "dbms.ssl.policy.https.enabled"                | "true"
|
| "dbms.ssl.policy.https.private_key"            |
"/var/lib/neo4j/certificates/https/private.key"   |
| "dbms.ssl.policy.https.private_key_password"   | "No Value"
|
| "dbms.ssl.policy.https.public_certificate"     |
"/var/lib/neo4j/certificates/https/public.crt"    |
| "dbms.ssl.policy.https.revoked_dir"            | "/var/lib/neo4j/certificates/https/revoked"
|
| "dbms.ssl.policy.https.tls_versions"           | "TLSv1.2"
|
| "dbms.ssl.policy.https.trust_all"              | "false"
|
| "dbms.ssl.policy.https.trusted_dir"            | "/var/lib/neo4j/certificates/https/trusted"
|
| "dbms.ssl.policy.https.verify_hostname"        | "false"
|
+-----------------------------------------------------------------------------------------------
-------+

37 rows
ready to start consuming query after 212 ms, results consumed after another 11 ms
```

Authentication and authorization

Configure LDAP password through secret

Enterprise Edition

To configure the Neo4j Helm deployment to use the LDAP system password through secret, you need to create a Kubernetes secret with the LDAP password and then add the secret name and the mount path to the *values.yaml* file.

1. Create a secret with the LDAP password by running the following command. The secret must have a key called `LDAP_PASS`.

```
kubectl create secret generic <secret-name> --from-literal=LDAP_PASS=<ldap-password>
```

2. Add the secret name to the *values.yaml* file.

```
# ldapPasswordFromSecret defines the secret which holds the password for ldap system account
# Secret key name must be LDAP_PASS
# This secret is accessible by Neo4j at the path defined in ldapPasswordMountPath
ldapPasswordFromSecret: ""

# The above secret gets mounted to the path mentioned here
ldapPasswordMountPath: ""
```

   — The secret name as it appears in the Kubernetes cluster.

   — The path where the secret will be mounted as a volume in the Neo4j container.

Configure SSO

Neo4j supports SSO authentication and authorization through identity providers implementing the OpenID Connect (OIDC) standard.

To configure the Neo4j helm deployment to use SSO authentication, first, you need to configure your identity provider for authentication and authorization using ID tokens. And then, you configure the Neo4j helm deployment to use that identity provider for authentication by adding all the SSO configurations to the *values.yaml* file.

For more information on how to configure your identity provider and what settings you should define, see Neo4j Single Sign-On (SSO) configuration.

An example of configuring Neo4j to use Azure SSO for authentication

```
config:
  dbms.security.oidc.azure.audience: "00f3a7d3-d855-4849-9e3c-57d7b6e12794"
  dbms.security.oidc.azure.params: "client_id=00f3a7d3-d855-4849-9e3c-
57d7b6e12794;response_type=code;scope=openid profile email"
  dbms.security.oidc.azure.well_known_discovery_uri: "https://portal.azure.comda501982-4ca7-420c-8926-
1e65b5bf565f/v2.0/.well-known/openid-configuration"
  dbms.security.authorization_providers: "oidc-azure,native"
  dbms.security.authentication_providers: "oidc-azure,native"
  dbms.security.oidc.azure.display_name: "Azure SSO on K8s"
  dbms.security.oidc.azure.auth_flow: "pkce"
  server_type_principal=id_token;token_type_authentication=id_token"
  dbms.security.oidc.azure.config: "principal=unique_name;code_challenge_method=S256;
  dbms.security.oidc.azure.claims.username: "sub"
  dbms.security.oidc.azure.claims.groups: "groups"
  dbms.security.oidc.azure.authorization.group_to_role_mapping: "group1=editor;group2=editor,publisher"
```

Important:

`sub` is the only claim guaranteed to be unique and stable. Other claims, such as `email` or `preferred_username`, may change over time and should **not** be used for authentication. Neo4j may assign permissions to a user based on this username value in a hybrid authorization configuration. Thus, changing the username claim from `sub` is not recommended. For details, see Microsoft documentation as well as the Openld spec.

Configure a service account

In some deployment situations, it may be desirable to assign a Kubernetes Service Account to the Neo4j

pod. For example, if processes in the pod want to connect to services that require Service Account authorization. To configure the Neo4j pod to use a Kubernetes service account, set `podSpec.serviceAccountName` to the name of the service account to use.

For example:

```
# neo4j-values.yaml
neo4j:
  password: "my-password"

podSpec:
  serviceAccountName: "neo4j-service-account"
```

Note:

The service account must already exist. In the case of clusters, the Neo4j Helm chart creates and configures the service account.

Plugins

There are three recommended methods for adding Neo4j plugins to Neo4j Helm chart deployments. You can use:

- an automatic plugin download
- a custom container image.
- a `plugins` volume.

Add plugins using an automatic plugin download

You can configure the Neo4j deployment to automatically download and install plugins. If licenses are required for the plugins, you must provide the licenses in a secret.

Install GDS Community Edition (CE)

GDS Community Edition does not require a license. To add the GDS CE, configure the Neo4j *values.yaml* and set the `env` to download the plugins:

```
neo4j:
  name: licenses
  acceptLicenseAgreement: "yes"
  edition: enterprise
volumes:
  data:
    mode: defaultStorageClass
env:
  NEO4J_PLUGINS: '["graph-data-science"]'
config:
  dbms.security.procedures.unrestricted: "gds.*"
```

Install GDS Enterprise Edition (EE) and Bloom plugins

To install GDS EE and Bloom, you must provide a license for each plugin. You provide the licenses in a Kubernetes secret.

1. Create a secret containing the licenses:

```
kubectl create secret  generic --from-file=gds.license,bloom.license gds-bloom-license
```

2. Configure the Neo4j *values.yaml* file using the secret as the */licenses* volume mount, and set the `env` to download the plugins:

```
neo4j:
  name: licenses
  acceptLicenseAgreement: "yes"
  edition: enterprise
volumes:
  data:
    mode: defaultStorageClass
  licenses:
    disableSubPathExpr: true
    mode: volume
    volume:
      secret:
        secretName: gds-bloom-license
        items:
          - key: gds.license
            path: gds.license
          - key: bloom.license
            path: bloom.license
env:
  NEO4J_PLUGINS: '["graph-data-science", "bloom"]'
config:
  gds.enterprise.license_file: "/licenses/gds.license"
  dbms.security.procedures.unrestricted: "gds.*,apoc.*,bloom.*"
  server.unmanaged_extension_classes: "com.neo4j.bloom.server=/bloom,semantics.extension=/rdf"
  dbms.security.http_auth_allowlist: "/,/browser.*,/bloom.*"
  dbms.bloom.license_file: "/licenses/bloom.license"
```

Add plugins using a custom container image

The best method for adding plugins to Neo4j running in Kubernetes is to create a new Docker container image that contains both Neo4j and the Neo4j plugins. This way, you can ensure when building the container that the correct plugin version for the Neo4j version of the container is used and that the resulting image encapsulates all Neo4j runtime dependencies.

Note:

link:The Neo4j Bloom plugin requires a license activation key, which needs to be placed in a directory accessible by the Neo4j Docker container, for example, mounted to */licenses* (default). To obtain a valid license, reach out to your Neo4j account representative or use the form Contact Neo4j.

Building a Docker container image that is based on the official Neo4j Docker image and does not override the official image's `ENTRYPOINT` and `COMMAND` is the recommended method to use with the Neo4j Helm chart, as shown in this example Dockerfile:

```
ARG  NEO4J_VERSION
FROM neo4j:${NEO4J_VERSION}
```

```
# copy my-plugins into the Docker image
COPY my-plugins/ /var/lib/neo4j/plugins

# install the apoc core plugin that is shipped with Neo4j
RUN cp /var/lib/neo4j/labs/apoc-* /var/lib/neo4j/plugins
```

Once the docker image has been built, push it to a container repository that is accessible to your Kubernetes cluster.

```
CONTAINER_REPOSITORY="my-container-repository.io"
IMAGE_NAME="my-neo4j"

# export this so that it's accessible as a docker build arg
export NEO4J_VERSION=5.26.25-enterprise

docker build --build-arg NEO4J_VERSION --tag ${CONTAINER_REPOSITORY}/${IMAGE_NAME}:${NEO4J_VERSION} .
docker push ${CONTAINER_REPOSITORY}/${IMAGE_NAME}:${NEO4J_VERSION}
```

To use the image that you have created, in the Neo4j Helm deployment's *values.yaml* file, set `image.customImage` to use the image.

Note:

Many plugins require additional Neo4j configuration to work correctly. Plugin configuration should be set on the `config` object in the Helm deployment's *values.yaml* file. In some cases, plugin configuration can cause Neo4j's strict config validation to fail. Strict config validation can be disabled by setting `server.config.strict_validation.enabled: "false"`.

Add plugins using a plugins volume

An alternative method for adding Neo4j plugins to a Neo4j Helm deployment uses a `plugins` volume mount. With this method, the plugin jar files are stored on a Persistent Volume that is mounted to the `/plugins` directory of the Neo4j container.

Note:

The Neo4j Bloom plugin requires a license activation key, which needs to be placed in a directory accessible by the Neo4j Docker container, for example, mounted to */licenses* (default). To obtain a valid license, reach out to your Neo4j account representative or use the form Contact Neo4j.

The simplest way to set up a persistent `plugins` volume is to share the Persistent Volume that is used for storing Neo4j data. This example shows how to configure that in the Neo4j Helm deployment *values.yaml* file:

```
# neo4j-values.yaml
volumes:
  data:
    # your data volume configuration
    ...

  plugins:
    mode: "share"
```

```
  share:
    name: "data"
```

Details of different ways to configure volume mounts are covered in Mapping volume mounts to persistent volumes.

The Neo4j container now has an empty */plugins* directory backed by a persistent volume. Plugin jar files can be copied onto the volume using `kubectl cp`. Because it is backed by a persistent volume, plugin files will persist even if the Neo4j pod is restarted or moved.

Note:

Neo4j loads plugins only on startup. Therefore, you must restart the Neo4j pod to load them once all plugins are in place.

For example:

```
# Copy plugin files into the Neo4j container
kubectl cp my-plugins/* <namespace>/<neo4j-pod-name>:/plugins/

# Restart Neo4j
kubectl rollout restart statefulset/<neo4j-statefulset-name>

# Verify plugins are still present after restart
kubectl exec <neo4j-pod-name> -- ls /plugins
```

Configure and install APOC core only

APOC core library is shipped with Neo4j and is located in the *labs* folder.

If APOC core is the *only* plugin that you want to add to Neo4j, it is not necessary to perform plugin installation as described above. Instead, you can configure the helm deployment to use APOC core by upgrading the deployment with these additional settings in the *values.yaml* file:

1. Configure APOC core by directly pointing to the location of the APOC core library in the *labs* folder and by loading and unrestricting the functions and procedures you need (for more details see APOC installation guide). For example:

   ```
   config:
     server.directories.plugins: "/var/lib/neo4j/labs"
     dbms.security.procedures.unrestricted: "apoc.*"
     server.config.strict_validation.enabled: "false"
     dbms.security.procedures.allowlist: "apoc.math.*,apoc.cypher.*"
   ```

2. Under `apoc_config`, configure the APOC settings that you want, for example:

   ```
   apoc_config:
     apoc.trigger.enabled: "true"
     apoc.jdbc.neo4j.url: "jdbc:foo:bar"
     apoc.import.file.enabled: "true"
   ```

3. Run `helm upgrade` to apply the changes:

```
helm upgrade <release-name> neo4j/neo4j -f values.yaml
```

4. After the Helm upgrade rollout is complete, verify that APOC core has been configured by running the following Cypher query using `cypher-shell` or Neo4j Browser:

```
RETURN apoc.version()
```

Configure credentials for the plugin's aliases using APOC-extended

From 5.11, the Neo4j Helm chart supports configuring credentials for the plugin's aliases using a Kubernetes secret mounted on the provided path. This feature is available `apoc.jdbc.<aliasname>.url` and `apoc.es.<aliasname>.url` via APOC-extended.

> **Note:**
>
> The secret must be created beforehand and must contain the key-named `URL`, otherwise, the Helm chart throws an error. For example: `kubectl create secret generic jdbcsecret --from -literal=URL="jdbc:mysql://30.0.0.0:3306/Northwind?user=root&password=password"`

Under `apoc_credentials`, configure `aliasName`, `secretName`, and `secretMountPath`. For example:

```
apoc_credentials: {}
#   jdbc:
#    aliasName: "jdbc"
#    secretName: "jdbcsecret"
#    secretMountPath: "/secret/jdbcCred"
#
#   elasticsearch:
#     aliasName: "es"
#     secretName: "essecret"
#     secretMountPath: "/secret/esCred"
```

Accessing Neo4j

A Neo4j running on Kubernetes is accessible via Kubernetes Services. Neo4j has a number of different interfaces for different application and operational purposes. For more details, see Neo4j ports.

Supported Kubernetes services

Neo4j supports the following Kubernetes services:

- **Default Service** — a ClusterIP service for application `neo4j`/`bolt` and `http(s)` connections to the Neo4j database, originating from inside the Kubernetes cluster.
- **Admin Service** — a "Headless" (DNS only) service that includes all Neo4j ports for admin connections to Neo4j inside Kubernetes. It is only available inside the Kubernetes cluster and access to it should be guarded. The Admin service can be used for Neo4j DBMS administration, performing backups, and collecting metrics.
- **Internal Service** — a "Headless" (DNS only) internal service that includes all Neo4j ports required for

the Neo4j cluster.

- **Neo4j** — a LoadBalancer service for application `neo4j`/`bolt` and `http(s)` connections originating from outside the Kubernetes cluster.

The following is a list of the default Kubernetes services per Neo4j interface and deployment type:

Neo4j Interface	Default Port	Default Service	Admin Service	Neo4j Service
Bolt (`neo4j://` and `bolt://` protocols)	`7687`	Yes	Yes *	Yes
Neo4j Browser HTTP	`7474`	Yes	Yes *	Yes
Neo4j Browser HTTPS	`7473`	Yes	Yes *	Yes
Neo4j Cypher HTTP API	`7474`	Yes	Yes *	Yes
Neo4j Cypher HTTPS API	`7473`	Yes	Yes *	Yes
Neo4j Backup	`6362`	No	Yes	No but configurable
Graphite Monitoring	`2003`	No	Yes	No
Prometheus Metrics	`2004`	No	Yes	No
Java Management Extensions (JMX)	`3637`	No	No but configurable	No

Neo4j Interface	Default Port	Default Service	Admin Service	Internal Service	Neo4j Service
Bolt (`neo4j://` and `bolt://` protocols)	`7687`	Yes	Yes *	Yes	Yes
Neo4j Browser HTTP	`7474`	Yes	Yes *	Yes	Yes
Neo4j Browser HTTPS	`7473`	Yes	Yes *	Yes	Yes
Neo4j Cypher HTTP API	`7474`	Yes	Yes *	Yes	Yes

Neo4j Interface	Default Port	Default Service	Admin Service	Internal Service	Neo4j Service
Neo4j Cypher HTTPS API	`7473`	Yes	Yes*	Yes	Yes
Neo4j Backup	`6362`	No	Yes	Yes	No
Graphite Monitoring	`2003`	No	No but configurable	No but configurable	No
Prometheus Metrics	`2004`	No	No but configurable	No but configurable	No
Java Management Extensions (JMX)	`3637`	No	No but configurable	No but configurable	No
Cluster discovery management	`5000`	No	No	Yes	No
Cluster transaction	`6000`	No	No	Yes	No
Cluster RAFT	`7000`	No	No	Yes	No
Cluster routing connector	`7688`	No	No	Yes	No

*The Admin service bypasses health checks. This allows it to be used to make connections for administrative purposes when the database is in an unhealthy state. However, you must not use it to connect from applications that require the database to be in a healthy state.

Applications accessing Neo4j from inside Kubernetes

Access Neo4j using DNS

To access Neo4j from an application in the same Kubernetes cluster use the Neo4j service DNS address `<release-name>.<namespace>.svc.<cluster domain>`.

The default cluster domain is `cluster.local` and the default namespace is `default`. Generally, the Neo4j service DNS address is *<release-name>.default.svc.cluster.local*.

For example, if using the release name `my-release` in the `default` namespace, the cluster's DNS address would be `my-release.default.svc.cluster.local`, and the `bolt` address for use with Neo4j drivers would be *neo4j://my-release.default.svc.cluster.local:7687*.

Tip:

To allow for an application running inside Kubernetes to access a Neo4j cluster, you can

also use the Neo4j headless service that is installed via the *neo4j/neo4j-cluster-headless-service* Helm chart. For more information and a detailed example, see Access the Neo4j cluster using headless service.

Access Neo4j using K8s label selector

Alternatively, the Neo4j service (default) in Kubernetes can be located using Kubernetes service discovery by searching with the label selector:

```
helm.neo4j.com/service=default/admin/internals,helm.neo4j.com/instance=<release-name>
```

For example:

```
# install neo4j
helm install "my-release" …
# lookup installed service
kubectl get service -l helm.neo4j.com/service=default,helm.neo4j.com/instance=my-release
```

```
helm.neo4j.com/service=neo4j,helm.neo4j.com/instance=<release-name>
```

The following is an example of how to look up the installed services:

```
# Neo4j service:

kubectl get service -l helm.neo4j.com/service=default,helm.neo4j.com/instance=my-release

# Admin service:

kubectl get service -l helm.neo4j.com/service=admin,helm.neo4j.com/instance=my-release

# internals service:

kubectl get service -l helm.neo4j.com/service=internals,helm.neo4j.com/instance=my-release
```

Applications accessing Neo4j from outside Kubernetes

To access Neo4j from an application outside the Kubernetes cluster, you can use a LoadBalancer service or an Ingress controller.

Access Neo4j using a LoadBalancer

Neo4j Helm chart provides a `LoadBalancer` service for accessing Neo4j from outside the Kubernetes cluster. The `LoadBalancer` service is created by default when installing the Neo4j Helm chart. The `LoadBalancer` service is configured to expose the Neo4j ports `7687`, `7474`, `7473`, and `6362` (backup) by default.

The external IP(s) of the `LoadBalancer` can be found using `kubectl`:

- The service name is based on the value of the `neo4j.name` — `<my-neo4j-name>-lb-neo4j`:

```
kubectl get service `<my-neo4j-name>-lb-neo4j` -ocustom-columns
=ip:.status.loadBalancer.ingress[].ip
```

- Using a label selector:

```
kubectl get service -l helm.neo4j.com/service=neo4j,helm.neo4j.com/name=<release-name>
-ocustom-columns=ip:.status.loadBalancer.ingress[].ip
```

If the Kubernetes `LoadBalancer` implementation that you are using supports setting a static IP, the IP address of the `LoadBalancer` can be configured in the Neo4j Helm release by setting `services.neo4j.spec.loadBalancerIP`. If a static IP address is not explicitly set, then Kubernetes does not guarantee that a dynamically assigned IP address will not change.

When exposing a Neo4j database on the Internet, it is recommended to use a static IP and configure SSL on the exposed services. For more information, see Configuring SSL.

If you have static IPs, you can associate DNS with them and obtain trusted certificates.

The ports that are exposed on the external service can be configured in the Helm release by changing the `services.neo4j` object. The default values are:

```
services:
  neo4j:
    enabled: true
    annotations: { }
    spec:
      type: LoadBalancer
      loadBalancerIP: NULL
    ports:
      http:
        enabled: true # Set this to false to remove HTTP from this service (this does not affect
whether http is enabled for the neo4j process)
        # uncomment to publish http on port 80 (neo4j default is 7474)
        #port: 80
        #targetPort: 7474
        #name: http
        #nodePort: <your-nodeport>, enabled only when type set to NodePort
      https:
        enabled: true # Set this to false to remove HTTPS from this service (this does not
affect whether https is enabled for the neo4j process)
        # uncomment to publish http on port 443 (neo4j default is 7473)
        #port: 443
        #targetPort: 7473
        #name: https
        #nodePort: <your-nodeport>, enabled only when type set to NodePort
      bolt:
        enabled: true # Set this to false to remove BOLT from this service (this does not affect
whether https is enabled for the neo4j process)
        # Uncomment to explicitly specify the port to publish Neo4j Bolt (7687 is the default)
        #port: 7687
        #targetPort: 7687
        #name: tcp-bolt
        #nodePort: <your-nodeport>, enabled only when type set to NodePort
      backup:
        enabled: false # Set this to true to expose backup port externally (n.b. this could have
security implications. Backup is not authenticated by default)
        # Uncomment to explicitly specify the port to publish Neo4j Backup (6362 is the default)
```

```
        #port: 6362
        #targetPort: 6362
        #name: tcp-backup
        #nodePort: <your-nodeport>, enabled only when type set to NodePort
```

Disabling/enabling a port on the `services.neo4j` object removes it from the load balancer but does not affect whether it is disabled/enabled in Neo4j.

Note:

Backup is not secure unless SSL-with-client-auth is enforced in the Neo4j configuration.

For a detailed example, see Access the Neo4j cluster from outside Kubernetes.

Customizing Kubernetes Resources

The Neo4j Helm chart creates various Kubernetes resources. Some of them can be customized by adding extra configuration to the helm deployment values file.

The following is a list of the supported K8s resources customizations:

Customization	*values.yaml* field	Type
Setting a pod securityContext for the Neo4j Pod	`securityContext`	`PodSecurityContext`
Adding annotations to Services	`services.default.annotations`	Annotations object for `ClusterIP` service.
	`services.admin.annotations`	Annotations object for headless (DNS) service.
	`services.neo4j.annotations`	Annotations object for `LoadBalancer` service.

Customization	*values.yaml* field	Type
Setting a pod securityContext for the Neo4j Pod	`securityContext`	`PodSecurityContext`

Customization	*values.yaml* field	Type
Adding annotations to Services	`services.default.annotations`	Annotations object for `ClusterIP` service.
	`services.admin.annotations`	Annotations object for headless (DNS) service.
	`services.internal.annotations`	Annotations object for internal service.
Adding annotations to Load Balancer Service	`services.neo4j.annotations`	Annotations object for `LoadBalancer` service.

Accessing Neo4j for DBMS administration and monitoring

The Neo4j Helm chart creates the admin service for the purposes of Neo4j administration. The admin service is a “Headless” service in Kubernetes and does not depend on Neo4j health checks. Therefore, it permits connections to Neo4j even if Neo4j is not healthy. In general, that is not desirable for applications but can be useful for administration and debugging.

Access Neo4j using DNS

To access the admin service inside Kubernetes use the DNS address *<release-name>-admin.<namespace>.svc.<cluster domain>*.

For example, if using the release name `my-release` in the `default` namespace, the cluster’s DNS address would be `my-release-admin.default.svc.cluster.local`.

The admin service can be used to access a range of Neo4j interfaces:

- Neo4j Bolt for Neo4j administration via Cypher commands.
- Neo4j Backup for taking database backups.
- Graphite for metrics collection.
- Prometheus for metrics collection.
- Java Management Extensions (JMX) for metrics collection and JVM administration.

Access Neo4j using `kubectl` for troubleshooting

To get an interactive `cypher-shell` console for troubleshooting, use this command:

```
kubectl run -it --rm --image neo4j:5.26.25 cypher-shell -- cypher-shell -a bolt://my-release-admin.default.svc.cluster.local
```

Generally, the `neo4j://` protocol is used for connecting to Neo4j. For troubleshooting, though, the direct `bolt://` protocol is used because it allows a connection in some situations where a `neo4j://` connection

will not succeed.

Accessing Neo4j using Kubernetes Ingress

The Neo4j Helm charts provide a Helm chart that allows you to use a Kubernetes Ingress to access Neo4j on port `:80` or `:443`. The Helm chart is called *neo4j/neo4j-reverse-proxy* and is available on the Neo4j Helm repository from version 5.12.0. For more information about Kubernetes Ingress, see the Kubernetes official documentation → Ingress.

The Helm chart creates a reverse proxy that is configured to route traffic to the Neo4j service URL using the `serviceName`, `namespace`, and `domain` values. For example, if the `serviceName` is *standalone-admin*, the `namespace` is *default*, and the `domain` is *cluster.local*, then the Neo4j service URL is *standalone-admin.default.svc.cluster.local*.
For Neo4j clusters, the Neo4j headless service can be used to route the traffic to the cluster instances. For more information and a detailed example of how to install the *neo4j/neo4j-cluster-headless-service* Helm chart, see Access the Neo4j cluster using headless service.

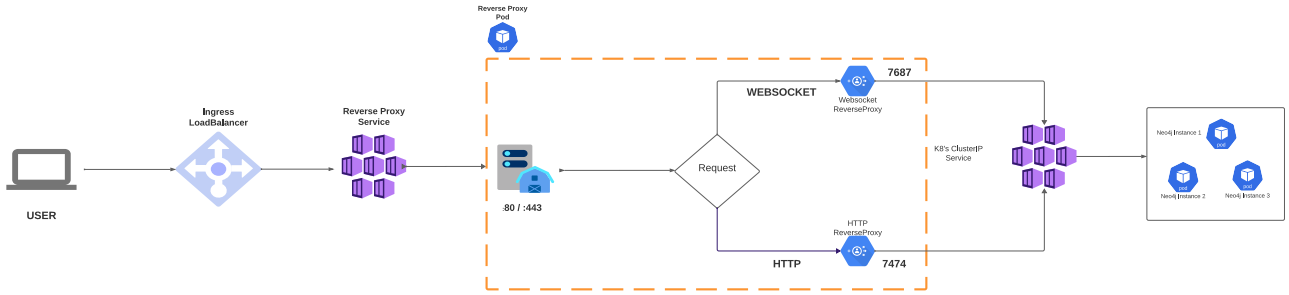


Figure 1. Reverse proxy flow diagram

The Reverse proxy Helm chart creates an HTTP server, which routes requests to either the Bolt reverse proxy or HTTP reverse proxy based on the request headers. Upon receiving a response, the Bolt reverse proxy updates the response to replace the Bolt port with either `:80` or `:443`.

From version 5.17.0, the Reverse proxy Helm chart supports defining privilege and access control settings for a Container. Make sure that you do not run Neo4j as a root user.

Configuration options

To see all configurable options, run the following command:

```
helm show values neo4j/neo4j-reverse-proxy
```

```
# Default values for neo4j reverse proxy helm chart

## @param nameOverride String to partially override common.names.fullname
nameOverride: ""
## @param fullnameOverride String to fully override common.names.fullname
fullnameOverride: ""

# Parameters for reverse proxy
reverseProxy:
  image: "neo4j/helm-charts-reverse-proxy:5.17.0"
  # Name of the kubernetes service. This service should have the ports 7474 and 7687 open.
  # This could be the admin service ex: "standalone-admin" or the loadbalancer service ex: "standalone"
created via the neo4j helm chart
```

```
  # serviceName , namespace , domain together will form the complete k8s service url. Ex: standalone-
admin.default.svc.cluster.local
  # When used against a cluster ensure the service being used is pointing to all the cluster instances.
  # This could be the loadbalancer from neo4j helm chart or the headless service installed via neo4j-
headless-service helm chart
  serviceName: ""
  # default is set to cluster.local
  domain: "cluster.local"

  # securityContext defines privilege and access control settings for a Container. Making sure that we
dont run Neo4j as root user.
  containerSecurityContext:
    allowPrivilegeEscalation: false
    runAsNonRoot: true
    runAsUser: 7474
    runAsGroup: 7474
    capabilities:
      drop: [ "ALL" ]

  podSecurityContext:
    runAsNonRoot: true
    runAsUser: 7474
    runAsGroup: 7474
    fsGroup: 7474
    fsGroupChangePolicy: "Always"


  # This assumes ingress-nginx controller or haproxy-ingress-controller is already installed in your
kubernetes cluster.
  # You can install ingress-nginx by following instructions on this link
https://github.com/kubernetes/ingress-nginx/blob/main/docs/deploy/index.md#quick-start
  # You can install haproxy-ingress by following instructions on this link https://haproxy-
ingress.github.io/docs/getting-started/
  ingress:
    enabled: true
    #default value is nginx. It can be either nginx or haproxy
    className: nginx
    annotations: {}
#      "demo": "value"
#      "demo2": "value2"
    host: ""
    tls:
      enabled: false
      config: []
#      - secretName: "demo2"
#        hosts:
#          - localhost
```

The following steps assume that you have a Kubernetes cluster running and a standalone Neo4j Helm chart installed. The standalone Neo4j has a Neo4j service with the name *standalone-admin*, and it has `:7474` an `:7687` opened. To verify that, run:

```
kubectl get all, pvc, pv, configmaps, secrets
```

You also need to have an Ingress controller for the Kubernetes Ingress to work. The following steps use the *Nginx Ingress Controller*. See Ingress-Nginx Controller official documentation for more information.

If you do not have one, you can use the following command to install it:

```
helm upgrade --install ingress-nginx ingress-nginx \
      --repo https://kubernetes.github.io/ingress-nginx \
      --namespace ingress-nginx --create-namespace
```

```
helm upgrade --install ingress-nginx ingress-nginx \
      --repo https://kubernetes.github.io/ingress-nginx \
      --namespace ingress-nginx --create-namespace --set
controller.service.externalTrafficPolicy=Local
```

Configure the Kubernetes Ingress

Configure the *ingress-values.yaml* file that you will use to install the Reverse proxy Helm chart.

Configure the *ingress-values.yaml* file to access Neo4j on port `:443`

The following example shows how to configure the *ingress-values.yaml* file to access Neo4j on port `:443`:

1. Create a Kubernetes secret containing the Ingress self-signed certificates and then create the *ingress-values.yaml* file.
   a. Create a directory for the Ingress self-signed certificates:

   ```
   mkdir certs
   cd certs
   ```

   b. Create Ingress self-signed certificates:

   ```
   openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout ingress.key -out ingress.cert -subj
   "/CN=localhost/O=neo4j" -addext "subjectAltName = DNS:localhost"
   ```

   c. Create Kubernetes secret using the Ingress self-signed certificates:

   ```
   kubectl create secret tls ingress-cert --key /path/to/your/certs/ingress.key --cert
   /path/to/your/certs/ingress.cert
   ```

2. Configure the *ingress-values.yaml* file with the correct values for the `serviceName` and `secretName`. Ensure that the `secretName` is the same as the one created in the previous step. Enable TLS by setting `tls.enabled` to `true`.

   ```
   reverseProxy:
       image: neo4j/helm-charts-reverse-proxy:5.12.0
       serviceName: "standalone-admin"
       ingress:
           enabled: true
           tls:
               enabled: true
               config:
                   - secretName: ingress-cert
                     hosts:
                       - localhost
   ```

Configure the *ingress-values.yaml* file to access Neo4j on port `:80`

Alternatively, if you want to access Neo4j on port `:80`, leave `tls.enabled` with its default value `false`, and

create the *ingress-values.yaml* file with the following content:

```
reverseProxy:
    #Use image only when need a specific version or using your internal artifactory.
    #Otherwise let it default to what is in the values.yaml
    #image: neo4j/helm-charts-reverse-proxy:5.12.0
    serviceName: "standalone-admin"
    ingress:
        enabled: true
        tls:
            enabled: false
```

Install the Reverse proxy Helm chart

Install the Reverse proxy Helm chart using the *ingress-values.yaml* file that you have created:

```
helm install rp neo4j/neo4j-reverse-proxy -f /path/to/your/ingress-values.yaml
```

Access your data via Neo4j Browser

1. Get the Ingress LoadBalancer IP:

   ```
   kubectl get ingress/rp-reverseproxy-ingress -n default -o jsonpath
   ='{.status.loadBalancer.ingress[0].ip}'
   ```

2. Open Neo4j Browser on *https://INGRESS_IP:443* or *http://INGRESS_IP:80* and log in with your credentials.

Access your data via Cypher Shell

Alternatively, if you want to use Cypher Shell to access your data via Nginx Ingress Controller only, you need to create a `configmap`, because Cypher Shell expects a TCP connection and Ingress is an HTTP connection. For more information about exposing TCP/UDP services, see Ingress-Nginx Controller official documentation → Exposing TCP and UDP services.

1. Create a `configmap` with the following content:

   ```
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: tcp-services
     namespace: ingress-nginx
   data:
     9000: "default/standalone-admin:7687"
   ```

2. Apply the `configmap`:

   ```
   kubectl apply -f /path/to/your/nginx-tcp.yaml
   ```

3. Update the Ingress controller LoadBalancer service to use the port :9000:

   a. Get the IP address of the Ingress controller:

```
kubectl get svc -n ingress-nginx
```

b. Open the Ingress controller service for editing:

```
kubectl edit svc ingress-nginx-controller -n ingress-nginx -o yaml
```

c. Add the following lines to the `spec.ports` section:

```
- name: proxied-tcp-9000
  port: 9000
  protocol: TCP
  targetPort: 9000
```

d. Save the changes and exit the editor.

4. Update the Ingress controller deployment to use the `configmap`:

a. Open the Ingress controller deployment for editing:

```
kubectl edit deployment ingress-nginx-controller -n ingress-nginx
```

b. Add the following lines to the `spec.template.spec.containers.args` section:

```
- --tcp-services-configmap=ingress-nginx/tcp-services
```

c. Save the changes and exit the editor.

d. Verify that the changes are applied by running `kubectl get all -n ingress-nginx`. You should see the new port :9000 in the Ingress controller deployment.

5. Get the IP address of the Ingress controller:

```
kubectl get ingress
```

```
NAME                    CLASS    HOSTS   ADDRESS        PORTS   AGE
rp-reverseproxy-igress   nginx    *       34.89.91.112   80      2m
```

6. Connect to the Neo4j database using Cypher Shell:

```
cypher-shell -a neo4j://34.89.91.112:9000 -u neo4j -p <password>
```

Importing data

There is a wide range of ways to import data from files into a Neo4j instance. This page describes the most common ways to import data into a Neo4j instance running on a Kubernetes cluster.

Importing data into Neo4j on Kubernetes

The Neo4j Helm chart configures a volume mount at */import* as the Neo4j *import* directory, as described in Default file locations. You place all the files that you want to import in this volume.

To import data from CSV files into Neo4j, use the command `neo4j-admin database import` or the Cypher query `LOAD CSV`.

- The `neo4j-admin database import` command can be used to do batch imports of large amounts of data into a previously unused database and can only be performed once per database.
- `LOAD CSV` Cypher statement can be used to import small to medium-sized CSV files into an existing database. `LOAD CSV` can be run as many times as needed and does not require an empty database. For details, see Cypher manual → `LOAD CSV`.

> **Note:**
>
> Depending on your Neo4j configuration, some methods support fetching data to import from a remote location (e.g., using HTTP or fetching from cloud object storage). Therefore, it is not always necessary to place the source data files in the Neo4j *import* directory.

Configure the import volume mount

The default configuration of the `/import` volume mount is to share the `/data` volume mount. Generally, this is sufficient, and it is unnecessary to explicitly configure an *import* volume in the Helm deployment's *values.yaml* file. For the full details of configuring volume mounts for a Neo4j Helm deployment, see Volume mounts and persistent volumes.

This example shows how to configure `/import` to use a dynamically provisioned Persistent Volume of the default `StorageClass`:

```
volumes:
  import:
    mode: "defaultStorageClass"
    defaultStorageClass:
      requests:
        storage: 100Gi
```

Copy files to the *import* volume using `kubectl cp`

Files can be copied to the *import* volume using `kubectl cp`. This example shows how to copy a local directory `my-files/` to `/import/files-1` to a Neo4j instance with the release name `my-graph-db` in the namespace `default`.

```
kubectl cp my-files/ default/my-graph-db-0:/import/files-1

# Validate: list the contents of /import/files-1
kubectl exec my-graph-db-0 -- ls /import/files-1
```

Instead of using `kubectl cp`, data can also be loaded onto the `/import` directory by:

- using an additional container or `initContainer` to load data.

- using `kubectl exec` to run commands to load data.
- mounting a volume that is already populated with data.

> **Note:**
>
> Data must be placed in the volume's `/import` directory.

Use `neo4j-admin database import`

The simplest way to run `neo4j-admin database import` is to use `kubectl exec` to run it in the Neo4j container. However, running `neo4j-admin database import` to perform a large import in the same container as the Neo4j process may cause resource contention problems, including causing either or both processes to be OOM Killed by the node operating system. To avoid this, either use a separate container or `initContainer` or place the Neo4j Helm deployment in offline maintenance mode to run `neo4j-admin database import`.

`neo4j-admin database import` cannot be used to replace an existing database while Neo4j is running. To replace an existing database, either `DROP` the database or put the Neo4j Helm deployment into offline maintenance mode before running `neo4j-admin database import`.

Alternative approach

An alternative approach to importing data into Neo4j is to run a separate Neo4j standalone instance outside Kubernetes, perform the import on that Neo4j instance, and then copy the resulting database into the Kubernetes-based Neo4j instance using the backup and restore or dump and load procedures.

Monitoring

You can monitor a Neo4j DBMS running on Kubernetes using the same mechanisms as you would for a Neo4j running on-prem.

Logging

When using the Helm chart, Neo4j logging output is written to files in the `/logs` directory. This directory is mounted on a `PersistentVolume` so that logs are persisted if the pod is moved or restarted. For full details of Neo4j logging, see Neo4j logging.

- To view the Neo4j user log (*neo4j.log*), use the command `kubectl exec`:

  Follow neo4j.log

  ```
  kubectl exec <neo4j-pod-name> -- tail -f /logs/neo4j.log
  ```

- To copy the log files from a Neo4j instance, use `kubectl cp`:

  Copy all logs

  ```
  $ kubectl cp <neo4j-pod-name>:/logs neo4j-logs/
  ```

```
$ ls neo4j-logs
debug.log       neo4j.log       query.log       security.log
```

Log collection

The Neo4j log output can be collected from the log files and sent to a unified location using tools, such as Fluentd (https://www.fluentd.org) or Logstash (https://www.elastic.co/logstash). It is recommended to run these either as "sidecar" containers in the Neo4j pods or as separate DaemonSets.

- For more information about Pods and the sidecar pattern, see Kubernetes Pod documentation.
- For more information about DaemonSets, see Kubernetes DaemonSet documentation.
- For more information and examples of these logging patterns, see Kubernetes cluster administration documentation.

Metrics

If Neo4j is configured to listen for Graphite, JMX, or Prometheus connections for metrics, those services can be accessed as described in Accessing Neo4j.

The Helm chart supports standard Neo4j metrics configuration settings, for example:

```
# To listen for Prometheus connections
# Neo4j configuration (yaml format)
config:
  server.metrics.prometheus.enabled: "true"
  server.metrics.prometheus.endpoint: "0.0.0.0:2004"
```

```
# To publish Graphite connections
# Neo4j configuration (yaml format)
config:
  server.metrics.graphite.enabled: "true"
  server.metrics.graphite.interval: "3s"
  server.metrics.graphite.server: "graphite.default.svc.cluster.local:2003"
```

```
# To write CSV metrics
# Neo4j configuration (yaml format)
config:
  server.metrics.csv.enabled: "true"
  server.metrics.csv.interval: "10s"
```

```
# To enable JMX
# Neo4j configuration (yaml format)
config:
  server.metrics.jmx.enabled: "true"
```

For more information and examples, see Neo4j metrics.

Operations

This section describes some maintenance operations when running Neo4j in a Kubernetes cluster.

It covers the following topics:

- Maintenance mode
- Reset the neo4j user password
- Restart a Neo4j pod after a configuration change
- Dump and load databases (offline)
- Back up and restore a single database (online)
- Upgrade Neo4j on Kubernetes
- Migrate Neo4j from the Labs Helm charts to the Neo4j Helm charts (offline)
- Scale a Neo4j deployment
- Use custom images from private registries
- Assign Neo4j pods to specific nodes

Maintenance modes

Neo4j supports two maintenance modes: online and offline, which you can use to perform different maintenance tasks.

Online Maintenance

Online maintenance does not require stopping the `neo4j` process. It is performed using the command `kubectl exec`.

To directly run tasks:

```
kubectl exec <release-name>-0 -- neo4j-admin database info --from-path=/var/lib/neo4j/data/databases
--expand-commands
```

Note:

All `neo4j-admin` commands need the `--expand-commands` flag to run in the Neo4j container. This is because the Neo4j Helm chart defines the Neo4j configuration using command expansion to dynamically resolve some configuration parameters at runtime.

To run a series of commands, use an interactive shell:

```
kubectl exec -it <release-name>-0 -- bash
```

Note:

Processes executed using `kubectl exec` count towards the Neo4j container's memory allocation. Therefore, running tasks that use a significant amount of memory or running Neo4j in an extremely memory-constrained configuration could cause the Neo4j container to be terminated by the underlying Operating System.

Offline Maintenance

You use the Neo4j offline maintenance mode to perform maintenance tasks that require Neo4j to be offline. In this mode, the `neo4j` process is not running. However, the Neo4j Pod does run, but it never reaches the status `READY`.

Put the Neo4j instance in offline mode

1. To put the Neo4j instance in offline maintenance mode, you set the `offlineMaintenanceModeEnabled: true` and upgrade the helm release.
   - You can do that by using the *values.yaml* file:
     a. Open your *values.yaml* file and add `offlineMaintenanceModeEnabled: true` to the `neo4j` object:

     ```
     neo4j:
      offlineMaintenanceModeEnabled: true
     ```

     b. Run `helm upgrade` to apply the changes:

     ```
     helm upgrade <release-name> neo4j/neo4j -f values.yaml
     ```

   - Alternatively, you can set `neo4j.offlineMaintenanceModeEnabled` to `true` as part of the `helm upgrade` command:

     ```
     helm upgrade <release-name> neo4j/neo4j --version=5.26.25 --reuse-values --set
     neo4j.offlineMaintenanceModeEnabled=true
     ```

2. Poll `kubectl get pods` until the pod has restarted (`STATUS` = `Running`).

   ```
   kubectl get pod <release-name>-0
   ```

3. Connect to the pod with an interactive shell:

   ```
   kubectl exec -it "<release-name>-0" -- bash
   ```

4. View running java processes:

   ```
   jps
   ```

   ```
   19 Jps
   ```

   The result shows no running java process other than `jps` itself.

Run task in offline mode

Offline maintenance tasks are performed using the command `kubectl exec`.

- To directly run tasks:

```
kubectl exec <release-name>-0 -- neo4j-admin database info --from-path=/var/lib/neo4j/data/databases
--expand-commands
```

- To run a series of commands, use an interactive shell:

```
kubectl exec -it <release-name>-0 -- bash
```

- For long-running commands, use a shell and run tasks using `nohup` so they continue if the `kubectl exec` connection is lost:

```
kubectl exec -it <release-name>-0 -- bash
  $ nohup neo4j-admin database check neo4j --expand-commands &>job.out </dev/null &
  $ tail -f job.out
```

Put the Neo4j DBMS in online mode

When you finish with the maintenance tasks, return the Neo4j instance to normal operation:

- You can do that by using the *values.yaml* file:
  1. Open your *values.yaml* file and add `offlineMaintenanceModeEnabled: false` to the `neo4j` object:

     ```
     neo4j:
      offlineMaintenanceModeEnabled: false
     ```

  2. Run `helm upgrade` to apply the changes:

     ```
     helm upgrade <release-name> neo4j/neo4j -f values.yaml
     ```

- Alternatively, you can run `helm upgrade` with the flag set to `false`:

```
helm upgrade <release-name> neo4j/neo4j-standalone --version=5.26.25 --reuse-values --set
neo4j.offlineMaintenanceModeEnabled=false
```

Reset the `neo4j` user password

You reset the `neo4j` user password by disabling authentication and then re-enabling it.

1. In the *values.yaml* file, set `dbms.security.auth_enabled:` to `false` to disable the authentication:

> **Note:**
>
> All Neo4j `config` values must be YAML strings, not YAML booleans. Therefore, make sure you put quotes around values, such as `"true"` or `"false"`, so that they are handled correctly by Kubernetes.

```
# Neo4j Configuration (yaml format)
config:
  dbms.security.auth_enabled: "false"
```

2. Run the following command to apply the changes:

```
helm upgrade <release-name> neo4j/neo4j -f values.yaml
```

Authentication is now disabled.

3. Connect with `cypher-shell` and set the desired password:

```
ALTER USER neo4j SET PASSWORD '<new-password>'
```

4. Update the Neo4j configuration to enable authentication:

```
# Neo4j Configuration (yaml format)
config:
  dbms.security.auth_enabled: "true"
```

5. Run the following command to apply the update and re-enable authentication:

```
helm upgrade <release-name> neo4j/neo4j -f values.yaml
```

Authentication is now enabled, and the Neo4j user password has been reset to the desired password.

Restart a Neo4j pod after configuration change

When you update ConfigMaps or secrets (for example, updating GDS or Bloom license values), you have to restart your Neo4j pod so that the changes take effect.

You have two options for restarting:

- To run the `kubectl delete pod` command as follows:

```
kubectl delete pod <pod_name>
```

You manually delete the Neo4j pod. Kubernetes will automatically recreate it with the updated configuration.

- To use the `kubectl rollout restart` command:

```
kubectl rollout restart statefulset/<neo4j-statefulset-name>
```

A rollout restart means a gradual recreation of a pod — the old pod is not removed entirely until the new pod is running successfully. This process is safer and ensures that your application remains available throughout the restart process.

If you need to change Neo4j password, see Reset the `neo4j` user password.

For information on how to install and manage plugins, refer to the Plugins.

Dump and load databases (offline)

You can use the `neo4j-admin database dump` command to make a full backup (an archive) of an **offline** database(s) and `neo4j-admin database load` to load it back into a Neo4j deployment. These operations are performed in offline maintenance mode.

Dump the `neo4j` and `system` databases

1. Put your Neo4j in offline mode.
2. Dump `neo4j` and `system` databases:

```
neo4j-admin database dump --expand-commands system --to-path=/backups && neo4j-admin database dump
--expand-commands neo4j --to-path=/backups
```

3. Put your Neo4j back to online mode.
4. Verify that Neo4j is working by refreshing Neo4j Browser.

> **Tip:**
>
> For information about the command syntax, options, and usage, see Back up an offline database.

Load the `neo4j` and `system` databases

1. Put your Neo4j in offline mode..
2. Run `neo4j-admin database load` commands:

```
neo4j-admin database load --expand-commands system --from-path=/backups --overwrite-destination=true
&& neo4j-admin database load --expand-commands neo4j --from-path=/backups --overwrite-destination=true
```

> **Tip:**
>
> For information about the command syntax, options, and usage, see Restore a database dump.

3. Put your Neo4j back to online mode.
4. Verify that Neo4j is working by refreshing Neo4j Browser.

Back up, aggregate, and restore (online)

> **Note:**

For performing backups, Neo4j uses the *Admin Service*, which is only available inside the Kubernetes cluster and access to it should be guarded. For more information, see Accessing Neo4j.

Prepare to back up a database(s) to a cloud provider (AWS, GCP, and Azure) bucket

You can perform a backup of a Neo4j database(s) to any cloud provider (AWS, GCP, and Azure) bucket using the *neo4j/neo4j-admin* Helm chart. From Neo4j 5.10, the *neo4j/neo4j-admin* Helm chart also supports performing a backup of multiple databases. From 5.13, the *neo4j/neo4j-admin* Helm chart also supports workload identity integration for GCP, AWS, and Azure. From 5.14, the *neo4j/neo4j-admin* Helm chart also supports MinIO (an AWS S3-compatible object storage API) for Non-TLS/SSL endpoints.

Prerequisites

Before you can back up a database and upload it to your bucket, verify that you have the following:

- A cloud provider bucket (AWS, GCP, or Azure) with read and write access to be able to upload the backup.
- Credentials to access the cloud provider bucket, such as a service account JSON key file for GCP, a credentials file for AWS, or storage account credentials for Azure.
- A service account with workload identity if you want to use workload identity integration to access the cloud provider bucket.
  - For more information on setting up a service account with workload identity on GCP and AWS, see:
    - Google Kubernetes Engine (GKE) → Use Workload Identity
    - Amazon EKS → Configuring a Kubernetes service account to assume an IAM role
  - For more information on setting up an Azure storage account with workload identity, Microsoft Azure → Use Microsoft Entra Workload ID with Azure Kubernetes Service (AKS)
- A Kubernetes cluster running on one of the cloud providers with the Neo4j Helm chart installed. For more information, see Quickstart: Deploy a standalone instance or Quickstart: Deploy a cluster.
- MinIO server (an AWS S3-compatible object storage API) if you want to push your backups to a MinIO bucket. For more information, see MinIO official documentation.
- The latest Neo4j Helm charts. You can update the repository to get the latest charts using `helm repo update`.

Create a Kubernetes secret

You can create a Kubernetes secret with the credentials that can access the cloud provider bucket using one of the following options:

Create the secret named `gcpcreds` using your GCP service account JSON key file. The JSON key file contains all the details of the service account that has access to the bucket.

```
kubectl create secret generic gcpcreds --from-file=credentials=/path/to/gcpcreds.json
```

1. Create a credentials file in the following format:

```
[ default ]
region = us-east-1
aws_access_key_id = <your-aws_access_key_id>
aws_secret_access_key = <your-aws_secret_access_key>
```

2. Create the secret named `awscreds` via the credentials file:

```
kubectl create secret generic awscreds --from-file=credentials=/path/to/your/credentials
```

1. Create a credentials file in the following format:

```
AZURE_STORAGE_ACCOUNT_NAME=<your-azure-storage-account-name>
AZURE_STORAGE_ACCOUNT_KEY=<your-azure-storage-account-key>
```

2. Create the secret named `azurecred` via the credentials file:

```
kubectl create secret generic azurecred --from-file=credentials=/path/to/your/credentials
```

Configure the backup parameters

You can configure the backup parameters in the *backup-values.yaml* file either by using the `secretName` and `secretKeyName` parameters or by mapping the Kubernetes service account to the workload identity integration.

> **Note:**
>
> The following examples show the minimum configuration required to perform a backup to a cloud provider bucket. For more information about the available backup parameters, see Backup parameters.

Configure the *backup-values.yaml* file using the `secretName` and `secretKeyName` parameters

```
neo4j:
  image: "neo4j/helm-charts-backup"
  imageTag: "5.26.25"
  jobSchedule: "* * * * *"
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
  backoffLimit: 3
```

```
backup:
  bucketName: "my-bucket"
  databaseAdminServiceName:  "standalone-admin" #This is the Neo4j Admin Service name.
  database: "neo4j,system"
  cloudProvider: "gcp"
  secretName: "gcpcreds"
  secretKeyName: "credentials"

consistencyCheck:
  enabled: true
```

```
neo4j:
  image: "neo4j/helm-charts-backup"
  imageTag: "5.26.25"
  jobSchedule: "* * * * *"
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
  backoffLimit: 3

backup:
  bucketName: "my-bucket"
  databaseAdminServiceName:  "standalone-admin"
  database: "neo4j,system"
  cloudProvider: "aws"
  secretName: "awscreds"
  secretKeyName: "credentials"

consistencyCheck:
  enabled: true
```

```
neo4j:
  image: "neo4j/helm-charts-backup"
  imageTag: "5.26.25"
  jobSchedule: "* * * * *"
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
  backoffLimit: 3

backup:
  bucketName: "my-bucket"
  databaseAdminServiceName:  "standalone-admin"
  database: "neo4j,system"
  cloudProvider: "azure"
  secretName: "azurecreds"
  secretKeyName: "credentials"

consistencyCheck:
  enabled: true
```

Configure the *backup-values.yaml* file using service account workload identity integration

In certain situations, it may be useful to assign a Kubernetes Service Account with workload identity integration to the Neo4j backup pod. This is particularly relevant when you want to improve security and have more precise access control for the pod. Doing so ensures that secure access to resources is granted based on the pod's identity within the cloud ecosystem. For more information on setting up a service account with workload identity, see Google Kubernetes Engine (GKE) → Use Workload Identity, Amazon EKS → Configuring a Kubernetes service account to assume an IAM role, and Microsoft Azure → Use Microsoft Entra Workload ID with Azure Kubernetes Service (AKS).

To configure the Neo4j backup pod to use a Kubernetes service account with workload identity, set `serviceAccountName` to the name of the service account to use. For Azure deployments, you also need to set the `azureStorageAccountName` parameter to the name of the Azure storage account, where the backup files will be uploaded. For example:

```
neo4j:
  image: "neo4j/helm-charts-backup"
  imageTag: "5.26.25"
  jobSchedule: "* * * * *"
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
  backoffLimit: 3

backup:
  bucketName: "my-bucket"
  databaseAdminServiceName:  "standalone-admin" #This is the Neo4j Admin Service name.
  database: "neo4j,system"
  cloudProvider: "gcp"
  secretName: ""
  secretKeyName: ""

consistencyCheck:
  enabled: true

serviceAccountName: "demo-service-account"
```

```
neo4j:
  image: "neo4j/helm-charts-backup"
  imageTag: "5.26.25"
  jobSchedule: "* * * * *"
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
  backoffLimit: 3

backup:
  bucketName: "my-bucket"
  databaseAdminServiceName:  "standalone-admin"
  database: "neo4j,system"
  cloudProvider: "aws"
  secretName: ""
  secretKeyName: ""

consistencyCheck:
  enabled: true

serviceAccountName: "demo-service-account"
```

```
neo4j:
  image: "neo4j/helm-charts-backup"
  imageTag: "5.26.25"
  jobSchedule: "* * * * *"
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
  backoffLimit: 3

backup:
  bucketName: "my-bucket"
  databaseAdminServiceName:  "standalone-admin"
  database: "neo4j,system"
  cloudProvider: "azure"
  azureStorageAccountName: "storageAccountName"
```

```
consistencyCheck:
  enabled: true

serviceAccountName: "demo-service-account"
```

The */backups* mount created by default is an *emptyDir* type volume. This means that the data stored in this volume is not persistent and will be lost when the pod is deleted. To use a persistent volume for backups add the following section to the *backup-values.yaml* file:

```
tempVolume:
  persistentVolumeClaim:
    claimName: backup-pvc
```

Note:

You need to create the persistent volume and persistent volume claim before installing the *neo4j-admin* Helm chart. For more information, see Volume mounts and persistent volumes.

Configure S3-compatible storage endpoints

The backup system supports any S3-compatible storage service. You can configure both TLS and non-TLS endpoints using the following parameters in your *backup-values.yaml* file:

```
backup:
  # Specify your S3-compatible endpoint (e.g., https://s3.amazonaws.com or your custom endpoint)
  s3Endpoint: "https://s3.custom-provider.com"

  # Enable TLS for secure connections (default: false)
  s3EndpointTLS: true

  # Optional: Provide a base64-encoded CA certificate for custom certificate authorities
  s3CACert: "base64_encoded_ca_cert_data"

  # Optional: Skip TLS verification (not recommended for production)
  s3SkipVerify: false
```

The following are examples of how to configure the backup system for different S3-compatible storage providers:

AWS S3 standard endpoint

```
neo4j:
  image: "neo4j/helm-charts-backup"
  imageTag: "5.26.25"
  jobSchedule: "* * * * *"
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
  backoffLimit: 3

backup:
  bucketName: "my-bucket"
  databaseAdminServiceName: "standalone-admin"
  s3Endpoint: "https://s3.amazonaws.com"
  s3EndpointTLS: true
  database: "neo4j,system"
  cloudProvider: "aws"
```

```
  secretName: "awscreds"
  secretKeyName: "credentials"

consistencyCheck:
  enabled: true
```

Custom S3-compatible provider with self-signed certificate

```
backup:
  bucketName: "my-bucket"
  s3Endpoint: "https://custom-s3.example.com"
  s3EndpointTLS: true
  s3CACert: "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0t..."  # Base64-encoded CA cert
  cloudProvider: "aws"
  secretName: "awscreds"
  secretKeyName: "credentials"
```

Legacy MinIO support

```
backup:
  bucketName: "my-bucket"
  databaseAdminServiceName: "standalone-admin"
  minioEndpoint: "http://minio.example.com:9000"  # Deprecated: Use s3Endpoint instead
  database: "neo4j,system"
  cloudProvider: "aws"
  secretName: "awscreds"
  secretKeyName: "credentials"
```

Important:

- The `s3EndpointTLS` parameter must be set to `true` when using HTTPS endpoints.
- When using custom CA certificates, provide them base64-encoded in the `s3CACert` parameter.
- The `s3SkipVerify` parameter should only be used in development environments.
- Legacy MinIO support through the `minioEndpoint` parameter is deprecated - use `s3Endpoint` instead.

Prepare to back up a database(s) to on-premises storage

Introduced in 5.16

You can perform a backup of a Neo4j database(s) to on-premises storage using the *neo4j/neo4j-admin* Helm chart. When configuring the *backup-values.yaml* file, keep the “cloudProvider” field empty and provide a persistent volume in the `tempVolume` section to ensure the backup files are persistent if the pod is deleted.

Note:

You need to create the persistent volume and persistent volume claim before installing the *neo4j-admin* Helm chart. For more information, see Volume mounts and persistent volumes.

For example:

```
neo4j:
  image: "neo4j/helm-charts-backup"
  imageTag: "5.26.25"
  jobSchedule: "* * * * *"
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
  backoffLimit: 3

backup:
  bucketName: "my-bucket"
  databaseAdminServiceName:  "standalone-admin"
  database: "neo4j,system"
  cloudProvider: ""

consistencyCheck:
  enabled: true

tempVolume:
  persistentVolumeClaim:
    claimName: backup-pvc
```

Backup parameters

To see what options are configurable on the Helm chart use `helm show values` and the Helm chart *neo4j/neo4j-admin*.
From Neo4j 5.10, the *neo4j/neo4j-admin* Helm chart also supports assigning your Neo4j pods to specific nodes using `nodeSelector` labels, and from Neo4j 5.11, using affinity/anti-affinity rules or tolerations. For more information, see Assigning backup pods to specific nodes and the Kubernetes official documentation on Affinity and anti-affinity rules and Taints and Tolerations.

For example:

```
helm show values neo4j/neo4j-admin
```

```
## @param nameOverride String to partially override common.names.fullname
nameOverride: ""
## @param fullnameOverride String to fully override common.names.fullname
fullnameOverride: ""
# disableLookups will disable all the lookups done in the helm charts
# This should be set to true when using ArgoCD since ArgoCD uses helm template and the helm lookups will
fail
# You can enable this when executing helm commands with --dry-run command
disableLookups: false

neo4j:
  image: "neo4j/helm-charts-backup"
  imageTag: "5.26.25"
  podLabels: {}
#    app: "demo"
#    acac: "dcdddc"
  podAnnotations: {}
#    ssdvvs: "svvvsvs"
#    vfsvswef: "vcfvgb"
  # define the backup job schedule . default is * * * * *
  jobSchedule: ""
  # default is 3
  successfulJobsHistoryLimit:
  # default is 1
```

```
  failedJobsHistoryLimit:
  # default is 3
  backoffLimit:
  #add labels if required
  labels: {}

backup:
  # Ensure the bucket is already existing in the respective cloud provider
  # In case of azure the bucket is the container name in the storage account
  # bucket: azure-storage-container
  bucketName: ""
  # Specify multiple backup endpoints as comma-separated string
  # e.g. "10.3.3.2:6362,10.3.3.3:6362,10.3.3.4:6362"
  databaseBackupEndpoints: ""
  #ex: standalone-admin.default.svc.cluster.local:6362
  # admin service name -  standalone-admin
  # namespace - default
  # cluster domain - cluster.local
  # port - 6362

  #ex: 10.3.3.2:6362
  # admin service ip - 10.3.3.2
  # port - 6362

  databaseAdminServiceName: ""
  databaseAdminServiceIP: ""
  #default name is 'default'
  databaseNamespace: ""
  #default port is 6362
  databaseBackupPort: ""
  #default value is cluster.local
  databaseClusterDomain: ""
  # specify S3-compatible endpoint (e.g., http://s3.amazonaws.com or your custom S3 endpoint)
  # This can be any S3-compatible endpoint including AWS S3, MinIO, or other S3-compatible storage
services
  # For TLS endpoints (https), set s3EndpointTLS to true
  s3Endpoint: ""
  # Enable TLS for S3 endpoint (default: false)
  s3EndpointTLS: false
  # Optional: Base64-encoded CA certificate for S3 endpoint TLS verification
  # Only needed for self-signed certificates or private CA
  s3CACert: ""
  # Optional: Skip TLS verification (not recommended for production)
  s3SkipVerify: false
  #name of the database to backup ex: neo4j or neo4j,system (You can provide command separated database
names)
  # In case of comma separated databases failure of any single database will lead to failure of complete
operation
  database: ""
  # cloudProvider can be either gcp, aws, or azure
  # if cloudProvider is empty then the backup will be done to the /backups mount.
  # the /backups mount can point to a persistentVolume based on the definition set in tempVolume
  cloudProvider: ""



  # name of the kubernetes secret containing the respective cloud provider credentials
  # Ensure you have read,write access to the mentioned bucket
  # For AWS :
  # add the below in a file and create a secret via
  # 'kubectl create secret generic awscred --from-file=credentials=/demo/awscredentials'

  #  [ default ]
  #  region = us-east-1
  #  aws_access_key_id = XXXXX
  #  aws_secret_access_key = XXXX

  # For AZURE :
  # add the storage account name and key in below format in a file create a secret via
  # 'kubectl create secret generic azurecred --from-file=credentials=/demo/azurecredentials'

  #  AZURE_STORAGE_ACCOUNT_NAME=XXXX
  #  AZURE_STORAGE_ACCOUNT_KEY=XXXX

  # For GCP :
  # create the secret via the gcp service account json key file.
  # ex: 'kubectl create secret generic gcpcred --from-file=credentials=/demo/gcpcreds.json'
```

```
  secretName: ""
  # provide the keyname used in the above secret
  secretKeyName: ""
  # provide the azure storage account name
  # this to be provided when you are using workload identity integration for azure
  azureStorageAccountName: ""
  #setting this to true will not delete the backup files generated at the /backup mount
  keepBackupFiles: true

  #Below are all neo4j-admin database backup flags / options
  #To know more about the flags read here : https://neo4j.com/docs/operations-manual/current/backup-
restore/online-backup/
  pageCache: ""
  includeMetadata: "all"
  type: "AUTO"
  keepFailed: false
  parallelRecovery: false
  verbose: true
  heapSize: ""

  # https://neo4j.com/docs/operations-manual/current/backup-restore/aggregate/
  # Performs aggregate backup. If enabled, NORMAL BACKUP WILL NOT BE DONE only aggregate backup
  # fromPath supports only s3 or local mount. For s3 , please set cloudProvider to aws and use either
serviceAccount or creds
  aggregate:
    enabled: false
    verbose: true
    keepOldBackup: false
    parallelRecovery: false
    # Only AWS S3 or local mount paths are supported
    # For S3 provide the complete path , Ex: s3://bucket1/bucket2
    fromPath: ""
    # database name to aggregate. Can contain * and ? for globbing.
    database: ""
    # Optional temporary directory for aggregation process
    # If not specified, will use the backup directory
    tempDir: ""

#Below are all neo4j-admin database check flags / options
#To know more about the flags read here : https://neo4j.com/docs/operations-manual/current/backup-restore/
consistency-checker/
consistencyCheck:
  enable: false
  checkIndexes: true
  checkGraph: true
  checkCounts: true
  checkPropertyOwners: true
  #The database name for which consistency check needs to be done.
  #Defaults to the backup.database values if left empty
  #The database name here should match with one of the database names present in backup.database. If not ,
the consistency check will be ignored
  database: ""
  maxOffHeapMemory: ""
  threads: ""
  verbose: true

# Set to name of an existing Service Account to use if desired
# Follow the following links for setting up a service account with workload identity
# Azure - https://learn.microsoft.com/en-us/azure/aks/workload-identity-overview?tabs=go
# GCP - https://cloud.google.com/kubernetes-engine/docs/how-to/workload-identity
# AWS - https://docs.aws.amazon.com/eks/latest/userguide/associate-service-account-role.html
serviceAccountName: ""

# Volume to use as temporary storage for files before they are uploaded to cloud. For large databases
local storage may not have sufficient space.
# In that case set an ephemeral or persistent volume with sufficient space here
# The chart defaults to an emptyDir, use this to overwrite default behavior
#tempVolume:
#  persistentVolumeClaim:
#    claimName: backup-pvc

# securityContext defines privilege and access control settings for a Pod. Making sure that we don't run
Neo4j as root user.
securityContext:
  runAsNonRoot: true
  runAsUser: 7474
  runAsGroup: 7474
```

```
  fsGroup: 7474
  fsGroupChangePolicy: "Always"

containerSecurityContext:
  runAsNonRoot: true
  runAsUser: 7474
  runAsGroup: 7474
  readOnlyRootFilesystem: false
  allowPrivilegeEscalation: false
  capabilities:
    drop: ["ALL"]
# default ephemeral storage of backup container
resources:
  requests:
    ephemeralStorage: "4Gi"
    cpu: ""
    memory: ""
  limits:
    ephemeralStorage: "5Gi"
    cpu: ""
    memory: ""

# nodeSelector labels
# please ensure the respective labels are present on one of nodes or else helm charts will throw an error
nodeSelector: {}
#  label1: "true"
#  label2: "value1"

# set backup pod affinity
affinity: {}
#  podAffinity:
#    requiredDuringSchedulingIgnoredDuringExecution:
#      - labelSelector:
#          matchExpressions:
#            - key: security
#              operator: In
#              values:
#                - S1
#        topologyKey: topology.kubernetes.io/zone
#  podAntiAffinity:
#    preferredDuringSchedulingIgnoredDuringExecution:
#      - weight: 100
#        podAffinityTerm:
#          labelSelector:
#            matchExpressions:
#              - key: security
#                operator: In
#                values:
#                  - S2
#          topologyKey: topology.kubernetes.io/zone

#Add tolerations to the Neo4j pod
tolerations: []
#  - key: "key1"
#    operator: "Equal"
#    value: "value1"
#    effect: "NoSchedule"
#  - key: "key2"
#    operator: "Equal"
#    value: "value2"
#    effect: "NoSchedule"
```

Back up your database(s)

To back up your database(s), you install the *neo4j-admin* Helm chart using the configured *backup-values.yaml* file.

1. Install *neo4j-admin* Helm chart using the *backup-values.yaml* file:

```
helm install backup-name neo4j/neo4j-admin -f /path/to/your/backup-values.yaml
```

The *neo4j/neo4j-admin* Helm chart installs a cronjob that launches a pod based on the job schedule. This pod performs a backup of one or multiple databases, a consistency check of the backup file(s), and uploads them to the cloud provider bucket.

2. Monitor the backup pod logs using `kubectl logs pod/<neo4j-backup-pod-name>` to check the progress of the backup.
3. Check that the backup files and the consistency check reports have been uploaded to the cloud provider bucket or on-premises storage.

Aggregate a database backup chain

Introduced in 5.21

The aggregate backup command turns a backup chain into a single backup file. This is useful when you have a backup chain that you want to restore to a different cluster, or when you want to archive a backup chain. For more information on the benefits of the aggregate backup chain operation, its syntax and available options, see Aggregate a database backup chain.

Starting from 5.26 LTS, the *neo4j-admin* Helm chart supports an optional temporary directory to be used by the aggregation process instead of the backup working directory. This is especially useful when the size of the backup chain is larger then the pods ephemeral storage. To avoid the backup aggregation job to fail due to lack of disk space, you can set the `tempDir` parameter to a persistent volume claim that has enough space to hold the backup files.

Note:

The *neo4j-admin* Helm chart supports aggregating a backup chain stored in an AWS S3 bucket or a local mount. If enabled, normal backup will not be done, only aggregate backup.

1. To aggregate a backup chain stored in an AWS S3 bucket or a local mount, you need to provide the following information in your *backup-values.yaml* file:

   If your backup chain is stored on AWS S3, you need to set cloudProvider to `aws` and use either `creds` or `serviceAccount` to connect to your AWS S3 bucket. For example:

   Connect to your AWS S3 bucket using the `awscreds` secret

```
neo4j:
  image: "neo4j/helm-charts-backup"
  imageTag: "5.26.25"
  jobSchedule: "* * * * *"
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
  backoffLimit: 3

backup:

  cloudProvider: "aws"
  secretName: "awscreds"
  secretKeyName: "credentials"

  aggregate:
```

```
    enabled: true
    verbose: false
    keepOldBackup: false
    parallelRecovery: false
    fromPath: "s3://bucket1/bucket2"
    # Database name to aggregate. Can contain * and ? for globbing.
    database: "neo4j"
    # Optional temporary directory for aggregation process
    # If not specified, will use the backup directory
    tempDir: "/custom/temp/dir"

resources:
  requests:
    ephemeralStorage: "4Gi"
  limits:
    ephemeralStorage: "5Gi"
```

Connect to your AWS S3 bucket using `serviceAccount`

```
neo4j:
  image: "neo4j/helm-charts-backup"
  imageTag: "5.26.25"
  jobSchedule: "* * * * *"
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
  backoffLimit: 3

backup:

    cloudProvider: "aws"

    aggregate:
      enabled: true
      verbose: false
      keepOldBackup: false
      parallelRecovery: false
      fromPath: "s3://bucket1/bucket2"
      # Database name to aggregate. Can contain * and ? for globbing.
      database: "neo4j"
      # Optional temporary directory for aggregation process
      # If not specified, will use the backup directory
      tempDir: "/custom/temp/dir"

#The service account must already exist in your cloud provider account and have the
necessary permissions to manage your S3 bucket, as well as to download and upload files. See
the example policy below.
#{
#   "Version": "2012-10-17",
#    "Id": "Neo4jBackupAggregatePolicy",
#    "Statement": [
#        {
#            "Sid": "Neo4jBackupAggregateStatement",
#            "Effect": "Allow",
#            "Action": [
#                "s3:ListBucket",
#                "s3:GetObject",
#                "s3:PutObject",
#                "s3:DeleteObject"
#            ],
#            "Resource": [
#                "arn:aws:s3:::mybucket/*",
#                "arn:aws:s3:::mybucket"
#            ]
#        }
#    ]
#}
serviceAccountName: "my-service-account"

resources:
  requests:
    ephemeralStorage: "4Gi"
  limits:
    ephemeralStorage: "5Gi"
```

```
neo4j:
  image: "neo4j/helm-charts-backup"
  imageTag: "5.26.25"
  successfulJobsHistoryLimit: 1
  failedJobsHistoryLimit: 1
  backoffLimit: 1

backup:

  aggregate:
    enabled: true
    verbose: false
    keepOldBackup: false
    parallelRecovery: false
    fromPath: "/backups"
    # Database name to aggregate. Can contain * and ? for globbing.
    database: "neo4j"
    # Optional temporary directory for aggregation process
    # If not specified, will use the backup directory
    tempDir: "/custom/temp/dir"

tempVolume:
  persistentVolumeClaim:
    claimName: aggregate-pv-pvc

resources:
  requests:
    ephemeralStorage: "4Gi"
  limits:
    ephemeralStorage: "5Gi"
```

2. Install the *neo4j-admin* Helm chart using the configured *backup-values.yaml* file:

```
helm install backup-name neo4j/neo4j-admin -f /path/to/your/backup-values.yaml
```

3. Monitor the pod logs using `kubectl logs pod/<neo4j-aggregate-backup-pod-name>` to check the progress of the aggregate backup operation.
4. Verify that the aggregated backup file has replaced your backup chain in the cloud provider bucket or on-premises storage.

Restore a single database

To restore a single offline database or a database backup, you first need to delete the database that you want to replace unless you want to restore the backup as an additional database in your DBMS. Then, use the restore command of `neo4j-admin` to restore the database backup. Finally, use the Cypher command `CREATE DATABASE name` to create the restored database in the `system` database.

Delete the database that you want to replace

Before you restore the database backup, you have to delete the database that you want to replace with that backup using the Cypher command `DROP DATABASE name` against the `system` database. If you want to restore the backup as an additional database in your DBMS, then you can proceed to the next section.

Note:

For Neo4j cluster deployments, you run the Cypher command `DROP DATABASE name` only on one of the cluster servers. The command is automatically routed from there to the other cluster members.

1. Connect to the Neo4j DBMS:

```
kubectl exec -it <release-name>-0 -- bash
```

2. Connect to the `system` database using `cypher-shell`:

```
cypher-shell -u neo4j -p <password> -d system
```

3. Drop the database you want to replace with the backup:

```
DROP DATABASE neo4j;
```

4. Exit the Cypher Shell command-line console:

```
:exit;
```

Restore the database backup

You use the `neo4j-admin database restore` command to restore the database backup, and then the Cypher command `CREATE DATABASE name` to create the restored database in the `system` database. For information about the command syntax, options, and usage, see Restore a database backup.

Note:

For Neo4j cluster deployments, restore the database backup on each cluster server.

1. Run the `neo4j-admin database restore` command to restore the database backup:

```
neo4j-admin database restore neo4j --from-path=/backups/neo4j --expand-commands
```

2. Connect to the `system` database using `cypher-shell`:

```
cypher-shell -u neo4j -p <password> -d system
```

3. Create the `neo4j` database.

Note:

For Neo4j cluster deployments, you run the Cypher command `CREATE DATABASE name` only on one of the cluster servers.

```
CREATE DATABASE neo4j;
```

4. Open the browser at *http://<external-ip>:7474/browser/* and check that all data has been successfully restored.
5. Execute a Cypher command against the `neo4j` database, for example:

```
MATCH (n) RETURN n
```

> **Note:**
>
> If you have backed up your database with the option `--include-metadata`, you can manually restore the users and roles metadata. For more information, see Restore a database backup → Example.

> **Note:**
>
> To restore the `system` database, follow the steps described in Dump and load databases (offline).

Upgrade Neo4j on Kubernetes

Upgrade to a new minor version

To upgrade to the next release of Neo4j, update your Neo4j *values.yaml* file and upgrade the helm release.

1. Open the *values.yaml* file, using the code editor of your choice, and add the following line to the `image` object:

```
image:
  customImage: neo4j:5.26.25
```

2. Run `helm upgrade` to apply the changes:

```
helm upgrade <release-name> neo4j/neo4j -f values.yaml
```

Upgrade from Community to Enterprise

To upgrade from Neo4j Community to Enterprise edition, run:

```
helm upgrade <release-name> neo4j/neo4j --reuse-values --set neo4j.edition=enterprise --set
neo4j.acceptLicenseAgreement=yes
```

Migrate Neo4j from Labs Helm to Neo4j Helm charts

To migrate your Neo4j deployment from the Labs Helm charts to the Neo4j Helm charts offline, back up

your standalone instance or cluster created with the Labs Helm chart and restore it in a standalone instance or a cluster created using the Neo4j Helm chart.

Neo4j supports the following migration paths for a single instance and a cluster:

Single instance

- From the Labs Helm chart 3.5 or earlier to either the Neo4j Helm chart 4.3 or 4.4 — upgrade your Neo4j deployment to whichever version you want to move to, using the steps in the https://neo4j.com/labs/neo4j-helm/1.0.0/ and then migrate from the Labs Helm chart (4.3 or 4.4) to the Neo4j Helm chart 4.3 or 4.4 using the steps described here.
- From the Labs Helm chart 4.3 to the Neo4j Helm chart 4.3 — follow the steps described here.
- From the Labs Helm chart 4.3 to the Neo4j Helm chart 4.4 — follow the steps described here.

Cluster

From the Labs Helm chart 4.3 or 4.4 to the Neo4j Helm chart 4.4 — follow the steps described here.

Back up a Neo4j deployment created with the Labs Helm chart

To back up your Neo4j deployment created with the Labs Helm chart, follow the steps in the Neo4j-Helm User Guide → Backing up Neo4j Containers.

Restore your backup into a standalone or a cluster created with the Neo4j Helm chart

If the backup exists on a cloud provider, you can take one of the following approaches:

Approach 1

1. Create a standalone or a cluster using the Neo4j Helm chart with a custom Neo4j image that has all the cloud provider utilities to download the backup from the respective cloud provider storage to your specific mount.
2. Restore the backup following the steps described in Restore a single database.

Approach 2

1. Get the backup on your local machine.
2. Copy the backup to the respective mount in your new cluster created using the Neo4j Helm chart, using the command `kubectl cp <local-path> <pod>:<path>`. For example,

   ```
   kubectl cp /Users/username/Desktop/backup/4.3.3/neo4j standalone-0:/tmp/
   ```

   where the */tmp* directory refers to the mount.
3. Restore the backup following the steps described in Restore a single database.

Scale a Neo4j deployment

Neo4j supports both vertical and horizontal scaling.

Vertical scaling

To increase or decrease the resources (CPU, memory) available to a Neo4j instance, change the `neo4j.resources` object in the *values.yaml* file to set the desired resource usage, and then perform a helm upgrade.

> **Note:**
>
> If you change the memory allocated to the Neo4j container, you should also change the Neo4j's memory configuration (`server.memory.heap.initial_size` and `server.memory.pagecache.size` in particular). See Important configuration parameters for more details.

For example, if your running Neo4j instance has the following allocated resources:

```
# values.yaml
neo4j:
  resources:
    cpu: "1"
    memory: "3Gi"

# Neo4j Configuration (yaml format)
config:
  server.memory.heap.initial_size: "2G"
  server.memory.heap.max_size: "2G"
  server.memory.pagecache.size: "500m"
```

And, you want to increase them to 2 CPUs and 4 GB of memory (allocating additional memory to the pagecache).

1. Modify the *values.yaml* file to set the desired resource usage:

   ```
   # values.yaml
   neo4j:
     resources:
       cpu: "2"
       memory: "4Gi"

   # Neo4j Configuration (yaml format)
   config:
     server.memory.heap.initial_size: "2G"
     server.memory.heap.max_size: "2G"
     server.memory.pagecache.size: "1G"
   ```

2. Run `helm upgrade` with the modified deployment *values.yaml* file and the Neo4j Helm chart to apply the changes. For example:

   ```
   helm upgrade <release-name> neo4j/neo4j -f values.yaml
   ```

Horizontal scaling

You can add a new server to the Neo4j cluster to scale out read workloads.

The following examples assume that you have a running Neo4j cluster with 3 servers and your Kubernetes cluster has a node available for the new server, `server4`.

Install `server4` using one of the following options:

Add a new server and enable it manually

1. Install `server4` using the same value for `neo4j.name` as your existing cluster:

   ```
   helm install server4 neo4j --set neo4j.edition=enterprise --set neo4j.acceptLicenseAgreement=yes --set
   volumes.data.mode=defaultStorageClass --set neo4j.password="password" --set neo4j.minimumClusterSize=3
   --set neo4j.name=my-cluster
   ```

   Alternatively, you can use a *values.yaml* file to set the values for the new server and the *neo4j/neo4j* Helm chart to install the new server. For more information, see Create Helm deployment values files and Install Neo4j cluster servers.

   When the new server joins the cluster, it will initially be in the `Free` state.

2. Enable `server4` to be able to host databases by using `cypher-shell` (or Neo4j Browser) to connect to one of the existing servers:

   a. Access the cypher-shell on `server1`:

      ```
      kubectl exec -ti server1-0 -- cypher-shell -u neo4j -p password -d neo4j
      ```

   b. When the `cypher-shell` prompt is ready, verify that `server4` is in the `Free` state, and take a note of its name:

      ```
      SHOW SERVERS;
      ```

      ```
      +-----------------------------------------------------------------------------------------------------
      -------------------------------+
      | name                                   | address                                    | state     |
      health      | hosting             |
      +-----------------------------------------------------------------------------------------------------
      -------------------------------+
      | "0908819d-238a-473d-9877-5cc406050ea2" | "server4.neo4j.svc.cluster.local:7687" | "Free"    |
      "Available" | ["system"]          |
      | "19817354-5cd1-4579-8c45-8b897808fdb4" | "server2.neo4j.svc.cluster.local:7687" | "Enabled" |
      "Available" | ["system", "neo4j"] |
      | "b3c91592-1806-41d0-9355-8fc6ba236043" | "server3.neo4j.svc.cluster.local:7687" | "Enabled" |
      "Available" | ["system", "neo4j"] |
      | "eefd7216-6096-46f5-9c41-a74f79684172" | "server1.neo4j.svc.cluster.local:7687" | "Enabled" |
      "Available" | ["system", "neo4j"] |
      +-----------------------------------------------------------------------------------------------------
      -------------------------------+
      ```

3. Using its name, enable `server4` to use it in the cluster:

   ```
   ENABLE SERVER "0908819d-238a-473d-9877-5cc406050ea2";
   ```

4. Run `SHOW SERVERS;` again to verify that `server4` is enabled:

   ```
   SHOW SERVERS;
   ```

```
+-------------------------------------------------------------------------------------------------------------
---------------------------+
| name                                 | address                                 | state     | health
| hosting             |
+-------------------------------------------------------------------------------------------------------------
---------------------------+
| "0908819d-238a-473d-9877-5cc406050ea2" | "server4.neo4j.svc.cluster.local:7687" | "Enabled" |
"Available" | ["system"]          |
| "19817354-5cd1-4579-8c45-8b897808fdb4" | "server2.neo4j.svc.cluster.local:7687" | "Enabled" |
"Available" | ["system", "neo4j"] |
| "b3c91592-1806-41d0-9355-8fc6ba236043" | "server3.neo4j.svc.cluster.local:7687" | "Enabled" |
"Available" | ["system", "neo4j"] |
| "eefd7216-6096-46f5-9c41-a74f79684172" | "server1.neo4j.svc.cluster.local:7687" | "Enabled" |
"Available" | ["system", "neo4j"] |
+-------------------------------------------------------------------------------------------------------------
---------------------------+
```

Add a new server and enable it automatically

This feature is available from Neo4j 5.20.

You can enable a new server automatically when it joins the cluster by setting the `neo4j.operations.enableServer` value to `true` in the *values.yaml* file or using the `--set` flag when running `helm install`. This feature is useful when you want to add a new server to the cluster and enable it without manual intervention.

1. Install `server4` using one of the following options:
   - Run `helm install` with the same value for `neo4j.name` as your existing cluster:

     ```
     helm install server4 neo4j --set neo4j.edition=enterprise --set neo4j.acceptLicenseAgreement=yes
     --set volumes.data.mode=defaultStorageClass --set neo4j.password="password" --set
     neo4j.minimumClusterSize=3 --set neo4j.name=my-cluster --set neo4j.operations.enableServer=true
     --set image="neo4j/helm-charts-operations:5.26.0" --set protocol="neo4j"
     ```

   - Use a *values.yaml* file to set the values for the new server and the *neo4j/neo4j* Helm chart to install the new server.

     a. Ensure that the `neo4j.name` value is the same as your existing cluster and the `neo4j.operations.enableServer` is set to `true`.

     ```yaml
     neo4j:
       name: "my-cluster"
       minimumClusterSize: 3
       resources:
         cpu: "0.5"
         memory: "2Gi"
       password: "my-password"
       edition: "enterprise"
       acceptLicenseAgreement: "yes"

     volumes:
       data:
         mode: "dynamic"
         dynamic:
           # * premium-rwo provisions SSD disks (recommended)
           # * standard-rwo provisions balanced SSD-backed disks
           # * standard provisions HDD disks
           storageClassName: premium-rwo

     operations:
           enableServer: true
           image: "neo4j/helm-charts-operations:5.26.0"
           # protocol can be "neo4j or "neo4j+ssc" or "neo4j+s". Default set to neo4j
     ```

```
    # Note: Do not specify bolt protocol here...it will FAIL.
    protocol: "neo4j"
    labels: {}
```

> **Tip:**
>
> For all possible configuration options and more information on customizing your deployment, see Customizing a Neo4j Helm chart. For more information on how to create a *values.yaml* file and install Neo4j cluster servers, see Create Helm deployment values files and Install Neo4j cluster servers.

b. Run `helm install` with the modified deployment *values.yaml* file and the Neo4j Helm chart to apply the changes.

```
helm install server4 neo4j -f values.yaml
```

When the new server joins the cluster, it will automatically be enabled and ready to host databases.

2. Using Cypher Shell (or Neo4j Browser), verify that the new server is in the state `Enabled`:

a. Access the Cypher Shell on `server1`:

```
kubectl exec -ti server1-0 -- cypher-shell -u neo4j -p password -d neo4j
```

b. Run `SHOW SERVERS;` to verify that `server4` is enabled:

```
SHOW SERVERS;
```

```
+-----------------------------------------------------------------------------------------------------
--------------------------------+
| name                                 | address                                 | state     |
health      | hosting             |
+-----------------------------------------------------------------------------------------------------
--------------------------------+
| "0908819d-238a-473d-9877-5cc406050ea2" | "server4.neo4j.svc.cluster.local:7687" | "Enabled" |
"Available" | ["system"]          |
| "19817354-5cd1-4579-8c45-8b897808fdb4" | "server2.neo4j.svc.cluster.local:7687" | "Enabled" |
"Available" | ["system", "neo4j"] |
| "b3c91592-1806-41d0-9355-8fc6ba236043" | "server3.neo4j.svc.cluster.local:7687" | "Enabled" |
"Available" | ["system", "neo4j"] |
| "eefd7216-6096-46f5-9c41-a74f79684172" | "server1.neo4j.svc.cluster.local:7687" | "Enabled" |
"Available" | ["system", "neo4j"] |
+-----------------------------------------------------------------------------------------------------
--------------------------------+
```

Change the database topology

Notice in the output that although `server4` is now enabled, it is not hosting the `neo4j` database. You need to change the database topology to also use the new server.

1. Alter the `neo4j` database topology to be hosted on three primary and one secondary servers:

```
ALTER DATABASE neo4j SET TOPOLOGY 3 PRIMARIES 1 SECONDARY;
```

2. Now run the `SHOW DATABASES;` command to verify the new topology:

```
SHOW DATABASES;
```

```
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+
| name     | type       | aliases | access       | address                                | role
| writer | requestedStatus | currentStatus | statusMessage | default | home  | constituents |
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+
| "neo4j"  | "standard" | []      | "read-write" | "server2.neo4j.svc.cluster.local:7687" | "primary"
| TRUE   | "online"        | "online"      | ""           | TRUE    | TRUE  | []           |
| "neo4j"  | "standard" | []      | "read-write" | "server4.neo4j.svc.cluster.local:7687" |
"secondary" | FALSE  | "online"        | "online"      | ""           | TRUE    | TRUE  | []
|
| "neo4j"  | "standard" | []      | "read-write" | "server3.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""           | TRUE    | TRUE  | []           |
| "neo4j"  | "standard" | []      | "read-write" | "server1.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""           | TRUE    | TRUE  | []           |
| "system" | "system"   | []      | "read-write" | "server2.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""           | FALSE   | FALSE | []           |
| "system" | "system"   | []      | "read-write" | "server4.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""           | FALSE   | FALSE | []           |
| "system" | "system"   | []      | "read-write" | "server3.neo4j.svc.cluster.local:7687" | "primary"
| TRUE   | "online"        | "online"      | ""           | FALSE   | FALSE | []           |
| "system" | "system"   | []      | "read-write" | "server1.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""           | FALSE   | FALSE | []           |
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+
```

Note that `server4` now hosts the `neo4j` database with the `secondary` role.

Use custom images from private registries

From Neo4j 4.4.4, you can use custom images from private registries by adding new or existing `imagePullSecrets`.

Add an existing `imagePullSecret`

You can use an existing `imagePullSecret` for your Neo4j deployment by specifying its name in the *values.yaml* file. The Neo4j Helm chart checks if the provided `imagePullSecret` exists in the Kubernetes cluster and uses it. If a Secret with the given name does not exist in the cluster, the Neo4j Helm chart throws an error.

Note:

For more information on how to set your Docker credentials in the cluster as a Secret, see the Kubernetes documentation.

Using an already existing Secret **mysecret**

```
# values.yaml
# Override image settings in Neo4j pod
image:
  imagePullPolicy: IfNotPresent
```

```
  # set a customImage if you want to use your own docker image
  customImage: demo_neo4j_image:v1

  #imagePullSecrets list
  imagePullSecrets:
      - "mysecret"
```

Create and add a new `imagePullSecret`

Alternatively, you can create a new `imagePullSecret` for your Neo4j deployment by defining an equivalent `imageCredential` in the *values.yaml* file.

The Neo4j Helm chart creates a Secret with the given name and uses it as an `imagePullSecret` to pull the custom image defined. The following example shows how to define a private docker registry `imageCredential` with the name `mysecret`.

Creating and adding `mysecret` as the `imagePullSecret` to the cluster.

```
# values.yaml
# Override image settings in Neo4j pod
image:
  imagePullPolicy: IfNotPresent
  # set a customImage if you want to use your own docker image
  customImage: custom_neo4j_image:v1

  #imagePullSecrets list
  imagePullSecrets:
      - "mysecret"

  #imageCredentials list for which Secret of type docker-registry will be created automatically using the
details provided
  # password and name are compulsory fields for an imageCredential, without these fields helm chart will
throw an error
  # registry, username, and email are optional fields, but either the username or the email must be
provided
  # imageCredential name should be part of the imagePullSecrets list or else the respective
imageCredential will be ignored and no Secret creation will be done
  # In case of a Secret already pre-existing you don't need to mention the imageCredential, just add the
pre-existing secretName to the imagePullSecret list
  # and that will be used as an imagePullSecret
  imageCredentials:
    - registry: "https://index.docker.io/v1/"
      username: "myusername"
      password: "your_password"
      email: "myusername@example.com"
      name: "mysecret"
```

Assign Neo4j pods to specific nodes

The Neo4j Helm charts `neo4j/neo4j` and `neo4j/neo4j-admin` (from Neo4j 5.10) provide support for assigning your Neo4j pods to specific nodes using `nodeSelector` labels.

You specify the `nodeSelector` labels in the *values.yaml* file.

Note:

If there is no node with the given labels, the Helm chart will throw an error.

nodeSelector labels in values.yaml

```
#nodeSelector labels
```

```
#Ensure the respective labels are present on one of the cluster nodes or else Helm chart will throw an
error.
nodeSelector:
   nodeNumber: one
   name: node1
```

`nodeSelector` *along with the* `--dry-run` *flag*

Note:

When running `helm install --dry-run` or `helm template --dry-run` with `nodeSelector`, you must disable the lookup function of `nodeSelector` by setting `disableLookups: true`. Otherwise, the commands will fail.

You can either add the following to the *values.yaml* file:

```
disableLookups: true
```

or, use `--set disableLookups=true` as part of the command, for example:

```
helm template standalone neo4j --set disableLookups=true .. ... .. --dry-run
```

Troubleshooting

The following information can help you diagnose and correct a problem.

Locate and investigate problems with the Neo4j Helm chart

The rollout of the Neo4j Helm chart in Kubernetes can be thought of in these approximate steps:

1. *Neo4j Pod* is created.
2. Neo4j Pod is scheduled to run on a specific Kubernetes Node.
3. All *Containers* in the Neo4j Pod are created.
4. *InitContainers* in the Neo4j Pod is run.
5. *Containers* in the Neo4j Pod are run.
6. `Startup` and `Readiness` probes are checked.

After all these steps are completed successfully, the Neo4j StatefulSet, Pod, and Services must be in a `ready` state. You should be able to connect to and use your Neo4j database.

If the Neo4j Helm chart is installed successfully, but Neo4j is not starting and reaching a `ready` state in Kubernetes, then troubleshooting has two steps:

1. Check the state of resources in Kubernetes using `kubectl get` commands. This will identify which step has failed.
2. Collect the information relevant to that step.

Depending on the failed step, you can collect information from Kubernetes (e.g., using `kubectl describe`) and from the Neo4j process (e.g., checking the Neo4j debug log).

The following table provides simple steps to get started investigating problems with the Neo4j Helm chart rollout. For more information on how to debug applications in Kubernetes, see the Kubernetes documentation.

Table 12. Investigating problems with the Neo4j Helm chart rollout

Step	Diagnosis	Further investigation
Neo4j Pod created	If `kubectl get pod <release-name>-0` does not return a single Pod result, there is a problem with the pod creation.	Describe the Neo4j StatefulSet — check the output of `kubectl describe statefulset <release-name>`.
Neo4j Pod scheduled	If the state, shown in `kubectl get pod <release-name>-0`, is stuck in `Pending`, there is a problem with pod scheduling.	Describe the Neo4j Pod `kubectl describe pod <release-name>-0` and check the output.
Containers in the Neo4j Pod created	If the state, shown in `kubectl get pod <release-name>-0`, is stuck in `Waiting`, there is a problem with creating or starting containers.	Describe the Neo4j Pod — check the output of `kubectl describe pod <release-name>-0`, paying particular attention to `Events`.
InitContainers in the Neo4j Pod	If the state, shown in `kubectl get pod <release-name>-0`, is stuck in `Init:` (e.g., `Init:CrashLoopBackOff`, `Init:Error` etc.), there is a problem with `InitContainers`. Note that if the pod `Status` is `PodInitializing` or `Running`, then `InitContainers` have already finished successfully.	Describe the Neo4j Pod — check the output of `kubectl describe pod <release-name>-0`, paying particular attention to `InitContainer` (note the `InitContainer` names) and `Events`. Fetch `InitContainer` logs using `kubectl logs <pod-name> -c <init-container-name>`.
Containers in the Neo4j Pod running	If the state, shown in `kubectl get pod <release-name>-0`, does NOT match any of the states listed above, but the Pod still does not reach `Running`, then there is a problem running containers in the Neo4j Pod.	Describe the Neo4j Pod — check the output of `kubectl describe pod <release-name>-0`, paying particular attention to the `Container` state (note the `Container` names) and `Events`. Fetch `Container` logs using `kubectl logs <pod-name> -c <init-container-name>`. If the Neo4j `Container` is starting but exits unexpectedly (e.g., the state is `CrashLoopBackOff`), follow the instructions for Neo4j crashes or restarts unexpectedly.

Startup and Readiness Probes	If the state, shown in `kubectl get pod <release-name>-0`, is `Running`, but the pod does not become `ready`, there is a problem with `Startup` or `Readiness` probes.	Describe the Neo4j Pod — check the output of `kubectl describe pod <release-name>-0`, paying particular attention to `Events` and probes. Check the pod log `kubectl logs <release-name>-0`, the Neo4j log `kubectl exec <release-name>-0 -- tail -n 100 /logs/neo4j.log`, and the Neo4j debug log `kubectl exec <release-name>-0 -- tail -n 500 /logs/debug.log`.

Neo4j crashes or restarts unexpectedly

If the Neo4j Pod starts but then crashes or restarts unexpectedly, there are a range of possible causes. Known causes include:

- An invalid or incorrect configuration of Neo4j, causing it to shut down shortly after the container is started.
- The Neo4j Java process runs out of memory and exits with `OutOfMemoryException`.
- There has been some disruption affecting the Kubernetes Node where the Neo4j Pod is scheduled, e.g., it is being shut drained or has shut down.
- Containers in the Neo4j Pod are shut down by the operating system for using more memory than the resource limit configured for the container (`OOMKilled`).
- Very long Garbage Collection pauses cause the Neo4j Pod `LivenessProbe` to fail, causing Kubernetes to restart Neo4j.

Note:

`OOMKILLED` and `OutOfMemoryException` appear very similar, but they appear in different places and have different fixes. It is important to be aware of this and be sure of what you are dealing with.

Here are some checks to help troubleshoot crashes and unexpected restarts:

Describe the Neo4j Pod

Use `kubectl` to describe the Neo4j Pod:

```
kubectl describe pod <release-name>-0
```

Check the Neo4j Container state

Check the `State` and `Last State` of the container. This shows how the `Last State` of a container that has restarted after being `OOMKilled` appears:

```
$ kubectl describe pod neo4j-0
```

```
State:          Running
  Started:      Mon, 1 Jan 2021 00:02:00 +0000
Last State:     Terminated
  Reason:       OOMKilled
  Exit Code:    137
  Started:      Mon, 1 Jan 2021 00:00:00 +0000
  Finished:     Mon, 1 Jan 2021 00:01:00 +0000
```

Note:

`Exit Code: 137` is indicative of `OOMKilled` if it appears here or in other logs, even if the `"OOMKilled"` string is not present.

Check recent `Events`

The `kubectl describe` output shows older events at the top and more recent events at the bottom. Generally, you can ignore older events.

A `Killing` event that shows that the Neo4j container was killed by the Kubernetes `kubelet`:

```
$ kubectl describe pod neo4j-0
```

```
Events:
Type    Reason       Age      From                 Message
----    ------       ----     ----                 -------
Normal  Scheduled    6m30s    default-scheduler    Successfully assigned default/neo4j-0 to k8s-node-a
...
Normal  Killing        56s    kubelet, k8s-node-a  Killing container with id docker://neo4j-0-neo4j:Need
to kill Pod
```

It is not clear from this event log alone *why* Kubernetes decided that the Neo4j container should be killed.

The next steps in this example could be to check:

- if the container was `OOMKilled`.
- if the container failed `Liveness` or `Startup` probes.
- investigate the node to see if there was some reason why it might kill the container, e.g., `kubectl describe node <k8s node>`.

Check Neo4j logs and metrics

The Neo4j Helm chart configures Neo4j to persist logs and metrics on provided volumes. If no volume is explicitly configured for logs or metrics, they are stored persistently on the Neo4j data volume. This ensures that the logs and metrics outputs from a Neo4j instance that crashes or shuts down unexpectedly are preserved.

Collect data from a running Neo4j Pod

- Download all Neo4j logs from a pod using `kubectl cp` commands:

```
kubectl cp <neo4j-pod-name>:/logs neo4j-logs/
```

- If CSV metrics collection is enabled for Neo4j (the default), download all Neo4j metrics from a pod using:

```
kubectl cp <neo4j-pod-name>:/metrics neo4j-metrics/
```

Collect data from a not running Neo4j Pod

If the Neo4j Pod is not running or is crashing so frequently that `kubectl cp` is not feasible, the Neo4j deployment should be put into offline maintenance mode to collect logs and metrics.

Check container logs

The logs for the main Neo4j DBMS process are persisted to disk and can be accessed as described in Check Neo4j logs and metrics. However, the logs for Neo4j startup and logs for other Containers in the Neo4j Pod are sent to the container's `stdout` and `stderr` streams. These container logs can be viewed using `kubectl logs <pod name> -c <container name>`.

Unfortunately, if the container has restarted following a crash or unexpected shutdown, typically, `kubectl logs` shows the logs for the new container instance (following the restart), and the logs for the previous container instance (the instance that shut down unexpectedly) are not available via `kubectl logs`.

To capture the logs for a crashing container, you can try:

- View the container logs in a log collector/aggregator that is connected to your Kubernetes cluster, e.g., Stackdriver, Cloudwatch Logs, Logstash, etc. If you are using a managed Kubernetes platform, this is usually enabled by default.
- Use `kubectl logs --follow` to stream the logs of a running container until it crashes again.

[1] Not recommended because of inconsistencies in Docker Desktop handling of `hostPath` volumes.

Configuration

The topics described are:

- The *neo4j.conf* file — An introduction to the primary configuration file in Neo4j.
- Command expansion — How to provide an additional capability to configure Neo4j by allowing to specify scripts that set values sourced from external files.
- Default file locations — An overview of where files are stored in the different Neo4j distributions and the necessary file permissions for running Neo4j.
- Ports — An overview of the ports relevant to a Neo4j installation.
- Configure network connectors — How to configure network connectors for Neo4j.
- Set initial password — How to set an initial password.
- Configure Neo4j plugins — How to load plugins to a Neo4j deployment.
- Update dynamic settings — How to configure certain Neo4j parameters while Neo4j is running.
- Configuration settings — A complete reference of all configuration settings.

For a complete reference of Neo4j configuration settings, see All configuration settings.

The neo4j.conf file

The *neo4j.conf* file is the main source of configuration settings in Neo4j and includes the mappings of configuration setting keys to values. The location of the *neo4j.conf* file in the different configurations of Neo4j is listed in Default file locations.

Most of the configuration settings in the *neo4j.conf* file apply directly to Neo4j itself, but there are also other settings related to the Java Runtime (the JVM) on which Neo4j runs. For more information, see the JVM specific configuration settings. Many of the configuration settings are also used by `neo4j` launcher scripts.

`neo4j.conf` conventions

The syntax in the `neo4j.conf` file follows the following conventions:

- The equals sign (`=`) maps configuration setting keys to configuration values.
- Lines that start with the number sign (`#`) are handled as comments.
- Trailing comments are not supported.
- Empty lines are ignored.
- Configuring a setting in *neo4j.conf* overwrites any default values. If you want to amend the default values with custom ones, you must explicitly list the default values along with the new ones.
- The configuration settings are not ordered.
- The configuration settings have strict validation enabled by default. It prevents Neo4j from starting if the *neo4j.conf* file contains *typos*, *incorrect information*, or *duplicates* (except for

`server.jvm.additional`). If you set more than one value for `server.jvm.additional`, each setting value adds another custom JVM argument to the `java` launcher.

To disable the strict validation, set `server.config.strict_validation.enabled=false`.

- By default, the character encoding is assumed to be ISO 8859-1. From Neo4j 4.8 onwards, this can be overridden by setting the environment variable `NEO4J_CONFIG_FILE_CHARSET` to, for example, `utf8`.

Configuration settings

General synopsis

Neo4j configuration settings have the following general synopsis:

`<prefix>.<scope>.<component>….<component>.<name>`

Prefix

Prefixes are reserved for denoting two special cases (most settings do not have a prefix):

- `initial` — Settings that are only used during the initialization but are ignored thereafter. For example, `initial.server.mode_constraint`, `initial.dbms.default_database`, etc.
- `internal` — The prefix replaces the terms `unsupported` and `experimental` used in previous versions. This namespace is dedicated to features that are used internally and may change without notice.

Scope

All configuration settings fall into one of the following scopes that behave differently:

- `db` settings can be varied between each database but must be consistent across all configuration files in a cluster/DBMS.
- `dbms` settings must be consistent across all configuration files in a cluster/DBMS.
- `server` settings apply only to the specific server and can be varied between configuration files across a cluster/DBMS.
- `browser` settings apply only to Neo4j Browser.
- `client` settings apply only to the client.

Note:

In Neo4j 5, the `fabric` scope is no longer available. All configuration settings identified by the `fabric` namespace in the `neo4j.conf` file are moved into the `system` database. The Cypher surface is extended to support the Fabric configuration.

For more information, see Composite databases.

Component

Component namespaces are used to group settings that affect similar systems.

Name

The name of the setting. It may contain a common verb and unit patterns, such as `size`, `enabled`, etc. Words are separated by an underscore.

> **Tip:**
>
> For a complete reference of Neo4j configuration settings, see Configuration settings.

JVM-specific configuration settings

A Java virtual machine (JVM) is a virtual machine that enables a computer to run Java programs and programs written in other languages that are also compiled in Java bytecode. The Java heap is where the objects of a Java program live. Depending on the JVM implementation, the JVM heap size often determines how and for how long time the virtual machine performs garbage collection.

Table 13. JVM-specific settings

Setting	Description
`server.memory.heap.initial_size`	Sets the initial heap size for the JVM. By default, the JVM heap size is calculated based on the available system resources.
`server.memory.heap.max_size`	Sets the maximum size of the heap for the JVM. By default, the maximum JVM heap size is calculated based on the available system resources.
`server.jvm.additional`	Sets additional options for the JVM. The options are set as a string and can vary depending on JVM implementation.

> **Note:**
>
> If you want to have good control of the system behavior, it is recommended to set the heap size parameters to the same value to avoid unwanted full garbage collection pauses.

Neo4j comes with several pre-defined values of the `server.jvm.additional` setting. You can find them in the *neo4j.conf* file. The table below lists them and explains their roles.

Table 14. Pre-defined values of `server.jvm.additional`

Default value	Note
`server.jvm.additional=-XX:+UseG1GC`	G1GC (Garbage-First Garbage Collector) provides a good balance between throughput and tail latency with minimal tuning requirements.
`server.jvm.additional=-XX:-OmitStackTraceInFastThrow`	Ensures that common exceptions always have stack traces, allowing effective debugging regardless of logs rotation frequency.

Default value	Note
`server.jvm.additional=-XX:+AlwaysPreTouch`	Ensures that `initmemory` is not only allocated but also committed to the process before the database starts. It is recommended to reduce the heap memory if this flag degrades performance.
`server.jvm.additional=-XX:+UnlockExperimentalVMOptions`	Trusts that non-static final fields are really final. This allows more optimizations and improves overall performance. Disable this flag if you use embedded mode or have extensions or dependencies that may use reflection or serialization to change the value of final fields.
`server.jvm.additional=-XX:+TrustFinalNonStaticFields`	
`server.jvm.additional=-XX:+DisableExplicitGC`	Disables explicit garbage collection, which is occasionally invoked by the JDK itself.
	Allows Neo4j to use `@Contended` annotation.
`server.jvm.additional=-Djdk.nio.maxCachedBufferSize=1024`	Restricts size of cached JDK buffers to 1 KB.
`server.jvm.additional=-Dio.netty.tryReflectionSetAccessible=true`	Enables Netty to allocate direct buffers more efficiently by allowing reflection-based access to internal JVM APIs. This bypasses the standard JVM cleanup mechanism.
`server.jvm.additional=-Djdk.tls.ephemeralDHKeySize=2048`	Expands Diffie Hellman (DH) key size from default 1024 to 2048 for DH-RSA cipher suites used in server TLS handshakes. This is to protect the server from any potential passive eavesdropping.
`server.jvm.additional=-Djdk.tls.rejectClientInitiatedRenegotiation=true`	Mitigates a DDoS vector.
`server.jvm.additional=-XX:FlightRecorderOptions=stackdepth=256`	Increases the default flight recorder stack sampling depth from 64 to 256, to avoid truncating frames when profiling.
`server.jvm.additional=-XX:+UnlockDiagnosticVMOptions`	Allows profilers to sample between safepoints. Without this, sampling profilers may produce less accurate results.
`server.jvm.additional=-XX:+DebugNonSafepoints`	
`server.jvm.additional=--add-opens=java.base/java.nio=ALL-UNNAMED`	Opens modules for Neo4j to allow internal access.
`server.jvm.additional=--add-opens=java.base/java.io=ALL-UNNAMED`	
`server.jvm.additional=--add-opens=java.base/sun.nio.ch=ALL-UNNAMED`	
Introduced in 5.23 `server.jvm.additional=--enable-native-access=ALL-UNNAMED`	Enables native memory access.
`server.jvm.additional=-Dlog4j2.disable.jmx=true`	Disables logging JMX endpoint.
Introduced in Neo4j 5.26 `server.jvm.additional=-Dlog4j.layout.jsonTemplate.maxStringLength=32768`	Increases the JSON log string maximum length.

List currently active settings

You can use `SHOW SETTINGS` to list the currently active configuration settings and their values.

Example 23. List currently active configuration settings

```
SHOW SETTINGS
YIELD name, value
WHERE name STARTS WITH 'server.default'
RETURN name, value
ORDER BY name
LIMIT 3;
```

```
+--------------------------------------------------+
| name                                | value      |
+--------------------------------------------------+
| "server.default_advertised_address" | "localhost" |
| "server.default_listen_address"     | "localhost" |
+--------------------------------------------------+
```

Tip:

For information about dynamic settings, see Update dynamic settings and Configuration settings reference.

Command expansion

Command expansion provides an additional capability to configure Neo4j by allowing you to specify scripts that set values sourced from external files. This is especially useful for:

- avoiding setting sensitive information, such as usernames, passwords, keys, etc., in the *neo4j.conf* file in plain text.
- handling the configuration settings of instances running in environments where the file system is not accessible.

How it works

The scripts are specified in the *neo4j.conf* file with a `$` prefix and the script to execute within brackets (), i.e., `dbms.setting=$(script_to_execute)`.
The configuration accepts any command that can be executed within a child process by the user who owns and executes the Neo4j server. This also means that, in the case of Neo4j set as a service, the commands are executed within the service.

A generic example would be:

```
neo4j.configuration.example=$(/bin/bash echo "expanded value")
```

By providing such a configuration in the *neo4j.conf* file upon server start with command expansion enabled, Neo4j evaluates the script and retrieves the value of the configuration settings prior to the instantiation of Neo4j. The values are then passed to the starting Neo4j instance and kept in memory, in

the running instance.

> **Note:**
>
> You can also use the `curl` (https://curl.se/docs/manpage.html) command to fetch a token or value for a configuration setting. For example, you can apply an extra level of security by replacing any sensitive information in your *neo4j.conf* file with a secured reference to a provider of some sort.

Scripts are run by the Neo4j process and are expected to exit with code `0` within a reasonable time. The script output should be of a valid type for the setting. Failure to do so prevents Neo4j from starting.

> **Note:**
>
> Scripts and their syntax differ between operating systems.

Enabling

To enable command expansion, you must add the `--expand-commands` argument to the Neo4j startup script or *neo4j.service* file.

Starting Neo4j with command expansion

To start Neo4j with command expansion enabled, you can use the following command:

```
bin/neo4j start --expand-commands
```

Enabling command expansion in Neo4j as a service

If you are using Neo4j as a service, you can enable command expansion by adding the `--expand-commands` argument to the */etc/systemd/system/neo4j.service* file. Otherwise, the commands in the configuration file are treated as invalid settings.

You must also add `Type=forking` under the `[Service]` section of */etc/systemd/system/neo4j.service* to allow for the command expansion.

```
[Service]
Type=forking
```

Security checks

Neo4j performs the following basic security checks on the *neo4j.conf* file. If they fail, Neo4j does not evaluate the script commands in *neo4j.conf*, and the Neo4j process does not start.

On Unix (both Linux and Mac OS)

- The *neo4j.conf* and *neo4j-admin.conf* files must, at most, be readable or writable by their owner

and readable by the user-group to which the owner belongs. The *neo4j-admin.conf* file is a configuration file located in the same directory as the *neo4j.conf* file. You can use the *neo4j-admin.conf* file to provide administration-task-specific settings.

- The Neo4j process must run as a user who is either the owner of the *neo4j.conf* file or in the user-group which owns the *neo4j.conf* file.

Note:

The Linux permissions bitmask for the least restrictive permissions is `640`. More restrictive Linux permissions are also allowed. For example, the *neo4j.conf* file can have no group permissions and only be readable by its owner (`400` bitmask).

On Windows

- The *neo4j.conf* and *neo4j-admin.conf* files must, at most, be readable/modifiable but not executable by the owner only.

Note:

The owner may have the following permissions from the Access Control List (ACL):

- `READ_DATA`
- `WRITE_DATA`
- `APPEND_DATA`
- `READ_ATTRIBUTES`
- `WRITE_ATTRIBUTES`
- `READ_NAMED_ATTRS`
- `WRITE_NAMED_ATTRS`
- `READ_ACL`
- `WRITE_ACL`
- `DELETE`
- `DELETE_CHILD`
- `WRITE_OWNER`
- `SYNCHRONIZE`

Logging

The execution of scripts is logged in *neo4j.log*. For each setting that requires the execution of an external command, Neo4j adds an entry into the log file that contains information, for example:

```
… Executing the external script to retrieve the value of <setting>...
```

Error Handling

The scripts' execution may generate two types of errors:

- Errors during the execution — These errors are reported in the *debug.log*, with a code returned from the external execution. In this case, the execution stops and the server does not start.
- Errors for incorrect values — The returned value is not the one expected for the setting. In this case, the server does not start.

For more information, see Exit codes.

Default file locations

Neo4j directories

The page describes the Neo4j directories, specifying their default locations per distribution and minimal file permissions.

If Neo4j was installed using a `tar.gz` or `zip` archive, *<NEO4J_HOME>* refers to the location the archive was extracted to.

Instructions provided for Neo4j Desktop are applicable across all operating systems where Neo4j Desktop is supported.

> **Note:**
>
> If *tmp* is set to `noexec`, it is recommended to set `server.jvm.additional=-Djava.io.tmpdir=/home/neo4j` in *conf/neo4j.conf* and replace */home/neo4j* with a path that has `exec` permissions.
>
> For */bin/cypher-shell*, set this via an environment variable: `export JAVA_OPTS=-Djava.io.tmpdir=/home/neo4j` and replace `/home/neo4j` with a path that has `exec` permissions.
>
> For the Neo4j's uses of the Java Native Access (JNA) library, set `server.jvm.additional=-Djna.tmpdir=/tmp` in *conf/neo4j.conf* and replace `/tmp` with a path that has `exec` permissions.

Bin

The *bin* directory contains the Neo4j running script and built-in tools, such as Cypher Shell and Neo4j Admin.

File permissions

Read only and execute.

Table 15. bin directory default location per distribution

Neo4j distribution	Default file location
Linux / macOS / Docker	*<NEO4J_HOME>/bin*
Windows	*<NEO4J_HOME>\bin*
Debian / RPM	*/usr/bin*
Neo4j Desktop	From the *Open* dropdown menu of your active Neo4j DBMS, select *Terminal* and run `cd bin`.

Certificates

The certificate directory contains the Neo4j TLS certificates.

File permissions

Read only.

Table 16. certificates directory default location per distribution

Neo4j distribution	Default file location
Linux / macOS / Docker	*<NEO4J_HOME>/certificates*
Windows	*<NEO4J_HOME>\certificates*
Debian / RPM	*/var/lib/neo4j/certificates*
Neo4j Desktop	From the *Open* dropdown menu of your active Neo4j DBMS, select *Terminal* and run `cd certificates`.

Configuration

The *configuration* directory contains the Neo4j configuration settings, Log4j configuration settings, and the JMX access credentials. For details about *neo4j.conf*, see The neo4j.conf file.

File permissions

Read only

Table 17. configuration directory default location per distribution

Neo4j distribution	Default file location
Linux / macOS / Docker	*<NEO4J_HOME>/conf/neo4j.conf* *<NEO4J_HOME>/conf/neo4j-admin.conf* *<NEO4J_HOME>/conf/server-logs.xml* *<NEO4J_HOME>/conf/user-log.xml*
Windows	*<NEO4J_HOME>\conf\neo4j.conf* *<NEO4J_HOME>\conf\neo4j-admin.conf* *<NEO4J_HOME>\conf\server-logs.xml* *<NEO4J_HOME>\conf\user-log.xml*

Neo4j distribution	Default file location
Debian / RPM	*/etc/neo4j/neo4j.conf* */etc/neo4j/neo4j-admin.conf* */etc/neo4j/server-logs.xml* */etc/neo4j/user-log.xml*
Neo4j Desktop	From the *Open* dropdown menu of your active Neo4j DBMS, select *Terminal* and run `cd conf`.

Data

The *data* directory contains all data-related content, such as databases, transactions, cluster-state (if applicable), dumps, and the *cypher.script* files (from the `neo4j-admin database restore` command). The data directory is internal to Neo4j and its structure is subject to change between versions without notice.

File permissions

Read and write.

Table 18. data directory default location per distribution

Neo4j distribution	Default file location
Linux / macOS / Docker	*<NEO4J_HOME>/data*
Windows	*<NEO4J_HOME>\data*
Debian / RPM	*/var/lib/neo4j/data*
Neo4j Desktop	From the *Open* dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd data`.

Import

The *import* directory contains all CSV files that the command `LOAD CSV` uses as sources to import data in Neo4j.

File permissions

Read only

Table 19. import directory default location per distribution

Neo4j distribution	Default file location
Linux / macOS / Docker	*<NEO4J_HOME>/import*
Windows	*<NEO4J_HOME>\import*
Debian / RPM	*/var/lib/neo4j/import*
Neo4j Desktop	From the *Open* dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd import`.

Labs

The *labs* directory contains APOC Core. For more information, see APOC User Guide → Installation.

File permissions

Read only.

Table 20. labs directory default location per distribution

Neo4j distribution	Default file location
Linux / macOS / Docker	*<NEO4J_HOME>/labs*
Windows	*<NEO4J_HOME>\labs*
Debian / RPM	*/var/lib/neo4j/labs*
Neo4j Desktop	From the *Open* dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd labs`.

Lib

The *lib* directory contains all Neo4j dependencies.

File permissions

Read only.

Table 21. lib directory default location per distribution

Neo4j distribution	Default file location
Linux / macOS / Docker	*<NEO4J_HOME>/lib*
Windows	*<NEO4J_HOME>\lib*
Debian / RPM	*/usr/share/neo4j/lib*
Neo4j Desktop	From the *Open* dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd lib`.

Important:

In Neo4j 5.26.22, the position of Browser is changed from the directory *lib* to *web* and from .jar to .zip. These changes are reverted in the Enterprise Edition 5.26.23. See Web for details.

Licenses

The *licenses* directory contains Neo4j license files.

File permissions

Read only.

Table 22. licenses directory default location per distribution

Neo4j distribution	Default file location
Linux / macOS / Docker	*<NEO4J_HOME>/licenses*
Windows	*<NEO4J_HOME>\licenses*
Debian / RPM	*/var/lib/neo4j/licenses*
Neo4j Desktop	From the *Open* dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd licences`.

Logs

The *logs* directory contains the Neo4j log files.

File permissions

Read and write.

Table 23. logs directory default location per distribution

Neo4j distribution	Default file location
Linux / macOS / Docker	*<NEO4J_HOME>/logs* [1]
Windows	*<NEO4J_HOME>\logs*
Debian / RPM	*/var/log/neo4j/* [2]
Neo4j Desktop	From the *Open* dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd logs`.

Metrics

The *metrics* directory contains the Neo4j built-in metrics for monitoring the Neo4j DBMS and each individual database.

File permissions

Read and write.

Table 24. metrics directory default location per distribution

Neo4j distribution	Default file location
Linux / macOS / Docker	*<NEO4J_HOME>/metrics*
Windows	*<NEO4J_HOME>\metrics*
Debian / RPM	*/var/lib/neo4j/metrics*
Neo4j Desktop	From the *Open* dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd metrics`.

Plugins

The *plugins* directory contains custom code that extends Neo4j, for example, user-defined procedures,

functions, and security plugins.

File permissions

Read only.

Table 25. plugins directory default location per distribution

Neo4j distribution	Default file location
Linux / macOS / Docker	*<NEO4J_HOME>/plugins*
Windows	*<NEO4J_HOME>\plugins*
Debian / RPM	*/var/lib/neo4j/plugins*
Neo4j Desktop	From the *Open* dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd plugins`.

Products

The *products* directory contains the JAR files of the Neo4j products.

For Enterprise Edition, these are:

- Neo4j Bloom
- Graph Data Science Library
- GenAI plugin
- Neo4j Ops Manager Server
- *README.txt* file — with information on enabling them.

Community Edition contains the GenAI plugin. Starting from 5.26.10, the Fleet management jar file is bundled with the Neo4j CE.

Note:

Using the Fleet management plugin in Neo4j CE deployments is subject to some limitations due to the limited set of features, such as no metrics being available. Also, it is offered on a best-effort basis. Neo4j does not offer any technical support for it.

File permissions

Read only.

Table 26. products directory default location per distribution

Neo4j distribution	Default file location
Linux / macOS / Docker	*<NEO4J_HOME>/products*
Windows	*<NEO4J_HOME>\products*
Debian / RPM	*/var/lib/neo4j/products*

Neo4j distribution	Default file location
Neo4j Desktop	From the *Open* dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd products`.

Run

The *run* directory contains the processes IDs.

File permissions

Read and write.

Table 27. run directory default location per distribution

Neo4j distribution	Default file location
Linux / macOS / Docker	*<NEO4J_HOME>/run*
Windows	*<NEO4J_HOME>\run*
Debian / RPM	*/var/lib/neo4j/run*
Neo4j Desktop	From the *Open* dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd run`.

Web

Introduced in 5.26.22

In Neo4j 5.26.22, a new directory *web* is introduced. The *web* directory contains Neo4j Browser bundled as a .zip file.

Starting from Neo4j 5.26.23, the *web* directory is included only in the Neo4j Community Edition. The Neo4j Enterprise Edition does not include the *web* directory; and Neo4j Browser continues to be distributed as a .jar file located in the *lib* directory.

File permissions

Read only.

Table 28. web directory default location per distribution

Neo4j distribution	Default file location
Linux / macOS / Docker	*<NEO4J_HOME>/web*
Windows	*<NEO4J_HOME>\web*
Debian / RPM	*/var/lib/neo4j/web*
Neo4j Desktop	From the *Open* dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd web`.

Customize your file locations

The file locations can also be customized by using environment variables and options.

The locations of *<NEO4J_HOME>* and *conf* can be configured using environment variables:

Table 29. Configuration of <NEO4J_HOME> and conf

Location	Default	Environment variable	Notes
<NEO4J_HOME>	parent of *bin*	`NEO4J_HOME`	Must be set explicitly if *bin* is not a subdirectory.
conf	*<NEO4J_HOME>/conf*	`NEO4J_CONF`	Must be set explicitly if it is not a subdirectory of *<NEO4J_HOME>*.

The rest of the locations can be configured by uncommenting the respective setting in the *conf/neo4j.conf* file and changing the default value.

```
#server.directories.data=data
#server.directories.plugins=plugins
#server.directories.logs=logs
#server.directories.lib=lib
#server.directories.run=run
#server.directories.licenses=licenses
#server.directories.metrics=metrics
#server.directories.transaction.logs.root=data/transactions
#server.directories.dumps.root=data/dumps
#server.directories.import=import
```

Security considerations

If a directory stores sensitive data, it must never be world-readable or world-executable. Even if files inside are restricted, a world-executable directory can leak structure.

- On Linux, follow the recommendations:

Scenario	Recommended mode
Only owner can access	`700`
Owner and specific group may have access	`750`

  Ensure the process creating the directory uses a restrictive umask (e.g., `077`) to prevent overly permissive defaults.

- On Windows, allow access only to the owning account (or service account) and `Administrators`.
  Remove access for `Users`, `Authenticated Users`, and `Everyone`.
  Before removing broad permissions disable inheritance on the directory, otherwise they may be reintroduced from the parent directory.

Consider security recommendations above when creating a directory for backup files.

Ports

An overview of the Neo4j-specific ports. Note that these ports are in addition to those necessary for ordinary network operation.

Specific recommendations on port openings cannot be made, as the firewall configuration must be performed according to your particular conditions.

> **Note:**
>
> When exposing network services, make sure they are always protected.

Listen address configuration settings

The listen address configuration settings will set the network interface and port to listen on. For example, the IP-address `127.0.0.1` and port `7687` can be set with the value `127.0.0.1:7687`. The table below shows an overview of available Neo4j-specific ports and related configuration settings.

Table 30. Listen address configuration settings overview

Name	Default port	Related configuration setting
Backup	`6362`	`server.backup.listen_address`
HTTP	`7474`	`server.http.listen_address`
HTTPS	`7473`	`server.https.listen_address`
Bolt	`7687`	`server.bolt.listen_address`
Cluster discovery v1	`5000`	`server.discovery.listen_address` Deprecated in 5.23
Cluster internal traffic	`6000`	`server.cluster.listen_address`
Cluster RAFT	`7000`	`server.cluster.raft.listen_address`
Cluster routing connector	`7688`	`server.routing.listen_address`
Graphite monitoring	`2003`	`server.metrics.graphite.server`
Prometheus monitoring	`2004`	`server.metrics.prometheus.endpoint`
JMX monitoring	`3637`	`server.jvm.additional=-Dcom.sun.management.jmxremote.port=3637`
Remote debugging	`5005`	`server.jvm.additional=-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=*:5005`

> The configuration setting `server.default_listen_address` configures the default network interface to listen for incoming connections.

Advertised address configuration settings

The advertised address configuration settings are used for routing purposes. An advertised address is composed of a hostname/IP-address and port. For example, the IP-address `127.0.0.1` and port `7687` can be set with the value `127.0.0.1:7687`. If a host name resolution service has been configured, the advertised address can use a hostname, for example, `example.com:7687`. The table below shows an overview of available Neo4j-specific ports and related configuration settings.

Table 31. Advertised address configuration settings overview

Name	Default port	Related configuration setting
HTTP	`7474`	`server.http.advertised_address`
HTTPS	`7473`	`server.https.advertised_address`
Bolt	`7687`	`server.bolt.advertised_address`
Cluster discovery v1	`5000`	`server.discovery.advertised_address` Deprecated in 5.23
Cluster internal traffic	`6000`	`server.cluster.advertised_address` is used for the discovery service v2 since Neo4j 5.23. See Cluster server discovery for more details.
Cluster RAFT	`7000`	`server.cluster.raft.advertised_address`
Cluster routing connector	`7688`	`server.routing.advertised_address`

> **Note:**
>
> The configuration setting `server.default_advertised_address` configures the default hostname/IP-address for advertised address.

Ports used by Neo4j

Backup

Default port: `6362`

Table 32. Backup

Related configuration setting	Default value	Description
`server.backup.listen_address`	`127.0.0.1:6362`	Network interface and port for the backup server to listen on.
`server.backup.enabled`	`true`	Enable support for running online backups.

In production environments, external access to the backup port should be blocked by a firewall.

For more information, see Backup and restore → Server configuration.

HTTP

Default port: `7474`

Table 33. HTTP connector

Related configuration setting	Default value	Description
`server.http.listen_address`	`:7474`	Network interface and port for the HTTP connector to listen on.
`server.http.advertised_address`	`:7474`	Advertised hostname/IP-address and port for the HTTP connector.
`server.http.enabled`	`true`	Enable the HTTP connector.

- The HTTP connector is enabled by default.
- The network communication is unencrypted.
- Used by Neo4j Browser and the HTTP API.

For more information, see Configure network connectors.

HTTPS

Default port: `7473`

Table 34. HTTPS connector

Related configuration setting	Default value	Description
`server.https.listen_address`	`:7473`	Network interface and port for the HTTPS connector to listen on.
`server.https.advertised_address`	`:7473`	Advertised hostname/IP-address and port for the HTTPS connector.
`server.https.enabled`	`false`	Enable the HTTPS connector.

- The network communication is encrypted.
- Used by Neo4j Browser and the HTTP API.

For more information, see Configure network connectors.

Bolt

Default port: `7687`

Table 35. Bolt connector

Related configuration setting	Default value	Description
`server.bolt.listen_address`	`:7687`	Network interface and port for the Bolt connector to listen on.
`server.bolt.advertised_address`	`:7687`	Advertised hostname/IP-address and port for the Bolt connector.
`server.bolt.enabled`	`true`	Enable the Bolt connector.
`server.bolt.tls_level`	`DISABLED`	Encryption level for the Bolt connector.

- By default, the Bolt connector is **enabled**, but its encryption is **turned off**.
- Used by Cypher Shell, Neo4j Browser, and the official Neo4j drivers.

For more information, see Configure network connectors.

Cluster

All instances of Neo4j Enterprise will open these ports, whether or not they are currently in a multi-process configuration.

Table 36. Cluster listen address

Name	Default port	Default value	Related configuration setting
Discovery v1	`5000`	`:5000`	`server.discovery.listen_address` Deprecated in 5.23
Internal traffic	`6000`	`:6000`	`server.cluster.listen_address` See Cluster server discovery.
RAFT	`7000`	`:7000`	`server.cluster.raft.listen_address`
Routing connector	`7688`	`:7688`	`server.routing.listen_address`

Table 37. Cluster advertised address

Name	Default port	Default value	Related configuration setting
Discovery v1	`5000`	`:5000`	`server.discovery.advertised_address` Deprecated in 5.23

Name	Default port	Default value	Related configuration setting
Internal traffic	`6000`	`:6000`	`server.cluster.advertised_address` is used for the discovery service v2 since Neo4j 5.23. See Cluster server discovery.
RAFT	`7000`	`:7000`	`server.cluster.raft.advertised_address`
Routing connector	`7688`	`:7688`	`server.routing.advertised_address`

The ports are likely be different in a production installation; therefore the potential opening of ports must be modified accordingly.

For more information, see:

- Deploy a basic cluster
- Settings reference

Graphite monitoring

Default port: `2003`

Table 38. Graphite

Related configuration setting	Default value	Description
`server.metrics.graphite.server`	`:2003`	Hostname/IP-address and port of the Graphite server.
`server.metrics.graphite.enabled`	`false`	Enable exporting metrics to the Graphite server.

This is an outbound connection that enables a Neo4j instance to communicate with a Graphite server.

For further information, see Expose metrics → Graphite and the Graphite official documentation.

Prometheus monitoring

Default port: `2004`

Table 39. Prometheus

Related configuration setting	Default value	Description
`server.metrics.prometheus.endpoint`	`localhost:2004`	Network interface and port for the Prometheus endpoint to listen on.

Related configuration setting	Default value	Description
`server.metrics.prometheus.enabled`	`false`	Enable exporting metrics with the Prometheus endpoint.

For more information, see Prometheus.

JMX monitoring

Default port: `3637`

Table 40. Java Management Extensions

Related configuration setting	Default value	Description
`server.jvm.additional=-Dcom.sun.management.jmxremote.port=3637`	`3637`	Additional setting for exposing the Java Management Extensions (JMX).

For further information, see the official documentation on Monitoring and Management Using JMX.

Remote debugging

Default port: `5005`

Table 41. Remote debugging

Related configuration setting	Default value	Description
`server.jvm.additional=-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=*:5005`	`:5005`	Additional setting for exposing remote debugging.

For more information, see the Java Reference → Setup for remote debugging.

Configure network connectors

Neo4j provides support for Bolt, HTTP, and HTTPS protocols using network connectors. Network connectors are configured in the *neo4j.conf* file.

Available network connectors

The table below lists the available network connectors in Neo4j:

Table 42. Neo4j network connectors and port number

Network connector name	Protocol	Default port number
`server.bolt`	Bolt	`7687`
`server.http`	HTTP	`7474`

Network connector name	Protocol	Default port number
`server.https`	HTTPS	`7473`

When configuring the HTTPS or Bolt connectors, see also SSL framework for details on how to work with SSL certificates.

Configuration options

The network connectors are configured by settings in the format `server.<network-connector-name>.<setting-suffix>`.

Table 43. Configuration option suffixes for network connectors

Option name	Default	Setting(s)	Description
`enabled`	`true` [3]	`server.bolt.enabled`, `server.http.enabled`, `server.https.enabled` [4]	This setting allows the client connector to be enabled or disabled. When disabled, Neo4j does not listen for incoming connections on the relevant port.
`listen_address`	`localhost:<network-connector-default-port>`	`server.bolt.listen_address`, `server.https.listen_address`, `server.http.listen_address`	This setting specifies how Neo4j listens for incoming connections. It consists of two parts; an IP address (e.g. 127.0.0.1 or 0.0.0.0) and a port number (e.g. 7687), and is expressed in the format `<ip-address>:<port-number>`. See below for an example of usage.
`advertised_address`	`localhost:<network-connector-default-port>`	`server.bolt.advertised_address`, `server.https.advertised_address`, `server.http.advertised_address`	This setting specifies the address that clients should use for this network connector. This is useful in a cluster as it allows each server to correctly advertise the addresses of the other servers in the cluster. The advertised address consists of two parts; an address (fully qualified domain name, hostname, or IP address) and a port number (e.g. 7687), and is expressed in the format `<address>:<port-number>`. See below for an example of usage.

Option name	Default	Setting(s)	Description
`tls_level`	`DISABLED`	`server.bolt.tls_level`	This setting is only applicable to the Bolt connector. It allows the Bolt connector to accept encrypted and/or unencrypted connections. The default value is `DISABLED`, where only unencrypted client connections are to be accepted by this connector, and all encrypted connections will be rejected. Other values are `REQUIRED` and `OPTIONAL`. Use `REQUIRED` when only encrypted client connections are to be accepted by this connector, and all unencrypted connections will be rejected. Use `OPTIONAL` where either encrypted or unencrypted client connections are accepted by this connector.

Example 24. Specify `listen_address` for the Bolt connector

To listen for Bolt connections on all network interfaces (0.0.0.0) and on port 7000, set the `listen_address` for the Bolt connector:

```
server.bolt.listen_address=0.0.0.0:7000
```

Example 25. Specify `advertised_address` for the Bolt connector

If routing traffic via a proxy, or if port mappings are in use, it is possible to specify `advertised_address` for each network connector individually. For example, if port `7687` on the Neo4j server is mapped from port `9000` on the external network, specify the `advertised_address` for the Bolt connector:

```
server.bolt.advertised_address=<server-name>:9000
```

Options for Bolt thread pooling

See Bolt thread pool configuration to learn more about Bolt thread pooling and how to configure it on the network connector level.

Defaults for addresses

It is possible to specify defaults for the configuration options with `listen_address` and `advertised_address` suffixes. Setting a default value applies to all network connectors unless specifically configured for a

certain connector.

`server.default_listen_address`

This configuration option defines a default IP address of the settings with the `listen_address` suffix for all network connectors. If the IP address part of the `listen_address` is not specified, it is inherited from the shared setting `server.default_listen_address`.

Example 26. Specify `listen_address` for the Bolt connector

To listen for Bolt connections on all network interfaces (0.0.0.0) and on port 7000, set the `listen_address` for the Bolt connector:

```
server.bolt.listen_address=0.0.0.0:7000
```

This is equivalent to specifying the IP address by using the `server.default_listen_address` setting, and then specifying the port number for the Bolt connector.

```
server.default_listen_address=0.0.0.0
server.bolt.listen_address=:7000
```

`server.default_advertised_address`

This configuration option defines a default address of the settings with the `advertised_address` suffix for all network connectors. If the address part of the `advertised_address` is not specified, it is inherited from the shared setting `server.default_advertised_address`.

Example 27. Specify `advertised_address` for the Bolt connector

Specify the address that clients should use for the Bolt connector:

```
server.bolt.advertised_address=server1:9000
```

This is equivalent to specifying the address by using the `server.default_advertised_address` setting, and then specifying the port number for the Bolt connector.

```
server.default_advertised_address=server1
server.bolt.advertised_address=:9000
```

Warning:

The default address settings can only accept the hostname or IP address portion of the full socket address. Port numbers are protocol-specific, and can only be added by the protocol-specific network connector configuration.

For example, if you configure the default address value to be `example.com:9999`, Neo4j will

fail to start and you will get an error in neo4j.log.

Set an initial password

You can use the `set-initial-password` command of `neo4j-admin` to define the password for the native user `neo4j`. If not set explicitly, the password defaults to `neo4j` and must be changed at first login. This command is intended to be used only once, before the first startup of the database.

If the password is not set explicitly using this method, it will be set to the default password `neo4j`. In that case, you will be prompted to change the default password at first login.

Note:

The default minimum password length is 8 characters. To change it, use the `dbms.security.auth_minimum_password_length` configuration setting.

Syntax

The `neo4j-admin dbms set-initial-password` has the following syntax:

```
neo4j-admin dbms set-initial-password [-h] [--expand-commands] [--verbose] [--require-password-
change[=true|false]]
                                     [--additional-config=<file>] <password>
```

Parameters

Table 44. `neo4j-admin dbms set-initial-password` *parameters*

Parameter	Description
`<password>`	

Options

The `neo4j-admin dbms set-initial-password` command has the following options:

Table 45. `neo4j-admin dbms set-initial-password` *options*

Option	Description	Default
`--additional-config=<file>`[5]	Configuration file with additional configuration.	
`--expand-commands`	Allow command expansion in config value evaluation.	
`-h, --help`	Show this help message and exit.	
`--require-password-change[=true|false]`	Require the user to change their password on first login.	`false`
`--verbose`	Enable verbose output.	

Example

You can set the password for the native `neo4j` user before starting the database for the first time.
For example, to set the password to `mySecretPassword`, run the following command in your terminal:

```
neo4j-admin dbms set-initial-password mySecretPassword
```

To enforce a password change at first login, use the `--require-password-change` flag. This is particularly useful when setting a temporary password during initial setup.

Caution:

For security reasons, it is not recommended to pass passwords as command-line arguments to avoid them being stored in the command-line history. One way to achieve this is by using Bash scripting (e.g., `whiptail` or `dialog`) to prompt for the password securely:

Prompt for password using `whiptail`

```
neo4j-admin dbms set-initial-password "$(whiptail --passwordbox "Enter password" 10 50 2>&1
>/dev/tty)"
```

Prompt for password using `dialog`

```
neo4j-admin dbms set-initial-password "$(dialog --passwordbox "Enter password" 10 50 2>&1
>/dev/tty)"
```

Configure plugins

Overview

Plugins are Java Archive (*.jar*) files that extend the functionality of Neo4j by adding new features and capabilities, such as graph algorithms, data integration, visualization, and monitoring tools.

Both Neo4j Community Edition (CE) and Enterprise Edition (EE) come with a range of pre-installed products, such as Gen AI, Graph Data Science, and Fleet management in the *products* directory and the APOC Core jar file in the *labs* directory.
The Fleet management jar file is bundled with the Neo4j server starting with Neo4j 5.26.10 and 2025.07.0 onwards. If you are using an earlier version of Neo4j, you can get the matching version of the Fleet management plugin by following the download link that matches your server version in the **Monitor Deployment** wizard in the Aura console. See Aura documentation → Fleet Management → Add a deployment for more information.

Note:

Using the Fleet management plugin in Neo4j CE deployments is subject to some limitations due to the limited set of features, such as no metrics being available. Also, it is offered on a best-effort basis. Neo4j does not offer any technical support for it.

If your Neo4j CE installation does not include the bundled GDS plugin, check the Neo4j Graph Data Science Library Manual → Supported Neo4j versions to find your matching GDS library version and download it separately. For detailed installation, refer to the corresponding Neo4j GDS manual.

Some of these plugins, such as Bloom and GDS Enterprise require a license activation key. Reach out to your Neo4j account representative or request a representative to contact you.

If you want to use your own plugins or any of the other supported plugins, such as APOC Extended and Neo4jsemantics, ensure that you download and add them to the *plugins* directory. See their respective documentation for more information.

Supported plugins and documentation

The following plugins are supported:

Table 46. Supported Neo4j plugins and documentation

Name	Key	License activation key required	Documentation
APOC Core	`apoc`		APOC Core documentation
APOC Extended	`apoc-extended`		APOC Extended user guide (Labs project)
Bloom	`bloom`	✔ [6]	Neo4j Bloom
Fleet management	`fleet-management`		Fleet management
GenAI	`genai`		GenAI plugin.
Graph Data Science	`graph-data-science`	✔ [7]	Graph Data Science
Neosemantics	`n10s`		Neosemantics

For more information on using plugins in a different Neo4j setup, see:

- Neo4j Desktop → Install a plugin
- Docker → Plugins
- Kubernetes → Plugins
- Java-Reference → Setting up a plugin project.

Install and configure plugins

To install and configure plugins in a Neo4j deployment, follow these steps:

1. Move or copy the plugins (*.jar files*) from *<NEO4J_HOME>/products* and *<NEO4J_HOME>/labs* to the *<NEO4J_HOME>/plugins* directory. See Default file locations for more information.

   Note:

Some plugins are not bundled with Neo4j and need to be downloaded separately, such as APOC Extended and Neosemantics. See Supported plugins and documentation for more information and links to the respective documentation.

2. Configure the plugins that you want to enable by modifying the following setting to the *neo4j.conf* file:
   - Add the plugin that you want to enable to the `dbms.security.procedures.unrestricted` setting.
   - If the plugin requires a license key, add the path to the license key file as well.
   - If `dbms.security.procedures.allowlist` is set in your configuration, also add the plugin to this setting, otherwise no change is needed. By default, all procedures are loaded. For more information, see Securing extensions.

```
# to enable APOC core:
# * dbms.security.procedures.unrestricted=apoc.*
# * dbms.security.procedures.allowlist=apoc.*

# to enable APOC Extended:
# * dbms.security.procedures.unrestricted=apoc.*,apoc.extended.*
# * dbms.security.procedures.allowlist=apoc.*,apoc.extended.*

#  to enable Bloom:
# * dbms.security.procedures.unrestricted=bloom.*
# * dbms.security.procedures.allowlist=bloom.*
# * dbms.bloom.license_file=/path/to/my/license/keyfile

# to enable Fleet management:
# * dbms.security.procedures.unrestricted=fleetManagement.*
# * dbms.security.procedures.allowlist=fleetManagement.*

# to enable GDS:
# * dbms.security.procedures.unrestricted=gds.*
# * dbms.security.procedures.allowlist=gds.*
# * gds.enterprise.license_file=/path/to/my/license/keyfile

#  to enable both GDS and Bloom:
# * dbms.security.procedures.unrestricted=gds.*,bloom.*
# * dbms.security.procedures.allowlist=gds.*,bloom.*
# * gds.enterprise.license_file=/path/to/my/license/keyfile
# * dbms.bloom.license_file=/path/to/my/license/keyfile

# to enable GenAI:
# * dbms.security.procedures.unrestricted=genai.*
# * dbms.security.procedures.allowlist=genai.*

# to enable Neosemantics:
# * dbms.security.procedures.unrestricted=n10s.*
# * dbms.security.procedures.allowlist=n10s.*
```

   For more information on configuring the plugins, see the respective documentation:

   - APOC Core documentation
   - APOC Extended documentation
   - Bloom documentation
   - Fleet management documentation
   - GDS documentation
   - GenAI documentation
   - Neosemantics documentation

3. Restart Neo4j for the plugins to be loaded and available for use.

> **Note:**
>
> All installed plugins will automatically be loaded every time Neo4j is started. Because of that, the number of plugins may impact the startup time. Install only the necessary plugins to avoid performance issues.

Update dynamic settings

Neo4j Enterprise Edition supports changing some configuration settings at runtime, without restarting the service.

> **Note:**
>
> Changes to the configuration at runtime are not persisted. To avoid losing changes when restarting Neo4j, make sure you update *neo4j.conf* as well.
>
> In a clustered environment, `CALL dbms.setConfigValue` affects only the cluster member it is run against, and it is not propagated to other members. If you want to change the configuration settings on all cluster members, you have to run the procedure against each of them and update their *neo4j.conf* file.

Discover dynamic settings

Use `SHOW SETTINGS` to discover which configuration values can be dynamically updated, or consult the Configuration settings reference.

Example 28. Discover dynamic settings

```
SHOW SETTINGS
YIELD name, isDynamic
WHERE isDynamic
RETURN name
```

```
+-------------------------------------------------------------------+
| name                                                              |
+-------------------------------------------------------------------+
| "db.checkpoint.iops.limit"                                        |
| "db.format"                                                       |
| "db.lock.acquisition.timeout"                                     |
| "db.logs.query.annotation_data_as_json_enabled"                   |
| "db.logs.query.annotation_data_format"                            |
| "db.logs.query.early_raw_logging_enabled"                         |
| "db.logs.query.enabled"                                           |
| "db.logs.query.max_parameter_length"                              |
| "db.logs.query.obfuscate_errors"
| "db.logs.query.obfuscate_literals"                                |
| "db.logs.query.parameter_logging_enabled"                         |
| "db.logs.query.plan_description_enabled"                          |
| "db.logs.query.threshold"                                         |
| "db.logs.query.transaction.enabled"                               |
| "db.logs.query.transaction.threshold"                             |
| "db.memory.transaction.max"                                       |
```

```
| "db.memory.transaction.total.max"                                |
| "db.track_query_cpu_time"                                        |
| "db.transaction.bookmark_ready_timeout"                          |
| "db.transaction.concurrent.maximum"                              |
| "db.transaction.sampling.percentage"                             |
| "db.transaction.timeout"                                         |
| "db.transaction.tracing.level"                                   |
| "db.tx_log.preallocate"                                          |
| "db.tx_log.rotation.retention_policy"                            |
| "db.tx_log.rotation.size"                                        |
| "dbms.cluster.network.connect_timeout"                           |
| "dbms.cypher.render_plan_description"                            |
| "dbms.memory.transaction.total.max"                              |
| "dbms.routing.client_side.enforce_for_domains"                   |
| "dbms.routing.reads_on_writers_enabled"                          |
| "dbms.security.key.name"                                         |
| "dbms.security.keystore.password"                                |
| "dbms.security.keystore.path"                                    |
| "dbms.security.ldap.authentication.attribute"                    |
| "dbms.security.ldap.authentication.user_dn_template"             |
| "dbms.security.ldap.authorization.access_permitted_group"        |
| "dbms.security.ldap.authorization.group_membership_attributes" |
| "dbms.security.ldap.authorization.group_to_role_mapping"         |
| "dbms.security.ldap.authorization.nested_groups_enabled"         |
| "dbms.security.ldap.authorization.nested_groups_search_filter" |
| "dbms.security.ldap.authorization.user_search_base"              |
| "dbms.security.ldap.authorization.user_search_filter"            |
| "server.cluster.catchup.connect_randomly_to_server_group"        |
| "server.cluster.catchup.connect_randomly_to_server_tags"         |
| "server.databases.default_to_read_only"                          |
| "server.databases.read_only"                                     |
| "server.databases.writable"                                      |
| "server.memory.pagecache.flush.buffer.enabled"                   |
| "server.memory.pagecache.flush.buffer.size_in_pages"             |
| "server.memory.query_cache.per_db_cache_num_entries"             |
| "server.memory.query_cache.shared_cache_num_entries"             |
+------------------------------------------------------------------+

51 rows
```

Update dynamic settings

An administrator is able to change some configuration settings at runtime, without restarting the service.

Syntax:

```
CALL dbms.setConfigValue( setting ,  value )
```

Returns:

Nothing on success.

Exceptions:

Unknown or invalid setting name.
The setting is not dynamic and cannot be changed at runtime.
Invalid setting value.

The following example shows how to dynamically enable query logging.

Example 29. Set a config value

```
CALL dbms.setConfigValue('db.logs.query.enabled', 'info')
```

If an invalid value is passed, the procedure will show a message to that effect.

Example 30. Try to set invalid config value

```
CALL dbms.setConfigValue('db.logs.query.enabled', 'yes')
```

```
Failed to invoke procedure `dbms.setConfigValue`: Caused by:
org.neo4j.graphdb.config.InvalidSettingException: Bad value 'yes' for setting
'db.logs.query.enabled': 'yes' not one of [OFF, INFO, VERBOSE]
```

To reset a config value to its default, pass an empty string as the *value* argument.

Example 31. Reset a config value to default

```
CALL dbms.setConfigValue('db.logs.query.enabled', '')
```

Configuration settings

The Neo4j configuration settings are set in *neo4j.conf*. Refer to The neo4j.conf file for details on how to use configuration settings.

For lists of deprecated, renamed, and removed configuration settings in 5.x, refer to the Upgrade and Migration Guide → Neo4j 5 → Reference.

To list all available configuration settings on a Neo4j server, run the `SHOW SETTINGS` command.

Dynamic configuration settings

Dynamic settings can be changed at runtime, without restarting the service.

Dynamic settings are labeled Dynamic.

Note:

Changes to the configuration at runtime are not persisted. To avoid losing changes when restarting Neo4j, make sure you update *neo4j.conf* as well.

In a clustered environment, `CALL dbms.setConfigValue` affects only the server it is run against, and it is not propagated to other members. If you want to change the configuration settings on all cluster members, you have to run the procedure against each of them and update their *neo4j.conf* file.

Each member of the cluster has its own *neo4j.conf* file. It is recommended that the settings for a database are the same across all members of the cluster.

For more information on how to update dynamic configuration settings, see Update dynamic settings.

Configuration setting group

Enterprise

When deploying a multi-data cluster in Neo4j, you can configure the load balancing framework.

In Neo4j, the load balancing system is based on a plugin architecture. The primary built-in plugin is `server_policies`, which is set up by the following property:

```
dbms.routing.load_balancing.plugin=server_policies
```

`server_policies` plugin determines which servers are eligible to serve client requests based on predefined routing policies. If a client does not specify a routing policy, the system defaults to using all available servers.

You can define routing policies by using the following property format:

```
dbms.routing.load_balancing.config.server_policies.<policy-name>=<policy-definition>
```

Where `<policy-name>` is the name of the routing policy, and `<policy-definition>` specifies the server selection logic.

For the default policy, the `default` policy name is reserved. Its default value is `all()`:

```
dbms.routing.load_balancing.config.server_policies.default=all()
```

See Clustering → Multi-data center routing for more details.

Checkpoint settings

Checkpointing is the process of flushing all pending page updates from the page cache to the store files. This is done periodically and is used to recover the database in case of a crash. The checkpoint settings control the frequency of checkpoints, and the amount of data that is written to disk in each checkpoint. See also, Transaction log settings.

`db.checkpoint`

Table 47. db.checkpoint

Description	Configures the general policy for when checkpoints should occur. Possible values are: • `PERIODIC` (default)- it runs a checkpoint as per the interval specified by `db.checkpoint.interval.tx` and `db.checkpoint.interval.time`. • `VOLUME` — it runs a checkpoint when the size of the transaction logs reaches the value specified by the `db.checkpoint.interval.volume` setting. By default, it is set to `250.00MiB`. • `CONTINUOUS` (Enterprise Edition) — it ignores `db.checkpoint.interval.tx` and `db.checkpoint.interval.time` settings and runs the checkpoint process all the time. • `VOLUMETRIC` (Enterprise Edition) — it makes the best effort to checkpoint often enough so that the database does not get too far behind on deleting old transaction logs as specified in the `db.tx_log.rotation.retention_policy` setting.
Valid values	One of [PERIODIC, CONTINUOUS, VOLUME, VOLUMETRIC].
Default value	`PERIODIC`

`db.checkpoint.interval.time`

Table 48. db.checkpoint.interval.time

Description	Configures the time interval between checkpoints. The database does not checkpoint more often than the specified interval (unless checkpointing is triggered by a different event) but might checkpoint less often if performing a checkpoint takes longer time than the configured interval. A checkpoint is a point in the transaction logs from which recovery starts. Longer checkpoint intervals typically mean that recovery takes longer to complete in case of a crash. On the other hand, a longer checkpoint interval can also reduce the I/O load that the database places on the system, as each checkpoint implies a flushing and forcing of all the store files.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`15m`

`db.checkpoint.interval.tx`

Table 49. db.checkpoint.interval.tx

Description	Configures the transaction interval between checkpoints. The database does not checkpoint more often than the specified interval (unless checkpointing is triggered by a different event) but might checkpoint less often if performing a checkpoint takes longer time than the configured interval. A checkpoint is a point in the transaction logs from which recovery starts. Longer checkpoint intervals typically mean that recovery takes longer to complete in case of a crash. On the other hand, a longer checkpoint interval can also reduce the I/O load that the database places on the system, as each checkpoint implies a flushing and forcing of all the store files. The default is `100000` for a checkpoint every 100000 transactions.
Valid values	An integer that is minimum `1`.
Default value	`100000`

`db.checkpoint.interval.volume`

Table 50. db.checkpoint.interval.volume

Description	Configures the volume of transaction logs between checkpoints. The database does not checkpoint more often than the specified interval (unless checkpointing is triggered by a different event) but might checkpoint less often if performing a checkpoint takes longer time than the configured interval. A checkpoint is a point in the transaction logs, which recovery would start from. Longer checkpoint intervals typically mean that recovery takes longer to complete in case of a crash. On the other hand, a longer checkpoint interval can also reduce the I/O load that the database places on the system, as each checkpoint implies a flushing and forcing of all the store files.
Valid values	A byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `T` `PiB`, `PB`, `EiB`, `EB`) that is minimum `1.00KiB`.
Default value	`250.00MiB`

`db.checkpoint.iops.limit`

Dynamic

Enterprise Edition Dynamic

Table 51. db.checkpoint.iops.limit

Description	Limit the number of IOs the background checkpoint process consumes per second. This setting is advisory. It is ignored in Neo4j Community Edition and is followed to best effort in Enterprise Edition. An IO is, in this case, an 8 KiB (mostly sequential) write. Limiting the write IO in this way leaves more bandwidth in the IO subsystem to service random-read IOs, which is important for the response time of queries when the database cannot fit entirely in memory. The only drawback of this setting is that longer checkpoint times may lead to slightly longer recovery times in case of a database or system crash. A lower number means lower IO pressure and, consequently, longer checkpoint times. Set this to `-1` to disable the IOPS limit and remove the limitation entirely. This lets the checkpointer flush data as fast as the hardware goes. Removing or commenting out the setting sets the default value of `600`.
Valid values	An integer.
Default value	`600`

Cloud storage integration settings

Configuration settings enable custom cloud storage and host authority endpoints, supporting non-public and private cloud Azure deployments.

> **Note:**
>
> Note that support for non-public and private cloud Azure deployments is introduced in Neo4j 5.26.4 and Neo4j 2025.03. This means the configuration options are available in Neo4j 5.26 LTS version starting from the 5.26.4 patch release and in Neo4j 2025.x series starting from 2025.03.

`dbms.integrations.cloud_storage.azb.blob_endpoint_suffix` Available in 5.26.4

Enterprise Edition

Table 52. dbms.integrations.cloud_storage.azb.blob_endpoint_suffix

Description	Azure blob storage endpoint suffix. You need to change this if you are not using Azure public cloud (e.g., if you are using Azure Government).
Valid values	A string.
Default value	`blob.core.windows.net`

`dbms.integrations.cloud_storage.azb.authority_endpoint` Available in 5.26.4

Enterprise Edition

Table 53. dbms.integrations.cloud_storage.azb.authority_endpoint

Description	Azure authority host endpoint (only required for certain methods of authentication, it should be specified in its full form - e.g., https://login.microsoftonline.com)
Valid values	A string.
Default value	`""`

Cluster settings

The cluster settings are used to configure the behavior of a Neo4j cluster. For more information, see also Clustering settings.

`db.cluster.catchup.pull_interval`

Enterprise Edition

Table 54. db.cluster.catchup.pull_interval

Description	The interval at which a secondary server fetches updates for a specific database from the primary server for that database.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`1s`

`db.cluster.raft.apply.buffer.max_bytes`

Enterprise Edition

Table 55. db.cluster.raft.apply.buffer.max_bytes

Description	The maximum number of bytes in the apply buffer. This parameter limits the amount of memory that can be consumed by the apply buffer. If the bytes limit is reached, buffer size will be limited even if max_entries is not exceeded.
Valid values	A byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `T` `PiB`, `PB`, `EiB`, `EB`).

Default value	`1.00GiB`

`db.cluster.raft.apply.buffer.max_entries`

Enterprise Edition

Table 56. db.cluster.raft.apply.buffer.max_entries

Description	The maximum number of entries in the raft log entry prefetch buffer.
Valid values	An integer.
Default value	`1024`

`db.cluster.raft.in_queue.batch.max_bytes`

Enterprise Edition

Table 57. db.cluster.raft.in_queue.batch.max_bytes

Description	Largest batch processed by RAFT in bytes.
Valid values	A byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `T` `PiB`, `PB`, `EiB`, `EB`).
Default value	`8.00MiB`

`db.cluster.raft.so_keepalive_enabled`

Enterprise Edition Introduced in 5.23

Table 58. db.cluster.raft.so_keepalive_enabled

Description	Set the keepalive socket option (SO_KEEPALIVE) for all Raft TCP channels.
Valid values	A boolean.
Default value	`false`

`db.cluster.raft.in_queue.max_bytes`

Enterprise Edition

Table 59. db.cluster.raft.in_queue.max_bytes

Description	Maximum number of bytes in the RAFT in-queue.
Valid values	A byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `T` `PiB`, `PB`, `EiB`, `EB`).
Default value	`2.00GiB`

db.cluster.raft.leader_transfer.priority_group

Deprecated 5.4"

Table 60. db.cluster.raft.leader_transfer.priority_group

Description	The name of a server_group whose members should be prioritized as leaders. This does not guarantee that the leader will always be a member of this group, but the cluster will attempt to transfer the leadership to such a member when possible. If a database is specified using `db.cluster.raft.leader_transfer.priority_group.<database>`, the specified priority group will apply to that database only. If no database is specified, that group will be the default and apply to all databases with no explicitly set priority group. Using this setting will disable leadership balancing.
Valid values	A string identifying a server tag.
Default value	

db.cluster.raft.leader_transfer.priority_tag

Enterprise Edition

Table 61. db.cluster.raft.leader_transfer.priority_tag

Description	The name of a server tag whose members should be prioritized as leaders. This does not guarantee that the leader will always be a member of this tag, but the cluster will attempt to transfer the leadership to such a member when possible. If a database is specified using `db.cluster.raft.leader_transfer.priority_tag`.<database>, the specified priority tag will apply only to that database. If no database is specified, that tag will be the default and apply to all databases with no explicitly set priority tag. Using this setting will disable leadership balancing.
Valid values	A string identifying a server tag.
Default value	

db.cluster.raft.log.prune_strategy

Enterprise Edition

Table 62. db.cluster.raft.log.prune_strategy

Description	RAFT log pruning strategy that determines which logs are to be pruned. Neo4j only prunes log entries up to the last applied index, which guarantees that logs are only marked for pruning once the transactions within are safely copied over to the local transaction logs and safely committed by a majority of cluster members. Possible values are a byte size or a number of transactions (e.g., 200K txs).
Valid values	A string.
Default value	`1g size`

db.cluster.raft.log_shipping.buffer.max_bytes

Enterprise Edition

Table 63. db.cluster.raft.log_shipping.buffer.max_bytes

Description	The maximum number of bytes in the in-flight cache. This parameter limits the amount of memory that can be consumed by the cache. If the bytes limit is reached, cache size will be limited even if max_entries is not exceeded.
Valid values	A byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `T` `PiB`, `PB`, `EiB`, `EB`).
Default value	`1.00GiB`

`db.cluster.raft.log_shipping.buffer.max_entries`

Enterprise Edition

Table 64. db.cluster.raft.log_shipping.buffer.max_entries

Description	The maximum number of entries in the in-flight cache. Increasing size requires more memory but might improve performance in high-load situations.
Valid values	An integer.
Default value	`1024`

`dbms.cluster.catchup.client_inactivity_timeout`

Enterprise Edition Deprecated in 5.26

Table 65. dbms.cluster.catchup.client_inactivity_timeout

Description	The catch-up protocol times out if the given duration elapses with no network activity. Every message received by the client from the server extends the timeout duration.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`10m`

`dbms.cluster.network.client_inactivity_timeout`

Enterprise Edition Introduced in 5.26

Table 66. dbms.cluster.network.client_inactivity_timeout

Description	A network request times out if the given duration elapses with no network activity. Every message received by the client from the server extends the timeout duration.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`10m`

`dbms.cluster.discovery.endpoints`

Enterprise Edition Deprecated in 5.23

Table 67. dbms.cluster.discovery.endpoints

Description	A comma-separated list of endpoints that a server should contact in order to discover other cluster members. All cluster members hosting a system database primary must be specified in this list. However, it is typical that all cluster members, including the current server, are specified in this list. The setting configures the endpoints for Discovery service V1.
Valid values	A comma-separated list where each element is a socket address in the format of `hostname:port`, `hostname`, or `:port`.
Default value	

`dbms.cluster.discovery.v2.endpoints`

Enterprise Edition | Introduced in 5.22

Table 68. dbms.cluster.discovery.v2.endpoints

Description	A comma-separated list of endpoints that a server should contact in order to discover other cluster members. All cluster members hosting a system database primary must be specified in this list. However, it is typical that all cluster members, including the current server, are specified in this list. The setting configures the endpoints for Discovery service V2.
Valid values	A comma-separated list where each element is a socket address in the format of `hostname:port`, `hostname`, or `:port`.
Default value	

Note:

`dbms.cluster.discovery.endpoints` and `dbms.cluster.discovery.v2.endpoints` must contain all cluster members hosting a `system` database in primary mode. However, it is typical that all cluster members, including the current server, are specified in those lists.

`dbms.cluster.discovery.version`

Enterprise Edition | Introduced in 5.22 | Deprecated in 5.26

Table 69. dbms.cluster.discovery.version

Description	This setting allows you to select which discovery service should be started. Possible values are: • `V1_ONLY` — it runs only discovery service v1. • `V1_OVER_V2` — it runs both Discovery Service V1 and Discovery Service V2, where V1 is the main service and V2 runs in the background. • `V2_OVER_V1` — it runs both Discovery Service V1 and Discovery Service V2, where V2 is the main service and V1 runs in the background. • `V2_ONLY` — it runs only discovery service v2. This setting is deprecated because in the next major version it is removed, since there will only be the V2 discovery service.

Valid values	One of [V1_ONLY, V1_OVER_V2, V2_OVER_V1, V2_ONLY].
Default value	`V1_ONLY`

`dbms.cluster.discovery.log_level`

Enterprise Edition | Deprecated in 5.23

Table 70. dbms.cluster.discovery.log_level

Description	The level of middleware logging.
Valid values	One of [DEBUG, INFO, WARN, ERROR, NONE].
Default value	`WARN`

`dbms.cluster.discovery.resolver_type`

Enterprise Edition

Table 71. dbms.cluster.discovery.resolver_type

Description	Configure the resolver type that the discovery service uses for determining who should be part of the cluster. Valid values are `LIST`, `SRV`, `DNS`, and `K8S`:
	
	`LIST`
	A static configuration where `dbms.cluster.discovery.endpoints` must contain a list of the addresses of the cluster members.
	
	`SRV` and `DNS`
	A dynamic configuration where `dbms.cluster.discovery.endpoints` must point to a DNS entry containing the cluster members' addresses.
	
	`K8S`
	At least `dbms.kubernetes.service_port_name` must be set. The addresses of the cluster members are queried dynamically from Kubernetes.

Valid values	A string.
Default value	`LIST`

`dbms.cluster.discovery.type`

Enterprise Edition | Deprecated in 5.7

Table 72. dbms.cluster.discovery.type

Description	This setting has been replaced by `dbms.cluster.discovery.resolver_type`.
Valid values	One of [DNS, LIST, SRV, K8S].
Default value	`LIST`

dbms.cluster.minimum_initial_system_primaries_count

Enterprise Edition

Table 73. dbms.cluster.minimum_initial_system_primaries_count

Description	Minimum number of machines initially required to form a clustered DBMS. The cluster is considered formed when at least this many members have discovered each other, bound together, and bootstrapped a highly available system database. As a result, at least this many of the cluster's initial machines must have `server.cluster.system_database_mode` set to `PRIMARY`. NOTE: If `dbms.cluster.discovery.resolver_type` is set to `LIST` and `dbms.cluster.discovery.endpoints` is empty, then the user is assumed to be deploying a standalone DBMS, and the value of this setting is ignored.
Valid values	An integer that is minimum `1`.
Default value	`3`

dbms.cluster.network.connect_timeout

Enterprise Edition Dynamic Introduced in 5.17

Table 74. dbms.cluster.network.connect_timeout

Description	The maximum amount of time to wait for a network connection to be established.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`30s`

dbms.cluster.network.handshake_timeout

Enterprise Edition

Table 75. dbms.cluster.network.handshake_timeout

Description	Time out for protocol negotiation handshake.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`20s`

dbms.cluster.network.max_chunk_size

Enterprise Edition

Table 76. dbms.cluster.network.max_chunk_size

Description	Maximum chunk size allowable across a network by clustering machinery.
Valid values	An integer that is in the range `4096` to `10485760`.

Default value	32768

dbms.cluster.network.supported_compression_algos

Enterprise Edition

Table 77. dbms.cluster.network.supported_compression_algos

Description	Network compression algorithms that this instance will allow in negotiation as a comma-separated list. For incoming connections, the algorithms are listed in descending order of preference. An empty list implies no compression. For outgoing connections, this merely specifies the allowed set of algorithms and the preference of the remote peer will be used for making the decision. Allowable values: [Gzip, Snappy, Snappy_validating, LZ4, LZ4_high_compression, LZ_validating, LZ4_high_compression_validating]
Valid values	A comma-separated list where each element is a string.
Default value	

dbms.cluster.raft.binding_timeout

Enterprise Edition

Table 78. dbms.cluster.raft.binding_timeout

Description	The time allowed for a database on a Neo4j server to either join a cluster or form a new cluster with at least the quorum of the members available. The members are provided by `dbms.cluster.discovery.endpoints` for the system database and by the topology graph for standard databases.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	1d

dbms.cluster.raft.client.max_channels

Enterprise Edition

Table 79. dbms.cluster.raft.client.max_channels

Description	The maximum number of TCP channels between two nodes to operate the raft protocol. Each database gets allocated one channel, but a single channel can be used by more than one database.
Valid values	An integer.
Default value	8

dbms.cluster.raft.election_failure_detection_window

Enterprise Edition

Table 80. dbms.cluster.raft.election_failure_detection_window

Description	The rate at which leader elections happen. Note that due to election conflicts, it might take several attempts to find a leader. The window should be significantly larger than typical communication delays to make conflicts unlikely.
Valid values	A duration-range <min-max> (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`3s-6s`

`dbms.cluster.raft.leader_failure_detection_window`

Enterprise Edition

Table 81. dbms.cluster.raft.leader_failure_detection_window

Description	The time window within which the loss of the leader is detected and the first re-election attempt is held. The window should be significantly larger than typical communication delays to make conflicts unlikely.
Valid values	A duration-range <min-max> (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`20s-23s`

`dbms.cluster.raft.leader_transfer.balancing_strategy`

Enterprise Edition

Table 82. dbms.cluster.raft.leader_transfer.balancing_strategy

Description	Which strategy to use when transferring database leaderships around a cluster. Note that if a `leadership_priority_group` is specified for a given database, the value of this setting will be ignored for that database. The following values are available: • `equal_balancing` automatically ensures that each primary server holds the leader role for an equal number of databases. • `no_balancing` prevents any automatic balancing of the leader role.
Valid values	One of [NO_BALANCING, EQUAL_BALANCING].
Default value	`EQUAL_BALANCING`

`dbms.cluster.raft.log.pruning_frequency`

Enterprise Edition

Table 83. dbms.cluster.raft.log.pruning_frequency

Description	RAFT log pruning frequency.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`10m`

dbms.cluster.raft.log.reader_pool_size

Enterprise Edition

Table 84. dbms.cluster.raft.log.reader_pool_size

Description	RAFT log reader pool size.
Valid values	An integer.
Default value	`8`

dbms.cluster.raft.log.rotation_size

Enterprise Edition

Table 85. dbms.cluster.raft.log.rotation_size

Description	RAFT log rotation size. The log will be rotated when it reaches this size.
Valid values	A byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `T` `PiB`, `PB`, `EiB`, `EB`) that is minimum `1.00KiB`.
Default value	`250.00MiB`

dbms.cluster.raft.membership.join_max_lag

Enterprise Edition

Table 86. dbms.cluster.raft.membership.join_max_lag

Description	Maximum amount of lag accepted for a new follower to join the Raft group.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`10s`

dbms.cluster.raft.membership.join_timeout

Enterprise Edition

Table 87. dbms.cluster.raft.membership.join_timeout

Description	Timeout for a new member to catch up.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`10m`

dbms.cluster.store_copy.max_retry_time_per_request

Enterprise Edition

Table 88. dbms.cluster.store_copy.max_retry_time_per_request

Description	Maximum retry time per request during store copy. Regular store files and indexes are downloaded in separate requests during store copy. This configures the maximum time failed requests are allowed to resend.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`20m`

`initial.dbms.automatically_enable_free_servers`

Enterprise Edition | Introduced in 5.10

Table 89. initial.dbms.automatically_enable_free_servers

Description	Automatically enables servers that are in the `FREE` state - not only during the initial DBMS startup but also whenever a new server joins the cluster.
Valid values	A boolean.
Default value	`false`

`initial.dbms.database_allocator`

Enterprise Edition | Deprecated in 5.23

Table 90. initial.dbms.database_allocator

Description	Name of the initial database allocator. After the creation of the DBMS, it can be set by running the `CALL dbms.setDatabaseAllocator()` procedure.
Valid values	A string.
Default value	`EQUAL_NUMBERS`

`initial.dbms.default_primaries_count`

Enterprise Edition

Table 91. initial.dbms.default_primaries_count

Description	The initial default number of primaries for the standard databases. Initialized at the first DBMS startup. If the user does not specify the number of primaries in `CREATE DATABASE`, this value will be used unless overwritten by the `dbms.setDefaultAllocationNumbers` procedure.
Valid values	An integer that is minimum `1` and is maximum `11`.
Default value	1

`initial.dbms.default_secondaries_count`

Enterprise Edition

Table 92. initial.dbms.default_secondaries_count

Description	The initial default number of secondaries for the standard databases. Initialized at the first DBMS startup. If the user does not specify the number of secondaries in `CREATE DATABASE`, this value will be used unless overwritten by the `dbms.setDefaultAllocationNumbers` procedure.
Valid values	An integer that is minimum `0` and is maximum `20`.
Default value	`0`

`initial.server.allowed_databases`

Enterprise Edition

Table 93. initial.server.allowed_databases

Description	List of database names allowed on this server; all others are denied. Empty means all are allowed. This configuration is initialized at the first DBMS startup and/or when a newly added server is enabled. The setting is used as the default input for the `ENABLE SERVER` command; can be overriden when the command is executed. Exclusive with `server.initial_denied_databases`.
Valid values	A comma-separated set where each element is a valid database name containing only alphabetic characters, numbers, dots, and dashes with a length between 3 and 63 characters, starting with an alphabetic character but not with the name `system`.
Default value	

`initial.server.denied_databases`

Enterprise Edition

Table 94. initial.server.denied_databases

Description	List of database names not allowed on this server. Empty means nothing is denied. This configuration is initialized at the first DBMS startup and/or when a newly added server is enabled. The setting is used as the default input for the `ENABLE SERVER` command; can be overriden when the command is executed. Exclusive with `server.initial_allowed_databases`.
Valid values	A comma-separated set where each element is a valid database name containing only alphabetic characters, numbers, dots, and dashes with a length between 3 and 63 characters, starting with an alphabetic character but not with the name `system`.
Default value	

`initial.server.mode_constraint`

Enterprise Edition

Table 95. initial.server.mode_constraint

Description	Determines whether the server is configured to host primary databases only, secondary databases only, or both. Initialized at the first DBMS startup and/or when a newly added server is enabled. The setting is used as the default input for the `ENABLE SERVER` command; can be overriden when the command is executed.

Valid values	One of [PRIMARY, SECONDARY, NONE].
Default value	NONE

initial.server.tags

Enterprise Edition

Table 96. initial.server.tags

Description	A list of server tag names used by the database allocation and when configuring load balancing and replication policies. Initialized at the first DBMS startup and/or when a newly added server is enabled. The setting is used as the default input for the `ENABLE SERVER` command; can be overriden when the command is executed.
Valid values	A comma-separated list where each element is a string identifying a server tag, which contains no duplicate items.
Default value	

server.cluster.advertised_address

Enterprise Edition

Table 97. server.cluster.advertised_address

Description	Advertised hostname/IP address and port for the transaction shipping server.
Valid values	A socket address in the format of `hostname:port`, `hostname`, or `:port` that is an accessible address. If missing, it is acquired from server.default_advertised_address.
Default value	`:6000`

server.cluster.catchup.connect_randomly_to_server_group

Enterprise Edition Dynamic Deprecated in 5.4

Table 98. server.cluster.catchup.connect_randomly_to_server_group

Description	Comma-separated list of groups to be used by the connect-randomly-to-server-group selection strategy. The connect-randomly-to-server-group strategy is used when the list of strategies (`server.cluster.catchup.upstream_strategy`) includes the value `connect-randomly-to-server-group`.
Valid values	A comma-separated list where each element is a string identifying a server tag.
Default value	

server.cluster.catchup.connect_randomly_to_server_tags

Enterprise Edition Dynamic Introduced in 5.5

Table 99. server.cluster.catchup.connect_randomly_to_server_tags

Description	Comma-separated list of tags to be used by the connect-randomly-to-server-with-tag selection strategy. The connect-randomly-to-server-with-tag strategy is used when the list of strategies (`server.cluster.catchup.upstream_strategy`) includes the value `connect-randomly-to-server-with-tag`.
Valid values	A comma-separated list where each element is a string identifying a server tag.
Default value	

`server.cluster.catchup.upstream_strategy`

Enterprise Edition

Table 100. server.cluster.catchup.upstream_strategy

Description	A descending-ordered list of strategies secondaries use to choose the upstream server from which to pull transactional updates. If none are valid or the list is empty, the default strategy is `typically-connect-to-random-secondary`.
Valid values	A comma-separated list where each element is a string.
Default value	

`server.cluster.catchup.user_defined_upstream_strategy`

Enterprise Edition

Table 101. server.cluster.catchup.user_defined_upstream_strategy

Description	Configuration of a user-defined upstream selection strategy. The user-defined strategy is used when the list of strategies (`server.cluster.catchup.upstream_strategy`) includes the value `user_defined`.
Valid values	A string.
Default value	

`server.cluster.listen_address`

Enterprise Edition

Table 102. server.cluster.listen_address

Description	Network interface and port for the transaction shipping server to listen on. Note that it is also possible to run the backup client against this port, so always limit access to it via the firewall and configure an SSL policy.
Valid values	A socket address in the format of `hostname:port`, `hostname`, or `:port`. If missing, it is acquired from server.default_listen_address.
Default value	`:6000`

`server.cluster.network.native_transport_enabled`

Enterprise Edition

Table 103. server.cluster.network.native_transport_enabled

Description	Use native transport if available. Epoll for Linux or Kqueue for MacOS/BSD. If this setting is set to `false`, or if native transport is not available, Nio transport will be used.
Valid values	A boolean.
Default value	`true`

`server.cluster.raft.advertised_address`

Enterprise Edition

Table 104. server.cluster.raft.advertised_address

Description	Advertised hostname/IP address and port for the RAFT server.
Valid values	A socket address in the format of `hostname:port`, `hostname`, or `:port` that is an accessible address. If missing, it is acquired from server.default_advertised_address.
Default value	`:7000`

`server.cluster.raft.listen_address`

Enterprise Edition

Table 105. server.cluster.raft.listen_address

Description	Network interface and port for the RAFT server to listen on.
Valid values	A socket address in the format of `hostname:port`, `hostname`, or `:port`. If missing, it is acquired from server.default_listen_address.
Default value	`:7000`

`server.cluster.system_database_mode`

Enterprise Edition

Table 106. server.cluster.system_database_mode

Description	Users must manually specify the mode for the system database on each server.
Valid values	One of [PRIMARY, SECONDARY].
Default value	`PRIMARY`

`server.discovery.listen_address`

Enterprise Edition Deprecated in 5.23

Table 107. server.discovery.listen_address

Description	Host and port to bind the cluster member discovery management communication.
Valid values	A socket address in the format of `hostname:port`, `hostname`, or `:port`. If missing, it is acquired from server.default_listen_address.
Default value	`:5000`

`server.groups`

Enterprise Edition Deprecated in 5.4

Table 108. server.groups

Description	A list of tag names for the server used when configuring load balancing and replication policies.
Valid values	A comma-separated list where each element is a string identifying a server tag.
Default value	
Replaced by	`initial.server.tags`

Connection settings

Connection settings control the communication between servers and between a server and a client. Neo4j provides support for Bolt, HTTP, and HTTPS protocols via connectors. For more information about the connectors, see Configure connectors.

When configuring the HTTPS or Bolt, see also Security settings and SSL framework for details on how to work with SSL certificates.

`server.bolt.advertised_address`

Table 109. server.bolt.advertised_address

Description	Advertised address for this connector.
Valid values	A socket address in the format of `hostname:port`, `hostname`, or `:port` that is an accessible address. If missing, it is acquired from `server.default_advertised_address`.
Default value	`:7687`

`server.bolt.connection_keep_alive`

Table 110. server.bolt.connection_keep_alive

Description	The maximum time to wait before sending a NOOP on connections waiting for responses from active ongoing queries.The minimum value is 1 millisecond.

Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) that is minimum `1ms`.
Default value	`1m`

`server.bolt.connection_keep_alive_for_requests`

Table 111. server.bolt.connection_keep_alive_for_requests

Description	The type of messages to enable keep-alive messages for `ALL`, `STREAMING`, or `OFF`.
Valid values	One of [ALL, STREAMING, OFF].
Default value	`ALL`

`server.bolt.connection_keep_alive_probes`

Table 112. server.bolt.connection_keep_alive_probes

Description	The total number of probes to be missed before a connection is considered stale. The minimum value is 1.
Valid values	An integer that is minimum `1`.
Default value	`2`

`server.bolt.connection_keep_alive_streaming_scheduling_interval`

Table 113. server.bolt.connection_keep_alive_streaming_scheduling_interval

Description	The interval between every scheduled keep-alive check on all connections with active queries. Zero duration turns off keep-alive service.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) that is minimum `0s`.
Default value	`1m`

`server.bolt.enabled`

Table 114. server.bolt.enabled

Description	Enable the Bolt connector.
Valid values	A boolean.
Default value	`true`

`server.bolt.listen_address`

Table 115. server.bolt.listen_address

Description	Address the connector should bind to.
Valid values	A socket address in the format of `hostname:port`, `hostname`, or `:port`. If missing, it is acquired from server.default_listen_address.

Default value	`:7687`

`server.bolt.additional_listen_addresses`

Introduced in 5.23

Table 116. server.bolt.additional_listen_addresses

Description	Additional addresses the connector should bind to.
Valid values	A comma-separated set where each element is a socket address in the format of `hostname:port`, `hostname`, or `:port`.
Default value	

`server.bolt.ocsp_stapling_enabled`

Table 117. server.bolt.ocsp_stapling_enabled

Description	Enable server OCSP stapling for bolt and http connectors.
Valid values	A boolean.
Default value	`false`

`server.bolt.telemetry.enabled`

Introduced in 5.4

Table 118. server.bolt.telemetry.enabled

Description	Enable the collection of driver telemetry.
Valid values	A boolean.
Default value	`false`

`server.bolt.enable_network_error_accounting`

Introduced in 5.18

Table 119. server.bolt.enable_network_error_accounting

Description	Enables accounting-based reporting of benign errors within the Bolt stack. When enabled, benign errors are reported only when such events occur with unusual frequency. When disabled, all benign network errors are reported.
Valid values	A boolean.
Default value	`true`

`server.bolt.network_abort_clear_window_duration`

Introduced in 5.18

Table 120. server.bolt.network_abort_clear_window_duration

Description	The duration for which network-related connection aborts need to remain at a reasonable level before the error is cleared.
Valid values	A duration (Valid units are: ns, µs, ms, s, m, h and d; default unit is s) that is minimum `1s`.
Default value	`10m`

`server.bolt.network_abort_warn_threshold`

Introduced in 5.18

Table 121. server.bolt.network_abort_warn_threshold

Description	The maximum number of network-related connection aborts allowed within a specified time window before emitting log messages. A value of zero reverts to legacy warning behavior.
Valid values	A long that is minimum `0`.
Default value	`2`

`server.bolt.network_abort_warn_window_duration`

Introduced in 5.18

Table 122. server.bolt.network_abort_warn_window_duration

Description	The duration of the window in which network-related connection aborts are sampled.
Valid values	A duration (Valid units are: ns, µs, ms, s, m, h and d; default unit is s) that is minimum `1s`.
Default value	`10m`

`server.bolt.thread_pool_keep_alive`

Table 123. server.bolt.thread_pool_keep_alive

Description	The maximum time an idle thread in the thread pool bound to this connector waits for new tasks.
Valid values	A duration (Valid units are: `ns`, `µ s`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`5m`

`server.bolt.thread_pool_max_size`

Table 124. server.bolt.thread_pool_max_size

Description	The maximum number of threads allowed in the thread pool bound to this connector.
Valid values	An integer.

Default value	`400`

`server.bolt.thread_pool_min_size`

Table 125. server.bolt.thread_pool_min_size

Description	The number of threads, including idle, to keep in the thread pool bound to this connector.
Valid values	An integer.
Default value	`5`

`server.bolt.thread_starvation_clear_window_duration`

Introduced in 5.18

Table 126. server.bolt.thread_starvation_clear_window_duration

Description	The duration for which unscheduled requests need to remain at a reasonable level before the error is cleared.
Valid values	A duration (Valid units are: ns, µs, ms, s, m, h and d; default unit is s) that is minimum `1s`.
Default value	`10m`

`server.bolt.thread_starvation_warn_threshold`

Introduced in 5.18

Table 127. server.bolt.thread_starvation_warn_threshold

Description	The maximum number of unscheduled requests allowed during thread starvation events within a specified time window before emitting log messages.
Valid values	A long that is minimum `0`.
Default value	`2`

`server.bolt.thread_starvation_warn_window_duration`

Introduced in 5.18

Table 128. server.bolt.thread_starvation_warn_window_duration

Description	The duration of the window in which unscheduled requests are sampled.
Valid values	A duration (Valid units are: ns, µs, ms, s, m, h and d; default unit is s) that is minimum `1s`.
Default value	`10m`

`server.bolt.tls_level`

Table 129. server.bolt.tls_level

Description	The encryption level to be used to secure communications with this connector.
Valid values	One of [REQUIRED, OPTIONAL, DISABLED].
Default value	`DISABLED`

`server.bolt.traffic_accounting_check_period`

Introduced in 5.18

Table 130. server.bolt.traffic_accounting_check_period

Description	Amount of time spent between samples of current traffic usage. Lower values result in more accurate reporting while incurring a higher performance penalty. A value of zero disables traffic accounting.
Valid values	A duration (Valid units are: ns, µs, ms, s, m, h and d; default unit is s) that is 0s or is minimum `1m`.
Default value	`5m`

`server.bolt.traffic_accounting_clear_duration`

Introduced in 5.18

Table 131. server.bolt.traffic_accounting_clear_duration

Description	Time to be spent below the configured traffic threshold to clear traffic warnings.
Valid values	A duration (Valid units are: ns, µs, ms, s, m, h and d; default unit is s) that is minimum `1m`.
Default value	`10m`

`server.bolt.traffic_accounting_incoming_threshold_mbps`

Introduced in 5.18

Table 132. server.bolt.traffic_accounting_incoming_threshold_mbps

Description	Maximum permitted incoming traffic within a configured accounting check window before emitting a warning (in Mbps).
Valid values	A long that is minimum `1`.
Default value	`950`

`server.bolt.traffic_accounting_outgoing_threshold_mbps`

Introduced in 5.18

Table 133. server.bolt.traffic_accounting_outgoing_threshold_mbps

Description	Maximum permitted outgoing traffic within a configured accounting check window before emitting a warning (in Mbps).
Valid values	A long that is minimum `1`.
Default value	`950`

`server.http.advertised_address`

Table 134. server.http.advertised_address

Description	Advertised address for this connector.
Valid values	A socket address in the format of `hostname:port`, `hostname`, or `:port` that is an accessible address. If missing, it is acquired from server.default_advertised_address.
Default value	`:7474`

`server.http.enabled`

Table 135. server.http.enabled

Description	Enable the HTTP connector.
Valid values	A boolean.
Default value	`true`

`server.http.listen_address`

Table 136. server.http.listen_address

Description	Address the connector should bind to.
Valid values	A socket address in the format of `hostname:port`, `hostname`, or `:port`. If missing, it is acquired from server.default_listen_address.
Default value	`:7474`

`server.http_enabled_modules`

Table 137. server.http_enabled_modules

Description	Defines the set of modules loaded into the Neo4j web server. The enterprise management endpoints are only available in the Enterprise edition.
Valid values	A comma-separated set where each element is one of [TRANSACTIONAL_ENDPOINTS, UNMANAGED_EXTENSIONS, BROWSER, ENTERPRISE_MANAGEMENT_ENDPOINTS, QUERY_API_ENDPOINTS].
Default value	`TRANSACTIONAL_ENDPOINTS,UNMANAGED_EXTENSIONS,BROWSER,ENTERPRISE_MANAGEMENT_ENDPOINTS,QUERY_API_ENDPOINTS`

`server.http_enabled_transports`

Introduced in 5.17

Table 138. server.http_enabled_transports

Description	Defines the set of transports available on the HTTP server.
Valid values	A comma-separated set where each element is one of [HTTP1_1, HTTP2].
Default value	`HTTP1_1,HTTP2`

`server.https.advertised_address`

Table 139. server.https.advertised_address

Description	Advertised address for this connector.
Valid values	A socket address in the format of `hostname:port`, `hostname`, or `:port` that is an accessible address. If missing, it is acquired from server.default_advertised_address.
Default value	`:7473`

`server.https.enabled`

Table 140. server.https.enabled

Description	Enable the HTTPS connector.
Valid values	A boolean.
Default value	`false`

`server.https.listen_address`

Table 141. server.https.listen_address

Description	Address the connector should bind to.
Valid values	A socket address in the format of `hostname:port`, `hostname`, or `:port`. If missing, it is acquired from server.default_listen_address.
Default value	`:7473`

`server.default_advertised_address`

Table 142. server.default_advertised_address

Description	Default hostname or IP address the server uses to advertise itself.
Valid values	A hostname that has no specified port and is an accessible address.
Default value	`localhost`

`server.default_listen_address`

Table 143. server.default_listen_address

Description	Default network interface to listen for incoming connections. To listen for connections on all interfaces, use "0.0.0.0".
Valid values	A hostname that has no specified port.
Default value	`localhost`

`server.discovery.advertised_address`

Enterprise Edition | Deprecated in 5.23

Table 144. server.discovery.advertised_address

Description	Advertised cluster member discovery management communication.
Valid values	A socket address in the format of `hostname:port`, `hostname`, or `:port` that is an accessible address. If missing, it is acquired from `server.default_advertised_address`.
Default value	`:5000`

`server.routing.advertised_address`

Enterprise Edition

Table 145. server.routing.advertised_address

Description	The advertised address for the intra-cluster routing connector.
Valid values	A socket address in the format of `hostname:port`, `hostname`, or `:port` that is an accessible address. If missing, it is acquired from `server.default_advertised_address`.
Default value	`:7688`

`server.routing.listen_address`

Table 146. server.routing.listen_address

Description	Address routing connector should bind to.
Valid values	A socket address in the format of `hostname:port`, `hostname`, or `:port`. If missing, it is acquired from server.default_listen_address.
Default value	`:7688`

`dbms.routing.client_side.enforce_for_domains`

Dynamic

Table 147. dbms.routing.client_side.enforce_for_domains

Description	Always use client-side routing (regardless of the default router) for `neo4j://` protocol connections to these domains. A comma-separated list of domains. Wildcards (`*`) are supported.
Valid values	A comma-separated set where each element is a string.
Default value	

dbms.routing.default_router

Table 148. dbms.routing.default_router

Description	Routing strategy for `neo4j://` protocol connections. Default is `CLIENT`, using client-side routing, with server-side routing as a fallback (if enabled). When set to `SERVER`, client-side routing is short-circuited, and requests rely on server-side routing, which must be enabled for proper operation using `dbms.routing.enabled` = `true`. Can be overridden by `dbms.routing.client_side.enforce_for_domains`.
Valid values	One of [SERVER, CLIENT].
Default value	`CLIENT`

dbms.routing.driver.connection.connect_timeout

Table 149. dbms.routing.driver.connection.connect_timeout

Description	Socket connection timeout. A timeout of zero is treated as an infinite timeout and will be bound by the timeout configured on the operating system level.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`5s`

dbms.routing.driver.connection.max_lifetime

Table 150. dbms.routing.driver.connection.max_lifetime

Description	Pooled connections older than this threshold will be closed and removed from the pool. Setting this option to a low value will cause a high connection churn and might result in a performance hit. It is recommended to set maximum lifetime to a slightly smaller value than the one configured in network equipment (load balancer, proxy, firewall, etc. can also limit maximum connection lifetime). Zero and negative values result in lifetime not being checked.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`1h`

dbms.routing.driver.connection.pool.acquisition_timeout

Table 151. dbms.routing.driver.connection.pool.acquisition_timeout

Description	Maximum amount of time spent attempting to acquire a connection from the connection pool. This timeout only kicks in when all existing connections are being used, and no new connections can be created because the maximum connection pool size has been reached. An error is raised when no connection can be acquired within the configured time. Negative values are allowed, which results in an unlimited acquisition timeout. A value of 0 is allowed, resulting in no timeout and immediate failure when the connection is unavailable.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`1m`

dbms.routing.driver.connection.pool.idle_test

Table 152. dbms.routing.driver.connection.pool.idle_test

Description	Pooled connections that have been idle in the pool for longer than this timeout will be tested to ensure they are still alive before being used again. If the value of this option is too low, acquiring a connection will require an additional network call, which will cause a performance hit. If the value of this option is too high, live connections might no longer be used, leading to errors. Hence, this parameter balances the likelihood of experiencing connection problems and performance. Usually, this parameter should not need tuning. Value 0 means connections will always be tested for validity. No connection liveliness check is done by default.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	

dbms.routing.driver.connection.pool.max_size

Table 153. dbms.routing.driver.connection.pool.max_size

Description	Maximum total number of connections to be managed by a connection pool. The limit is enforced for a combination of a host and user. Negative values are allowed and result in unlimited pool. Value of 0 is not allowed. Defaults to `-1` (unlimited).
Valid values	An integer.
Default value	`-1`

dbms.routing.driver.logging.level

Table 154. dbms.routing.driver.logging.level

Description	Sets the level for the driver's internal logging.
Valid values	One of [DEBUG, INFO, WARN, ERROR, NONE].
Default value	`INFO`

dbms.routing.enabled

Table 155. dbms.routing.enabled

Description	Enable server-side routing in clusters using an additional bolt connector. When configured, this allows requests to be forwarded from one cluster member to another, if the requests cannot be satisfied by the first member (e.g. write requests received by a non-leader).
Valid values	A boolean.
Default value	`true`

`dbms.routing.load_balancing.plugin`

Enterprise Edition

Table 156. dbms.routing.load_balancing.plugin

Description	The load balancing plugin to use.
Valid values	A string that specified load balancer plugin exist..
Default value	`server_policies`

`dbms.routing.load_balancing.shuffle_enabled`

Enterprise Edition

Table 157. dbms.routing.load_balancing.shuffle_enabled

Description	Vary the order of the entries in routing tables each time one is produced. This means that different clients should select a range of servers as their first contact, reducing the chance of all clients contacting the same server if alternatives are available. This makes the load across the servers more even.
Valid values	A boolean.
Default value	`true`

`dbms.routing.reads_on_primaries_enabled`

Enterprise Edition

Table 158. dbms.routing.reads_on_primaries_enabled

Description	Configure if the `dbms.routing.getRoutingTable()` procedure should include non-writer primaries as read endpoints or return only secondaries. NOTE: If there are no secondaries for the given database, primaries are returned as read endpoints, regardless the value of this setting. Defaults to `true` so that non-writer primaries are available for read-only queries in a typical heterogeneous setup.
Valid values	A boolean.
Default value	`true`

`dbms.routing.reads_on_writers_enabled`

Enterprise Edition | Dynamic

Table 159. dbms.routing.reads_on_writers_enabled

Description	Configure if the `dbms.routing.getRoutingTable()` procedure should include the writer as read endpoint or return only non-writers (non-writer primaries and secondaries). NOTE: Writer is returned as read endpoint if no other member is present.
Valid values	A boolean.
Default value	`false`

`dbms.routing_ttl`

Table 160. dbms.routing_ttl

Description	How long callers should cache the response of the routing procedure `dbms.routing.getRoutingTable()`.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) that is minimum `1s`.
Default value	`5m`

Cypher settings

The Cypher settings affect the behavior of Cypher queries. They can be used to tune the performance of Cypher queries or to restrict the kinds of queries that can be executed. For more information, see Statistics and execution plans.

`dbms.cypher.forbid_exhaustive_shortestpath`

Table 161. dbms.cypher.forbid_exhaustive_shortestpath

Description	This setting is associated with performance optimization. Set this to `true` in situations where it is preferable to have any queries using the 'shortestPath' function terminate as soon as possible with no answer, rather than potentially running for a long time attempting to find an answer (even if there is no path to be found). For most queries, the 'shortestPath' algorithm will return the correct answer very quickly. However there are some cases where it is possible that the fast bidirectional breadth-first search algorithm will find no results even if they exist. This can happen when the predicates in the `WHERE` clause applied to 'shortestPath' cannot be applied to each step of the traversal, and can only be applied to the entire path. When the query planner detects these special cases, it will plan to perform an exhaustive depth-first search if the fast algorithm finds no paths. However, the exhaustive search may be orders of magnitude slower than the fast algorithm. If it is critical that queries terminate as soon as possible, it is recommended that this option be set to `true`, which means that Neo4j will never consider using the exhaustive search for shortestPath queries. However, please note that if no paths are found, an error will be thrown at run time, which will need to be handled by the application.
Valid values	A boolean.
Default value	`false`

`dbms.cypher.forbid_shortestpath_common_nodes`

Table 162. dbms.cypher.forbid_shortestpath_common_nodes

Description	This setting is associated with performance optimization. The shortest path algorithm does not work when the start and end nodes are the same. With this setting set to `false` no path will be returned when that happens. The default value of `true` will instead throw an exception. This can happen if you perform a shortestPath search after a cartesian product that might have the same start and end nodes for some of the rows passed to shortestPath. If it is preferable to not experience this exception, and acceptable for results to be missing for those rows, then set this to `false`. If you cannot accept missing results, and really want the shortestPath between two common nodes, then re-write the query using a standard Cypher variable length pattern expression followed by ordering by path length and limiting to one result.
Valid values	A boolean.
Default value	`true`

`dbms.cypher.hints_error`

Table 163. dbms.cypher.hints_error

Description	Set this to specify the behavior when Cypher planner or runtime hints cannot be fulfilled. If `true`, then non-conformance will result in an error, otherwise only a warning is generated.
Valid values	A boolean.
Default value	`false`

`dbms.cypher.infer_schema_parts`

Table 164. dbms.cypher.infer_schema_parts

Description	Allow label inference during cardinality estimation. If the planner can logically deduce that a node has a label not explicitly expressed in the query, the planner will use this information during cardinality estimation. This setting controls to what extent the planner should do that: • `OFF`: No predicates are inferred. • `MOST_SELECTIVE_LABEL`: Relationship types are used to infer labels on the relationships' end nodes. The planner only infers at most one label per node. If more than one label can be inferred for a given node, the planner keeps the most selective one, the one corresponding to the smallest number of nodes in the graph.
Valid values	One of [MOST_SELECTIVE_LABEL, OFF].
Default value	`OFF`

For some queries, the planner can infer predicates such as labels or types from the graph structure that can improve estimating the number of rows that each operator produces. for more information, see Cypher Manual → Execution plans and query tuning → Understanding execution plans.
For details on how to configure this setting on a per-query basis,effectively overriding this setting on that particular query, see Cypher Manual → Query tuning → Cypher infer schema parts.

`dbms.cypher.lenient_create_relationship`

Table 165. dbms.cypher.lenient_create_relationship

Description	Set this to change the behavior for Cypher create relationship when the start or end node is missing. By default this fails the query and stops execution, but by setting this flag the create operation is simply not performed and execution continues.
Valid values	A boolean.
Default value	`false`

`dbms.cypher.min_replan_interval`

Table 166. dbms.cypher.min_replan_interval

Description	The minimum time between possible Cypher query replanning events. After this time, the graph statistics will be evaluated, and if they have changed by more than the value set by `dbms.cypher.statistics_divergence_threshold`, the query will be replanned. If the statistics have not changed sufficiently, the same interval will need to pass before the statistics will be evaluated again. Each time they are evaluated, the divergence threshold will be reduced slightly until it reaches 10% after 7h, so that even moderately changing databases will see query replanning after a sufficiently long time interval.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`10s`

`dbms.cypher.planner`

Table 167. dbms.cypher.planner

Description	Set this to specify the default planner for the default language version.
Valid values	One of [DEFAULT, COST].
Default value	`DEFAULT`

`dbms.cypher.render_plan_description`

Dynamic

Table 168. dbms.cypher.render_plan_description

Description	If set to `true` a textual representation of the plan description will be rendered on the server for all queries running with `EXPLAIN` or `PROFILE`. This allows clients such as the neo4j browser and Cypher shell to show a more detailed plan description.
Valid values	A boolean.
Default value	`true`

`dbms.cypher.statistics_divergence_threshold`

Table 169. dbms.cypher.statistics_divergence_threshold

Description	The threshold for statistics above which a plan is considered stale. If any of the underlying statistics used to create the plan have changed more than this value, the plan will be considered stale and will be replanned. Change is calculated as `abs(a-b)/max(a,b)`. This means that a value of `0.75` requires the database to quadruple in size before query replanning. A value of `0` means that the query will be replanned as soon as there is any change in statistics and the replan interval has elapsed. This interval is defined by `dbms.cypher.min_replan_interval` and defaults to 10s. After this interval, the divergence threshold will slowly start to decline, reaching 10% after about 7h. This will ensure that long running databases will still get query replanning on even modest changes, while not replanning frequently unless the changes are very large.
Valid values	A double that is in the range `0.0` to `1.0`.
Default value	`0.75`

`server.cypher.parallel.worker_limit`

Enterprise Edition | Introduced in 5.13

Table 170. server.cypher.parallel.worker_limit

Description	Number of threads to allocate to Cypher worker threads for the parallel runtime. If set to a positive number, that number of workers will be started. If set to `0`, one worker will be started for every logical processor available to the Java Virtual Machine. If set to a negative number, the total number of logical processors available on the server will be reduced by the absolute value of that number. For example, if the server has 16 available processors and you set `server.cypher.parallel.worker_limit` to `-1`, the parallel runtime will have 15 threads available.
Valid values	An integer.
Default value	`0`

Database settings

Database settings affect the behavior of a Neo4j database, for example, the file watcher service, the database format, the database store files, and the database timezone. They can be varied between each database but must be consistent across all configuration files in a cluster/DBMS.

`db.filewatcher.enabled`

Table 171. db.filewatcher.enabled

Description	Allows the enabling or disabling of the file watcher service. This is an auxiliary service but should be left enabled in almost all cases.
Valid values	A boolean.
Default value	`true`

db.format

Dynamic

Table 172. db.format

Description	Database format. This is the format that will be used for new databases. Valid values are `standard`, `aligned`, `high_limit` or `block`. The `aligned` format is essentially the `standard` format with some minimal padding at the end of pages such that a single record will never cross a page boundary. The `high_limit` and `block` formats are available for Enterprise Edition only. Either `high_limit` or `block` is required if you have a graph that is larger than 34 billion nodes, 34 billion relationships, or 68 billion properties.
Valid values	A string.
Default value	`block`

Note:

`standard` and `high_limit` formats are deprecated in Neo4j 5.23.

db.relationship_grouping_threshold

Table 173. db.relationship_grouping_threshold

Description	Relationship count threshold for considering a node to be dense. This setting applies only to `standard`, `aligned`, and `high_limit` formats.
Valid values	An integer that is minimum `1`.
Default value	`50`

db.store.files.preallocate

Table 174. db.store.files.preallocate

Description	Specify if Neo4j should try to preallocate store files as they grow.
Valid values	A boolean.
Default value	`true`

db.temporal.timezone

Table 175. db.temporal.timezone

Description	Database timezone for temporal functions. All Time and DateTime values that are created without an explicit timezone will use this configured default timezone.
Valid values	A string describing a timezone, either described by offset (e.g. `+02:00`) or by name (e.g. `Europe/Stockholm`).
Default value	`Z`

`db.track_query_cpu_time`

Enterprise Edition | Dynamic

Table 176. db.track_query_cpu_time

Description	Enables or disables tracking of how much time a query spends actively executing on the CPU. Calling `SHOW TRANSACTIONS` will display the time, but not in the *query.log*. If you want the CPU time to be logged in the *query.log*, set `db.track_query_cpu_time=true`.
Valid values	A boolean.
Default value	`false`

DBMS settings

The DBMS settings affect the Neo4j DBMS as a whole. You can use them to set the default database, the DBMS timezone, a list of seed providers, and the maximum number of databases. The DBMS settings must be consistent across all configuration files in a cluster/DBMS.

`initial.dbms.default_database`

Table 177. initial.dbms.default_database

Description	Specifies the default database name **before** the first DBMS startup. After the initial default database is created, changing this setting has no effect.
Valid values	A valid database name containing only alphabetic characters, numbers, dots, and dashes with a length between 3 and 63 characters, starting with an alphabetic character but not with the name system.
Default value	`neo4j`

Note:

The `initial.dbms.default_database` is not the same as `dbms.default_database`, which was used to set the default database in Neo4j 4.x and earlier versions.

To change the default database, use the `dbms.setDefaultDatabase()` procedure.

`dbms.db.timezone`

Table 178. dbms.db.timezone

Description	Database timezone. Among other things, this setting influences the monitoring procedures.
Valid values	One of [UTC, SYSTEM].
Default value	`UTC`

`dbms.databases.seed_from_uri_providers`

Enterprise Edition

Table 179. dbms.databases.seed_from_uri_providers

Description	Databases can be created from an existing seed (a database backup or dump) stored at a specific source URI. Different implementations of `com.neo4j.dbms.seeding.SeedProvider` support various types of seed sources. The following values are available: `S3SeedProvider`, `CloudSeedProvider`, `URLConnectionSeedProvider`, and `FileSeedProvider`. • `S3SeedProvider` supports seeds addressed with `s3`. • `CloudSeedProvider` supports seeds addressed with `s3`, `azb`, `gs`. • `URLConnectionSeedProvider` supports seeds addressed with `ftp`, `http`, and `https`. • `FileSeedProvider` supports seeds addressed with `file`. This list specifies enabled seed providers. If a seed source (URI scheme) is supported by multiple providers in the list, the first matching provider will be used. If the list is set to empty, the seed from URI functionality is effectively disabled. See Seed from a URI for more information.
Valid values	A comma-separated list where each element is a string.
Default value	`S3SeedProvider,CloudSeedProvider` Changed in 5.26

`dbms.max_databases`

Enterprise Edition

Table 180. dbms.max_databases

Description	The maximum number of databases.
Valid values	A long that is minimum `2`.
Default value	`100`

`dbms.usage_report.enabled`

Introduced in 5.20

Table 181. dbms.usage_report.enabled

Description	Usage data reporting.
Valid values	A boolean.
Default value	`true`

Import settings

The import settings control the size of the internal buffer used by `LOAD CSV` and the escaping of quotes in

CSV files.

`db.import.csv.buffer_size`

Table 182. db.import.csv.buffer_size

Description	The size of the internal buffer in bytes used by `LOAD CSV`. If the csv file contains huge fields this value may have to be increased.
Valid values	A long that is minimum `1`.
Default value	`2097152`

`db.import.csv.legacy_quote_escaping`

Table 183. db.import.csv.legacy_quote_escaping

Description	Selects whether to conform to the standard https://datatracker.ietf.org/doc/html/rfc4180 for interpreting escaped quotation characters in CSV files loaded using `LOAD CSV`. Setting this to `false` will use the standard, interpreting repeated quotes """" as a single in-lined quote, while `true` will use the legacy convention originally supported in Neo4j 3.0 and 3.1, allowing a backslash to include quotes in-lined in fields.
Valid values	A boolean.
Default value	`true`

Index settings

The index settings control the full-text index and the background index sampling (chunk size limit and sample size). For more information, see Index configuration.

`db.index.fulltext.default_analyzer`

Table 184. db.index.fulltext.default_analyzer

Description	The name of the analyzer that the full-text indexes should use by default.
Valid values	A string.
Default value	`standard-no-stop-words`

`db.index.fulltext.eventually_consistent`

Table 185. db.index.fulltext.eventually_consistent

Description	Whether or not full-text indexes should be eventually consistent by default or not.
Valid values	A boolean.
Default value	`false`

`db.index.fulltext.eventually_consistent_apply_parallelism`

Introduced in 5.21

Table 186. db.index.fulltext.eventually_consistent_apply_parallelism

Description	The number of threads processing queued index updates for eventually consistent full-text indexes.
Valid values	An integer that is minimum 1.
Default value	`1`

`db.index.fulltext.eventually_consistent_refresh_interval`

Introduced in 5.21

Table 187. db.index.fulltext.eventually_consistent_refresh_interval

Description	How often an eventually consistent full-text index is refreshed (changes are guaranteed to be visible). If set to `0`, refresh is done by the threads applying eventually consistent full-text index updates.
Valid values	A duration (Valid units are: ns, µs, ms, s, m, h and d; default unit is s).
Default value	`0s`

`db.index.fulltext.eventually_consistent_refresh_parallelism`

Introduced in 5.21

Table 188. db.index.fulltext.eventually_consistent_refresh_parallelism

Description	The number of threads that can do full-text index refresh in parallel, i.e. the number of eventually consistent full-text indexes that can be refreshed in parallel.
Valid values	An integer that is minimum 1.
Default value	`1`

`db.index.fulltext.eventually_consistent_index_update_queue_max_length`

Table 189. db.index.fulltext.eventually_consistent_index_update_queue_max_length

Description	The *eventually consistent* mode of the full-text indexes works by queueing up index updates to be applied later in a background thread. This newBuilder sets an upper bound on how many index updates are allowed to be in this queue at any one point in time. When it is reached, the commit process will slow down and wait for the index update applier thread to make some more room in the queue.
Valid values	An integer that is in the range `1` to `50000000`.
Default value	`10000`

`db.index_sampling.background_enabled`

Table 190. db.index_sampling.background_enabled

Description	Enable or disable background index sampling.
Valid values	A boolean.
Default value	`true`

`db.index_sampling.sample_size_limit`

Table 191. db.index_sampling.sample_size_limit

Description	Index sampling chunk size limit.
Valid values	An integer that is in the range `1048576` to `2147483647`.
Default value	`8388608`

`db.index_sampling.update_percentage`

Table 192. db.index_sampling.update_percentage

Description	Percentage of index updates of total index size required before sampling of a given index is triggered.
Valid values	An integer that is minimum `0`.
Default value	`5`

Logging settings

Neo4j has two different configuration files for logging, one for the *neo4j.log*, which contains general information about Neo4j, and one configuration file for all other types of logging via Log4j 2 (except *gc.log* which is handled by the Java Virtual Machine(JVM). For more information, see Logging.

`db.logs.query.annotation_data_as_json_enabled`

Introduced in 5.8 | Deprecated in 5.12 | Dynamic

Table 193. db.logs.query.annotation_data_as_json_enabled

Description	Log the annotation data as JSON strings instead of a Cypher map. This configuration has an effect only when the query log is in JSON format. From 5.9, if `true`, it collapses the nested JSON objects in the query logger.
Valid values	A boolean.
Default value	`false`
Replaced by	`db.logs.query.annotation_data_format`

`db.logs.query.annotation_data_format`

Introduced in 5.12 Dynamic

Table 194. db.logs.query.annotation_data_format

Description	The format to use for the JSON annotation data.
	
	`CYPHER`
	Formatted as a Cypher map. E.g. `{foo: 'bar', baz: {k: 1}}`.
	
	`JSON`
	Formatted as a JSON map. E.g. `{"foo": "bar", "baz": {"k": 1}}`.
	
	`FLAT_JSON`
	Formatted as a flattened JSON map. E.g. `{"foo": "bar", "baz.k": 1}`.
	
	This only have effect when the query log is in JSON format.

Valid values	One of [CYPHER, JSON, FLAT_JSON].
Default value	`CYPHER`

`db.logs.query.early_raw_logging_enabled`

Dynamic

Table 195. db.logs.query.early_raw_logging_enabled

Description	Log query text and parameters without obfuscating passwords. This allows queries to be logged earlier before parsing starts.
Valid values	A boolean.
Default value	`false`

`db.logs.query.enabled`

Dynamic

Table 196. db.logs.query.enabled

Description	Log executed queries. Valid values are `OFF`, `INFO`, or `VERBOSE`.
	
	`OFF`
	no logging.
	
	`INFO`
	log queries at the end of execution, that take longer than the configured threshold, `db.logs.query.threshold`.
	
	`VERBOSE`
	log queries at the start and end of execution, regardless of `db.logs.query.threshold`.
	
	Log entries are written to the query log.
	
	This feature is available in the Neo4j Enterprise Edition.

Valid values	One of [OFF, INFO, VERBOSE].
Default value	`VERBOSE`

`db.logs.query.max_parameter_length`

Dynamic

Table 197. db.logs.query.max_parameter_length

Description	Sets a maximum character length use for each parameter in the log. This only takes effect if `db.logs.query.parameter_logging_enabled` = `true`.
Valid values	An integer.
Default value	`2147483647`

`db.logs.query.obfuscate_errors`

Introduced in 5.26.21 Dynamic

Table 198. db.logs.query.obfuscate_errors

Description	If true, obfuscates all error information that can contain sensitive data before writing it to the query log. This applies to `failureReason` and `statusDescription` fields when the query log uses JSON format and error messages otherwise. It is recommended to set this setting to `true` in production.
Valid values	A boolean.
Default value	`false`

`db.logs.query.obfuscate_literals`

Dynamic

Table 199. db.logs.query.obfuscate_literals

Description	If true, obfuscates all literals in a query before writing the query to the query log. Note that node labels, relationship types, and map property keys remain visible. Changing the setting will not affect queries that are cached. To apply the change immediately, you must also call `CALL db.clearQueryCaches()`. It is recommended to set this setting to `true` in production.
Valid values	A boolean.
Default value	`false`

> **Note:**
>
> Keep in mind that if Neo4j receives a malformed query that cannot be parsed, it cannot obfuscate its literals (because it does not know which parts are literals) and, therefore, the query text will not be included in any logging.

`db.logs.query.parameter_logging_enabled`

Dynamic

Table 200. db.logs.query.parameter_logging_enabled

Description	Log parameters for the executed queries being logged.
Valid values	A boolean.
Default value	`true`

`db.logs.query.plan_description_enabled`

Dynamic

Table 201. db.logs.query.plan_description_enabled

Description	Log query plan description table, useful for debugging purposes.
Valid values	A boolean.
Default value	`false`

`db.logs.query.threshold`

Dynamic

Table 202. db.logs.query.threshold

Description	If the execution of a query takes more time than this threshold, the query is logged once completed - provided query logging is set to INFO. Defaults to 0 seconds, that is all queries are logged.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`0s`

db.logs.query.transaction.enabled

Dynamic | Enterprise Edition

Table 203. db.logs.query.transaction.enabled

Description	Log the start and end of a transaction. Valid values are 'OFF', 'INFO', or 'VERBOSE'. OFF: no logging. INFO: log the start and end of transactions that take longer than the configured threshold, `db.logs.query.transaction.threshold`. VERBOSE: log the start and end of all transactions. Log entries are written to the query log.
Valid values	One of [OFF, INFO, VERBOSE].
Default value	`OFF`

db.logs.query.transaction.threshold

Dynamic

Table 204. db.logs.query.transaction.threshold

Description	If the transaction is open for more time than this threshold, the transaction is logged once completed - provided transaction logging (`db.logs.query.transaction.enabled`) is set to `INFO`. Defaults to 0 seconds (all transactions are logged).
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`0s`

dbms.logs.http.enabled

Table 205. dbms.logs.http.enabled

Description	Enable HTTP request logging.
Valid values	A boolean.
Default value	`false`

server.logs.config

Table 206. server.logs.config

Description	Path to the logging configuration for debug, query, http and security logs.
Valid values	A path. If relative, it is resolved from server.directories.neo4j_home.
Default value	`conf/server-logs.xml`

server.logs.debug.enabled

Table 207. server.logs.debug.enabled

Description	Enable the debug log.
Valid values	A boolean.
Default value	`true`

`server.logs.gc.enabled`

Table 208. server.logs.gc.enabled

Description	Enable GC Logging.
Valid values	A boolean.
Default value	`false`

`server.logs.gc.options`

Table 209. server.logs.gc.options

Description	GC Logging Options.
Valid values	A string.
Default value	`-Xlog:gc*,safepoint,age*=trace`

`server.logs.gc.rotation.keep_number`

Table 210. server.logs.gc.rotation.keep_number

Description	Number of GC logs to keep.
Valid values	An integer.
Default value	`5`

`server.logs.gc.rotation.size`

Table 211. server.logs.gc.rotation.size

Description	Size of each GC log that is kept.
Valid values	A byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `T` `PiB`, `PB`, `EiB`, `EB`).
Default value	`20.00MiB`

`server.logs.user.config`

Table 212. server.logs.user.config

Description	Path to the logging configuration of user logs.
Valid values	A path. If relative, it is resolved from `server.directories.neo4j_home`.

Default value	`conf/user-logs.xml`

Memory settings

Memory settings control how much memory is allocated to Neo4j and how it is used. It is recommended to perform a certain amount of testing and tuning of these settings to figure out the optimal division of the available memory. For more information on how to tune these settings, see Memory configuration, Disks, RAM and other tips, and Tuning of the garbage collector.

`db.memory.pagecache.warmup.enable`

Table 213. db.memory.pagecache.warmup.enable

Description	Page cache can be configured to perform usage sampling of loaded pages that can be used to construct active load profile. According to that profile pages can be reloaded on the restart, replication, etc. This setting allows disabling that behavior. This feature is available in Neo4j Enterprise Edition.
Valid values	A boolean.
Default value	`true`

`db.memory.pagecache.warmup.preload`

Table 214. db.memory.pagecache.warmup.preload

Description	Page cache warmup can be configured to prefetch files, preferably when cache size is bigger than store size. Files to be prefetched can be filtered by 'dbms.memory.pagecache.warmup.preload.allowlist'. Enabling this disables warmup by profile.
Valid values	A boolean.
Default value	`false`

`db.memory.pagecache.warmup.preload.allowlist`

Table 215. db.memory.pagecache.warmup.preload.allowlist

Description	Page cache warmup prefetch file allowlist regex. By default matches all files.
Valid values	A string.
Default value	`.*`

`db.memory.pagecache.warmup.profile.interval`

Enterprise Edition

Table 216. db.memory.pagecache.warmup.profile.interval

Description	The profiling frequency for the page cache. Accurate profiles allow the page cache to do an active warmup after a restart, reducing the mean time to performance.

Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`1m`

db.memory.transaction.max

Dynamic

Table 217. db.memory.transaction.max

Description	Limit the amount of memory that a single transaction can consume, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm', and gigabytes with 'g'). Zero means 'largest possible value'.
Valid values	A byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `T` `PiB`, `PB`, `EiB`, `EB`) that is minimum `1.00MiB` or is `0B`.
Default value	`0B`

db.memory.transaction.total.max

Dynamic

Table 218. db.memory.transaction.total.max

Description	Limit the amount of memory that all transactions in one database can consume, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g'). Zero means 'unlimited'.
Valid values	A byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `T` `PiB`, `PB`, `EiB`, `EB`) that is minimum `10.00MiB` or is `0B`.
Default value	`0B`

db.tx_state.memory_allocation

Deprecated in 5.8

Table 219. db.tx_state.memory_allocation

Description	Defines whether memory for transaction state should be allocated on- or off-heap. Note that for small transactions you can gain up to 25% write speed by setting it to `ON_HEAP`.
Valid values	One of [ON_HEAP, OFF_HEAP].
Default value	`ON_HEAP`

server.db.query_cache_size

Deprecated in 5.7

Table 220. server.db.query_cache_size

Description	The number of cached Cypher query execution plans per database. The max number of query plans that can be kept in cache is the `number of databases` * `server.db.query_cache_size`. With 10 databases and `server.db.query_cache_size` =1000, the caches can keep 10000 plans in total on the instance, assuming that each DB receives queries that fill up its cache.
Valid values	An integer that is minimum `0`.
Default value	`1000`
Replaced by	`server.memory.query_cache.per_db_cache_num_entries`

dbms.memory.tracking.enable

Table 221. dbms.memory.tracking.enable

Description	Enable off heap and on heap memory tracking. Should not be set to `false` for clusters.
Valid values	A boolean.
Default value	`true`

dbms.memory.transaction.total.max

Changed in 5.2 Dynamic

Table 222. dbms.memory.transaction.total.max

Description	Limit the amount of memory that all of the running transactions can consume, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g'). Zero means 'unlimited'. Defaults to 70% of the heap size limit.
Valid values	A byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `T` `PiB`, `PB`, `EiB`, `EB`) that is minimum `10.00MiB` or is `0B`.
Default value	

server.memory.heap.initial_size

Table 223. server.memory.heap.initial_size

Description	Initial heap size. By default it is calculated based on available system resources.
Valid values	A byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `T` `PiB`, `PB`, `EiB`, `EB`).
Default value	

server.memory.heap.max_size

Table 224. server.memory.heap.max_size

Description	Maximum heap size. By default it is calculated based on available system resources.

Valid values	A byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `TiB`, `TB`, `T`, `tB`, `tb`, `t`, `PiB`, `PB`, `EiB`, `EB`).
Default value	

`server.memory.off_heap.block_cache_size`

Deprecated in 5.26

Table 225. server.memory.off_heap.block_cache_size

Description	Defines the size of the off-heap memory blocks cache. The cache will contain this number of blocks for each block size that is power of two. Thus, maximum amount of memory used by blocks cache can be calculated as 2 * server.memory.off_heap.max_cacheable_block_size * `server.memory.off_heap.block_cache_size`
Valid values	An integer that is minimum `16`.
Default value	`128`

`server.memory.off_heap.max_cacheable_block_size`

Deprecated in 5.26

Table 226. server.memory.off_heap.max_cacheable_block_size

Description	Defines the maximum size of an off-heap memory block that can be cached to speed up allocations. The value must be a power of 2.
Valid values	A byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `TiB`, `TB`, `T`, `tB`, `tb`, `t`, `PiB`, `PB`, `EiB`, `EB`) that is minimum `4.00KiB` and is power of 2.
Default value	`512.00KiB`

`server.memory.off_heap.transaction_max_size`

Deprecated in 5.8

Table 227. server.memory.off_heap.transaction_max_size

Description	The maximum amount of off-heap memory that can be used to store transaction state data; it's a total amount of memory shared across all active transactions. Zero means 'unlimited'. Used when db.tx_state.memory_allocation is set to 'OFF_HEAP'.
Valid values	A byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `TiB`, `TB`, `T`, `tB`, `tb`, `t`, `PiB`, `PB`, `EiB`, `EB`) that is minimum `0B`.
Default value	`2.00GiB`

`server.memory.pagecache.directio`

Table 228. server.memory.pagecache.directio

Description	Use direct I/O for page cache. This setting is supported only on Linux and only for a subset of record formats that use platform-aligned page size.
Valid values	A boolean.
Default value	`false`

`server.memory.pagecache.flush.buffer.enabled`

Dynamic

Table 229. server.memory.pagecache.flush.buffer.enabled

Description	Page cache can be configured to use a temporal buffer for flushing purposes. It is used to combine, if possible, sequence of several cache pages into one bigger buffer to minimize the number of individual IOPS performed and better utilization of available I/O resources, especially when those are restricted.
Valid values	A boolean.
Default value	`false`

`server.memory.pagecache.flush.buffer.size_in_pages`

Dynamic

Table 230. server.memory.pagecache.flush.buffer.size_in_pages

Description	Page cache can be configured to use a temporal buffer for flushing purposes. It is used to combine, if possible, sequence of several cache pages into one bigger buffer to minimize the number of individual IOPS performed and better utilization of available I/O resources, especially when those are restricted. Use this setting to configure individual file flush the buffer size in pages (8KiB). To be able to utilize this buffer during page cache flushing, buffered flush should be enabled.
Valid values	An integer that is in the range `1` to `512`.
Default value	`128`

`server.memory.pagecache.scan.prefetchers`

Table 231. server.memory.pagecache.scan.prefetchers

Description	The maximum number of worker threads to use for pre-fetching data when doing sequential scans. Set to '0' to disable pre-fetching for scans.
Valid values	An integer that is in the range `0` to `255`.
Default value	`4`

`server.memory.pagecache.size`

Table 232. server.memory.pagecache.size

Description	The amount of memory to use for mapping the store files. If Neo4j is running on a dedicated server, then it is generally recommended to leave about 2-4 gigabytes for the operating system, give the JVM enough heap to hold all your transaction state and query context, and then leave the rest for the page cache. If no page cache memory is configured, then a heuristic setting is computed based on available system resources. By default the size of page cache will be 50% of available RAM minus the max heap size. The size of the page cache will also not be larger than 70x the max heap size (due to some overhead of the page cache in the heap.
Valid values	A byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `T` `PiB`, `PB`, `EiB`, `EB`).
Default value	

`server.memory.query_cache.sharing_enabled`

Introduced in 5.8 | Enterprise Edition

Table 233. server.memory.query_cache.sharing_enabled

Description	Enable sharing cache space between different databases. With this option turned on, databases will share cache space, but not cache entries. This means that a database may store and retrieve entries from the shared cache, but it may not retrieve entries produced by another database. The database may, however, evict entries from other databases as necessary, according to the constrained cache size and cache eviction policy. In essence, databases may compete for cache space, but may not observe each other's entries. When this option is turned on, the cache space available to all databases is configured with `server.memory.query_cache.shared_cache_num_entries`. With this option turned off, the cache space available to each individual database is configured with `server.memory.query_cache.per_db_cache_num_entries`.
Valid values	A boolean.
Default value	`false`

`server.memory.query_cache.shared_cache_num_entries`

Introduced in 5.7 | Dynamic since 5.10 | Enterprise Edition

Table 234. server.memory.query_cache.shared_cache_num_entries

Description	The number of cached queries for all databases. The maximum number of queries that can be kept in a cache is exactly `server.memory.query_cache.shared_cache_num_entries`. This setting is only deciding cache size when `server.memory.query_cache.sharing_enabled` is set to `true`.
Valid values	An integer that is minimum `0`.
Default value	`1000`

`server.memory.query_cache.per_db_cache_num_entries`

Dynamic since 5.10

Table 235. server.memory.query_cache.per_db_cache_num_entries

Description	The number of cached queries per database. The maximum number of queries that can be kept in a cache is `number of databases` * `server.memory.query_cache.per_db_cache_num_entries`. With 10 databases and `server.memory.query_cache.per_db_cache_num_entries`=1000, the cache can keep 10000 plans in total. This setting is only deciding cache size when `server.memory.query_cache.sharing_enabled` is set to `false`.
Valid values	An integer that is minimum `0`.
Default value	`1000`

Metrics settings

The metrics settings control whether Neo4j will log metrics, what metrics to log, how to log them, and how to expose them. For better understanding of the metrics settings and how to configure them, see Metrics.

`server.metrics.csv.enabled`

Enterprise Edition

Table 236. server.metrics.csv.enabled

Description	Set to `true` to enable exporting metrics to CSV files.
Valid values	A boolean.
Default value	`true`

`server.metrics.csv.interval`

Enterprise Edition

Table 237. server.metrics.csv.interval

Description	The reporting interval for the CSV files. That is, how often new rows with numbers are appended to the CSV files.
Valid values	A duration (Valid units are: `ns`, `µ s`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) that is minimum `1ms`.
Default value	`30s`

`server.metrics.csv.rotation.compression`

Enterprise Edition

Table 238. server.metrics.csv.rotation.compression

Description	Decides what compression to use for the csv history files.
Valid values	One of [NONE, ZIP, GZ].

Default value	NONE

server.metrics.csv.rotation.keep_number

Enterprise Edition

Table 239. server.metrics.csv.rotation.keep_number

Description	Maximum number of history files for the csv files.
Valid values	An integer that is minimum 1.
Default value	7

server.metrics.csv.rotation.size

Enterprise Edition

Table 240. server.metrics.csv.rotation.size

Description	The file size in bytes at which the csv files will auto-rotate. If set to zero then no rotation will occur. Accepts a binary suffix k, m or g.
Valid values	A byte size (valid multipliers are B, KiB, KB, K, kB, kb, k, MiB, MB, M, mB, mb, m, GiB, GB, G, gB, gb, g, T PiB, PB, EiB, EB) that is in the range 0B to 8388608.00TiB.
Default value	10.00MiB

server.metrics.enabled

Enterprise Edition

Table 241. server.metrics.enabled

Description	Enable metrics. Setting this to false will to turn off all metrics.
Valid values	A boolean.
Default value	true

server.metrics.filter

Enterprise Edition

Table 242. server.metrics.filter

Description	Specifies which metrics should be enabled by using a comma separated list of globbing patterns. Only the metrics matching the filter will be enabled. For example *check_point*,neo4j.page_cache.evictions will enable any checkpoint metrics and the pagecache eviction metric.
Valid values	A comma-separated list where each element is A simple globbing pattern that can use * and ?..

Default value	`*bolt.connections*,*bolt.messages_received*,*bolt.messages_started*,*dbms.pool.bolt.free,*dbms.pool.bolt.total_size,*dbms.pool.bolt.total_used,*dbms.pool.bolt.used_heap,*cluster.raft.is_leader,*cluster.raft.last_leader_message,*cluster.raft.replication_attempt,*cluster.raft.replication_fail,*cluster.raft.last_applied,*cluster.raft.last_appended,*cluster.raft.append_index,*cluster.raft.commit_index,*cluster.raft.applied_index,*cluster.internal.discovery.memberset.left,*cluster.internal.discovery.crdt.gossip_id_data.size,*cluster.internal.discovery.crdt.server_data.size,*cluster.internal.discovery.crdt.database_data.size,*cluster.internal.discovery.crdt.leader_data.size,*cluster.internal.discovery.crdt.total_merge_operations,*cluster.internal.discovery.crdt.total_update_operations,*cluster.internal.discovery.gossip.incoming_queue_size,*cluster.internal.discovery.gossip.total_received_data,*cluster.internal.discovery.gossip.total_sent_data,*cluster.internal.discovery.gossip.uncontactable_members_exist,*check_point.*,*cypher.replan_events,*cypher.cache*,*ids_in_use*,*pool.transaction.*.total_used,*pool.transaction.*.used_heap,*pool.transaction.*.used_native,*store.size*,*transaction.active_read,*transaction.active_write,*transaction.committed*,*transaction.last_committed_tx_id,*transaction.peak_concurrent,*transaction.rollbacks*,*page_cache.hit*,*page_cache.page_faults,*page_cache.usage_ratio,*vm.file.descriptors.count,*vm.gc.time.*,*vm.heap.used,*vm.memory.buffer.direct.used,*vm.memory.pool.g1_eden_space,*vm.memory.pool.g1_old_gen,*vm.pause_time,*vm.thread*,*db.query.execution*,*protocol*`

`server.metrics.graphite.enabled`

Enterprise Edition

Table 243. server.metrics.graphite.enabled

Description	Set to `true` to enable exporting metrics to Graphite.
Valid values	A boolean.
Default value	`false`

`server.metrics.graphite.interval`

Enterprise Edition

Table 244. server.metrics.graphite.interval

Description	The reporting interval for Graphite. That is, how often to send updated metrics to Graphite.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`30s`

`server.metrics.graphite.server`

Enterprise Edition

Table 245. server.metrics.graphite.server

Description	The hostname or IP address of the Graphite server.

Valid values	A socket address in the format of `hostname:port`, `hostname`, or `:port`. If missing, it is acquired from server.default_listen_address.
Default value	`:2003`

`server.metrics.jmx.enabled`

Enterprise Edition

Table 246. server.metrics.jmx.enabled

Description	Set to `true` to enable the JMX metrics endpoint.
Valid values	A boolean.
Default value	`true`

`server.metrics.prefix`

Enterprise Edition

Table 247. server.metrics.prefix

Description	A common prefix for the reported metrics field names.
Valid values	A string.
Default value	`neo4j`

`server.metrics.prometheus.enabled`

Enterprise Edition

Table 248. server.metrics.prometheus.enabled

Description	Set to `true` to enable the Prometheus endpoint.
Valid values	A boolean.
Default value	`false`

`server.metrics.prometheus.endpoint`

Enterprise Edition

Table 249. server.metrics.prometheus.endpoint

Description	The hostname and port to use as Prometheus endpoint.
Valid values	A socket address in the format of `hostname:port`, `hostname`, or `:port`. If missing, it is acquired from `server.default_listen_address`.

Default value	`localhost:2004`

Neo4j Browser and client settings

Neo4j Browser and client settings apply only to Neo4j Browser and the client.

`browser.allow_outgoing_connections`

Enterprise Edition

Table 250. browser.allow_outgoing_connections

Description	Configure the policy for outgoing Neo4j Browser connections.
Valid values	A boolean.
Default value	`true`

`browser.credential_timeout`

Enterprise Edition

Table 251. browser.credential_timeout

Description	Configure the Neo4j Browser to time out logged in users after this idle period. Setting this to 0 indicates no limit.
Valid values	A duration (Valid units are: ns, µs, ms, s, m, h and d; default unit is s).
Default value	`0s`

`browser.post_connect_cmd`

Table 252. browser.post_connect_cmd

Description	Commands to be run when Neo4j Browser successfully connects to this server. Separate multiple commands with semi-colon.
Valid values	A string.
Default value	

`browser.remote_content_hostname_whitelist`

Table 253. browser.remote_content_hostname_whitelist

Description	Whitelist of hosts for the Neo4j Browser to be allowed to fetch content from.
Valid values	A string.
Default value	`guides.neo4j.com,localhost`

`browser.retain_connection_credentials`

Enterprise Edition

Table 254. browser.retain_connection_credentials

Description	Configure the Neo4j Browser to store or not store user credentials.
Valid values	A boolean.
Default value	`true`

`browser.retain_editor_history`

Enterprise Edition

Table 255. browser.retain_editor_history

Description	Configure the Neo4j Browser to store or not store user editor history.
Valid values	A boolean.
Default value	`true`

`client.allow_telemetry`

Table 256. client.allow_telemetry

Description	Configure client applications such as Browser and Bloom to send Product Analytics data.
Valid values	A boolean.
Default value	`true`

Kubernetes settings

The Kubernetes settings are used to configure a cluster running on Kubernetes, where each server is running as a Kubernetes service. The addresses of the other servers can be obtained using the List Service API, as described in the Kubernetes API documentation. For more information, see Discovery in Kubernetes.

`dbms.kubernetes.address`

Kubernetes settings

Enterprise Edition

Table 257. dbms.kubernetes.address

Description	Address for Kubernetes API.
Valid values	A socket address in the format of `hostname:port`, `hostname`, or `:port`.

Default value	`kubernetes.default.svc:443`

`dbms.kubernetes.ca_crt`

Enterprise Edition

Table 258. dbms.kubernetes.ca_crt

Description	File location of CA certificate for Kubernetes API.
Valid values	A path.
Default value	`/var/run/secrets/kubernetes.io/serviceaccount/ca.crt`

`dbms.kubernetes.cluster_domain`

Enterprise Edition

Table 259. dbms.kubernetes.cluster_domain

Description	Kubernetes cluster domain.
Valid values	A string.
Default value	`cluster.local`

`dbms.kubernetes.label_selector`

Enterprise Edition

Table 260. dbms.kubernetes.label_selector

Description	LabelSelector for Kubernetes API.
Valid values	A string.
Default value	

`dbms.kubernetes.namespace`

Enterprise Edition

Table 261. dbms.kubernetes.namespace

Description	File location of namespace for Kubernetes API.
Valid values	A path.
Default value	`/var/run/secrets/kubernetes.io/serviceaccount/namespace`

`dbms.kubernetes.service_port_name`

Enterprise Edition Deprecated in 5.23

Table 262. dbms.kubernetes.service_port_name

Description	Service port name for discovery for Kubernetes API.
Valid values	A string.
Default value	

`dbms.kubernetes.discovery.v2.service_port_name`

Enterprise Edition Introduced in 5.22

Table 263. dbms.kubernetes.discovery.v2.service_port_name

Description	Service port name for Discovery v2 for Kubernetes API.
Valid values	A string.
Default value	`transaction`

`dbms.kubernetes.token`

Enterprise Edition

Table 264. dbms.kubernetes.token

Description	File location of token for Kubernetes API.
Valid values	A path.
Default value	`/var/run/secrets/kubernetes.io/serviceaccount/token`

Security settings

The security settings are used to configure the security of your Neo4j deployment. Refer to the Security section for thorough information on security in Neo4j.

`dbms.security.allow_csv_import_from_file_urls`

Table 265. dbms.security.allow_csv_import_from_file_urls

Description	Determines if Cypher will allow using file URLs when loading data using `LOAD CSV`. Setting this value to `false` will cause Neo4j to fail `LOAD CSV` clauses that load data from the file system.
Valid values	A boolean.
Default value	`true`

`dbms.security.auth_cache_max_capacity`

Enterprise Edition

Table 266. dbms.security.auth_cache_max_capacity

Description	The maximum capacity for authentication and authorization caches (respectively).
Valid values	An integer.
Default value	`10000`

`dbms.security.auth_cache_ttl`

Enterprise Edition

Table 267. dbms.security.auth_cache_ttl

Description	The time to live (TTL) for cached authentication and authorization info when using external auth providers (OIDC, LDAP or plugin). Setting the TTL to 0 will disable auth caching. Disabling caching while using the LDAP auth provider requires the use of an LDAP system account for resolving authorization information.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`10m`

`dbms.security.auth_cache_use_ttl`

Enterprise Edition

Table 268. dbms.security.auth_cache_use_ttl

Description	Enable time-based eviction of the authentication and authorization info cache for external auth providers (OIDC, LDAP or plugin). Disabling this setting will make the cache live forever and only be evicted when `dbms.security.auth_cache_max_capacity` is exceeded.
Valid values	A boolean.
Default value	`true`

`dbms.security.auth_enabled`

Table 269. dbms.security.auth_enabled

Description	Enable auth requirement to access Neo4j.
Valid values	A boolean.
Default value	`true`

`dbms.security.auth_minimum_password_length`

Introduced in 5.3

Table 270. dbms.security.auth_minimum_password_length

Description	The minimum number of characters required in a password.
Valid values	An integer that is minimum `1`.
Default value	`8`

`dbms.security.auth_lock_time`

Table 271. dbms.security.auth_lock_time

Description	The amount of time user account should be locked after a configured number of unsuccessful authentication attempts. The locked out user will not be able to log in until the lock period expires, even if correct credentials are provided. Setting this configuration option to a low value is not recommended because it might make it easier for an attacker to brute force the password.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) that is minimum `0s`.
Default value	`5s`

`dbms.security.auth_max_failed_attempts`

Table 272. dbms.security.auth_max_failed_attempts

Description	The maximum number of unsuccessful authentication attempts before imposing a user lock for the configured amount of time, as defined by `dbms.security.auth_lock_time`.The locked out user will not be able to log in until the lock period expires, even if correct credentials are provided. Setting this configuration option to values less than 3 is not recommended because it might make it easier for an attacker to brute force the password.
Valid values	An integer that is minimum `0`.
Default value	`3`

`dbms.security.authentication_providers`

Enterprise Edition

Table 273. dbms.security.authentication_providers

Description	A list of security authentication providers containing the users and roles. This can be any of the built-in `native` or `ldap` providers, or it can be an externally provided plugin, with a custom name prefixed by `plugin-`, i.e. `plugin-<AUTH_PROVIDER_NAME>`. They will be queried in the given order when login is attempted.
Valid values	A comma-separated list where each element is a string.
Default value	`native`

`dbms.security.authorization_providers`

Enterprise Edition

Table 274. dbms.security.authorization_providers

Description	A list of security authorization providers containing the users and roles. This can be any of the built-in `native` or `ldap` providers, or it can be an externally provided plugin, with a custom name prefixed by `plugin-`, i.e. `plugin-<AUTH_PROVIDER_NAME>`. They will be queried in the given order when login is attempted.
Valid values	A comma-separated list where each element is a string.
Default value	`native`

dbms.security.cluster_status_auth_enabled

Enterprise Edition

Table 275. dbms.security.cluster_status_auth_enabled

Description	Require authorization for access to the Causal Clustering status endpoints.
Valid values	A boolean.
Default value	`true`

dbms.security.http_access_control_allow_origin

Table 276. dbms.security.http_access_control_allow_origin

Description	Value of the Access-Control-Allow-Origin header sent over any HTTP or HTTPS connector. This defaults to '*', which allows broadest compatibility. Note that any URI provided here limits HTTP/HTTPS access to that URI only.
Valid values	A string.
Default value	*

dbms.security.http_auth_allowlist

Table 277. dbms.security.http_auth_allowlist

Description	Defines an allowlist of http paths where Neo4j authentication is not required.
Valid values	A comma-separated list where each element is a string.
Default value	`/,/browser.*`

dbms.security.http_strict_transport_security

Table 278. dbms.security.http_strict_transport_security

Description	Value of the HTTP Strict-Transport-Security (HSTS) response header. This header tells browsers that a webpage should only be accessed using HTTPS instead of HTTP. It is attached to every HTTPS response. Setting is not set by default so 'Strict-Transport-Security' header is not sent. Value is expected to contain directives like 'max-age', 'includeSubDomains' and 'preload'.

Valid values	A string.
Default value	

dbms.security.http_static_content_security_policy_header

Introduced in 5.20

Table 279. dbms.security.http_static_content_security_policy_header

Description	Defines the Content-Security-Policy header to return to content returned on static endpoints.
Valid values	A string.
Default value	`default-src 'self'; script-src 'self' cdn.segment.com canny.io; img-src 'self' guides.neo4j.com data:; style-src 'self' fonts.googleapis.com 'unsafe-inline'; font-src 'self' fonts.gstatic.com; base-uri 'none'; object-src 'none'; frame-ancestors 'none'; connect-src 'self' api.canny.io api.segment.io ws: wss: http: https:`

dbms.security.key.name

Enterprise Edition Dynamic

Table 280. dbms.security.key.name

Description	Name of the 256 length AES encryption key, which is used for the symmetric encryption.
Valid values	A string.
Default value	`aesKey`

dbms.security.keystore.password

Enterprise Edition Dynamic

Table 281. dbms.security.keystore.password

Description	Password for accessing the keystore holding a 256 length AES encryption key, which is used for the symmetric encryption.
Valid values	A secure string.
Default value	

dbms.security.keystore.path

Enterprise Edition Dynamic

Table 282. dbms.security.keystore.path

Description	Location of the keystore holding a 256 length AES encryption key, which is used for the symmetric encryption of secrets held in system database.
Valid values	A path.
Default value	

dbms.security.ldap.authentication.attribute

Enterprise Edition | Dynamic

Table 283. dbms.security.ldap.authentication.attribute

Description	The attribute to use when looking up users. Using this setting requires `dbms.security.ldap.authentication.search_for_attribute` to be `true` and thus `dbms.security.ldap.authorization.system_username` and `dbms.security.ldap.authorization.system_password` to be configured.
Valid values	A string that matches the pattern `[A-Za-z0-9-]*` (has to be a valid LDAP attribute name, only containing letters [A-Za-z], digits [0-9] and hyphens [-].).
Default value	`samaccountname`

dbms.security.ldap.authentication.cache_enabled

Enterprise Edition

Table 284. dbms.security.ldap.authentication.cache_enabled

Description	Determines if the result of authentication via the LDAP server should be cached or not. Caching is used to limit the number of LDAP requests that have to be made over the network for users that have already been authenticated successfully. A user can be authenticated against an existing cache entry (instead of via an LDAP server) as long as it is alive (see `dbms.security.auth_cache_ttl`). An important consequence of setting this to `true` is that Neo4j then needs to cache a hashed version of the credentials in order to perform credentials matching. This hashing is done using a cryptographic hash function together with a random salt. Preferably a conscious decision should be made if this method is considered acceptable by the security standards of the organization in that this Neo4j instance is deployed.
Valid values	A boolean.
Default value	`true`

dbms.security.ldap.authentication.mechanism

Enterprise Edition

Table 285. dbms.security.ldap.authentication.mechanism

Description	LDAP authentication mechanism. This is one of `simple` or a SASL mechanism supported by JNDI, for example `DIGEST-MD5`. `simple` is basic username and password authentication and SASL is used for more advanced mechanisms. See RFC 2251 LDAPv3 documentation for more details.

Valid values	A string.
Default value	`simple`

`dbms.security.ldap.authentication.search_for_attribute`

Enterprise Edition

Table 286. dbms.security.ldap.authentication.search_for_attribute

Description	Perform authentication by searching for an unique attribute of a user. Using this setting requires `dbms.security.ldap.authorization.system_username` and `dbms.security.ldap.authorization.system_password` to be configured.
Valid values	A boolean.
Default value	`false`

`dbms.security.ldap.authentication.user_dn_template`

Enterprise Edition Dynamic

Table 287. dbms.security.ldap.authentication.user_dn_template

Description	LDAP user DN template. An LDAP object is referenced by its distinguished name (DN), and a user DN is an LDAP fully-qualified unique user identifier. This setting is used to generate an LDAP DN that conforms with the LDAP directory's schema from the user principal that is submitted with the authentication token when logging in. The special token \{0} is a placeholder where the user principal will be substituted into the DN string.
Valid values	A string that Must be a string containing '\{0}' to understand where to insert the runtime authentication principal..
Default value	`uid={0},ou=users,dc=example,dc=com`

`dbms.security.ldap.authorization.access_permitted_group`

Enterprise Edition Dynamic

Table 288. dbms.security.ldap.authorization.access_permitted_group

Description	The LDAP group to which a user must belong to get any access to the system.Set this to restrict access to a subset of LDAP users belonging to a particular group. If this is not set, any user to successfully authenticate via LDAP will have access to the PUBLIC role and any other roles assigned to them via `dbms.security.ldap.authorization.group_to_role_mapping`.
Valid values	A string.
Default value	

`dbms.security.ldap.authorization.group_membership_attributes`

Enterprise Edition Dynamic

Table 289. dbms.security.ldap.authorization.group_membership_attributes

Description	A list of attribute names on a user object that contains groups to be used for mapping to roles when LDAP authorization is enabled. This setting is ignored when `dbms.ldap_authorization_nested_groups_enabled` is `true`.
Valid values	A comma-separated list where each element is a string, which Can not be empty.
Default value	`memberOf`

dbms.security.ldap.authorization.group_to_role_mapping

Enterprise Edition Dynamic

Table 290. dbms.security.ldap.authorization.group_to_role_mapping

Description	An authorization mapping from LDAP group names to Neo4j role names. The map should be formatted as a semicolon separated list of key-value pairs, where the key is the LDAP group name and the value is a comma separated list of corresponding role names. For example: group1=role1;group2=role2;group3=role3,role4,role5 You could also use whitespaces and quotes around group names to make this mapping more readable, for example: `dbms.security.ldap.authorization.group_to_role_mapping`=\ "cn=Neo4j Read Only,cn=users,dc=example,dc=com" = reader; \ "cn=Neo4j Read-Write,cn=users,dc=example,dc=com" = publisher; \ "cn=Neo4j Schema Manager,cn=users,dc=example,dc=com" = architect; \ "cn=Neo4j Administrator,cn=users,dc=example,dc=com" = admin
Valid values	A string that must be a semicolon-separated list of key-value pairs or empty.
Default value	

dbms.security.ldap.authorization.nested_groups_enabled

Enterprise Edition Dynamic

Table 291. dbms.security.ldap.authorization.nested_groups_enabled

Description	This setting determines whether multiple LDAP search results will be processed (as is required for the lookup of nested groups). If set to `true` then instead of using attributes on the user object to determine group membership (as specified by `dbms.security.ldap.authorization.group_membership_attributes`), the `user` object will only be used to determine the user's Distinguished Name, which will subsequently be used with `dbms.security.ldap.authorization.user_search_filter` in order to perform a nested group search. The Distinguished Names of the resultant group search results will be used to determine roles.
Valid values	A boolean.
Default value	`false`

dbms.security.ldap.authorization.nested_groups_search_filter

Enterprise Edition Dynamic

Table 292. dbms.security.ldap.authorization.nested_groups_search_filter

Description	The search template which will be used to find the nested groups which the user is a member of. The filter should contain the placeholder token `\{0}` which will be substituted with the user's Distinguished Name (which is found for the specified user principle using `dbms.security.ldap.authorization.user_search_filter`). The default value specifies Active Directory's LDAP_MATCHING_RULE_IN_CHAIN (aka 1.2.840.113556.1.4.1941) implementation which will walk the ancestry of group membership for the specified user.
Valid values	A string.
Default value	`(&(objectclass=group)(member:1.2.840.113556.1.4.1941:={0}))`

dbms.security.ldap.authorization.system_password

Enterprise Edition

Table 293. dbms.security.ldap.authorization.system_password

Description	An LDAP system account password to use for authorization searches when `dbms.security.ldap.authorization.use_system_account` is `true`.
Valid values	A secure string.
Default value	

dbms.security.ldap.authorization.system_username

Enterprise Edition

Table 294. dbms.security.ldap.authorization.system_username

Description	An LDAP system account username to use for authorization searches when `dbms.security.ldap.authorization.use_system_account` is `true`. Note that the `dbms.security.ldap.authentication.user_dn_template` will not be applied to this username, so you may have to specify a full DN.
Valid values	A string.
Default value	

dbms.security.ldap.authorization.use_system_account

Enterprise Edition

Table 295. dbms.security.ldap.authorization.use_system_account

Description	Perform LDAP search for authorization info using a system account instead of the user's own account. If this is set to `false` (default), the search for group membership will be performed directly after authentication using the LDAP context bound with the user's own account. The mapped roles will be cached for the duration of `dbms.security.auth_cache_ttl`, and then expire, requiring re-authentication. To avoid frequently having to re-authenticate sessions you may want to set a relatively long auth cache expiration time together with this option. NOTE: This option will only work if the users are permitted to search for their own group membership attributes in the directory. If this is set to `true`, the search will be performed using a special system account user with read access to all the users in the directory. You need to specify the username and password using the settings `dbms.security.ldap.authorization.system_username` and `dbms.security.ldap.authorization.system_password` with this option. Note that this account only needs read access to the relevant parts of the LDAP directory and does not need to have access rights to Neo4j, or any other systems.
Valid values	A boolean.
Default value	`false`

`dbms.security.ldap.authorization.user_search_base`

Enterprise Edition | Dynamic

Table 296. dbms.security.ldap.authorization.user_search_base

Description	The name of the base object or named context to search for user objects when LDAP authorization is enabled. A common case is that this matches the last part of `dbms.security.ldap.authentication.user_dn_template`.
Valid values	A string that Can not be empty.
Default value	`ou=users,dc=example,dc=com`

`dbms.security.ldap.authorization.user_search_filter`

Enterprise Edition | Dynamic

Table 297. dbms.security.ldap.authorization.user_search_filter

Description	The LDAP search filter to search for a user principal when LDAP authorization is enabled. The filter should contain the placeholder token \{0} which will be substituted for the user principal.
Valid values	A string.
Default value	`(&(objectClass=*)(uid={0}))`

`dbms.security.ldap.connection_timeout`

Enterprise Edition

Table 298. dbms.security.ldap.connection_timeout

Description	The timeout for establishing an LDAP connection. If a connection with the LDAP server cannot be established within the given time the attempt is aborted. A value of 0 means to use the network protocol's (i.e., TCP's) timeout value.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`30s`

`dbms.security.ldap.host`

Enterprise Edition

Table 299. dbms.security.ldap.host

Description	URL of LDAP server to use for authentication and authorization. The format of the setting is `<protocol>://<hostname>:<port>`, where hostname is the only required field. The supported values for protocol are `ldap` (default) and `ldaps`. The default port for `ldap` is 389 and for `ldaps` 636. For example: `ldaps://ldap.example.com:10389`. You may want to consider using STARTTLS (`dbms.security.ldap.use_starttls`) instead of LDAPS for secure connections, in which case the correct protocol is `ldap`.
Valid values	A string.
Default value	`localhost`

`dbms.security.ldap.read_timeout`

Enterprise Edition

Table 300. dbms.security.ldap.read_timeout

Description	The timeout for an LDAP read request (i.e. search). If the LDAP server does not respond within the given time the request will be aborted. A value of 0 means wait for a response indefinitely.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`30s`

`dbms.security.ldap.referral`

Enterprise Edition

Table 301. dbms.security.ldap.referral

Description	The LDAP referral behavior when creating a connection. This is one of `follow`, `ignore` or `throw`. • `follow` automatically follows any referrals • `ignore` ignores any referrals • `throw` throws an exception, which will lead to authentication failure.
Valid values	A string.
Default value	`follow`

dbms.security.ldap.use_starttls

Enterprise Edition

Table 302. dbms.security.ldap.use_starttls

Description	Use secure communication with the LDAP server using opportunistic TLS. First an initial insecure connection will be made with the LDAP server, and a STARTTLS command will be issued to negotiate an upgrade of the connection to TLS before initiating authentication.
Valid values	A boolean.
Default value	`false`

dbms.security.log_successful_authentication

Enterprise Edition

Table 303. dbms.security.log_successful_authentication

Description	Set to log successful authentication events to the security log. If this is set to `false` only failed authentication events will be logged, which could be useful if you find that the successful events spam the logs too much, and you do not require full auditing capability.
Valid values	A boolean.
Default value	`true`

dbms.security.logs.ldap.groups_at_debug_level_enabled

Enterprise Edition

Table 304. dbms.security.logs.ldap.groups_at_debug_level_enabled

Description	When set to `true`, will log the groups retrieved from the ldap server. This will only take effect when the security log level is set to `DEBUG`.WARNING: It is strongly advised that this is set to `false` when running in a production environment in order to prevent logging of sensitive information.
Valid values	A boolean.
Default value	`false`

dbms.security.oidc.<provider>.audience

Enterprise Edition Dynamic

Table 305. dbms.security.oidc.<provider>.audience

Description	Expected values of the Audience (aud) claim in the id token.
Valid values	A comma-separated list where each element is a string, which Can not be empty.

dbms.security.oidc.<provider>.auth_endpoint

Enterprise Edition Dynamic

Table 306. dbms.security.oidc.<provider>.auth_endpoint

Description	The OIDC authorization endpoint. If this is not supplied Neo4j will attempt to discover it from the well_known_discovery_uri.
Valid values	a URI

dbms.security.oidc.<provider>.auth_flow

Enterprise Edition Dynamic

Table 307. dbms.security.oidc.<provider>.auth_flow

Description	The OIDC flow to use. This is exposed to clients via the discovery endpoint. Supported values are `pkce` and `implicit`
Valid values	One of [PKCE, IMPLICIT].
Default value	PKCE

dbms.security.oidc.<provider>.auth_params

Enterprise Edition Dynamic Deprecated in 5.0

Table 308. dbms.security.oidc.<provider>.auth_params

Description	Optional additional parameters that the auth endpoint requires. Please use params instead. The map is a semicolon separated list of key-value pairs. For example: `k1=v1;k2=v2`.
Valid values	A simple key value map pattern `k1=v1;k2=v2`.
Default value	{}

dbms.security.oidc.<provider>.authorization.group_to_role_mapping

Enterprise Edition Dynamic

Table 309. dbms.security.oidc.<provider>.authorization.group_to_role_mapping

Description	An authorization mapping from IdP group names to Neo4j role names. The map should be formatted as a semicolon separated list of key-value pairs, where the key is the IdP group name and the value is a comma separated list of corresponding role names. For example: group1=role1;group2=role2;group3=role3,role4,role5 You could also use whitespaces and quotes around group names to make this mapping more readable, for example:
	`dbms.security.oidc.<provider>.authorization.group_to_role_mapping=\`
	`"Neo4j Read Only" = reader; \`
	`"Neo4j Read-Write" = publisher; \`
	`"Neo4j Schema Manager" = architect; \`
	`"Neo4j Administrator" = admin`

Valid values	A string that must be semicolon-separated list of key-value pairs or empty

dbms.security.oidc.<provider>.claims.groups

Enterprise Edition | Dynamic

Table 310. dbms.security.oidc.<provider>.claims.groups

Description	The claim to use as the list of groups in Neo4j. These could be Neo4J roles directly, or can be mapped using dbms.security.oidc.<provider>.authorization.group_to_role_mapping. From Neo4j 5.4, the JWT claim may also contain a single group returned as A string. as well as a list of groups as was previously required.
Valid values	A string.

dbms.security.oidc.<provider>.claims.username

Enterprise Edition | Dynamic

Table 311. dbms.security.oidc.<provider>.claims.username

Description	The claim to use as the username in Neo4j. This would typically be sub, but in some situations it may be be desirable to use something else such as email.
Valid values	A string.
Default value	sub

dbms.security.oidc.<provider>.client_id

Enterprise Edition | Dynamic | Deprecated in 5.19

Table 312. dbms.security.oidc.<provider>.client_id

Description	Client id. Not used. This value was previously used to validate the `azp` claim in the id_token, but this validation has been removed in line with updates to the OIDC specification.
Valid values	A string.

dbms.security.oidc.<provider>.config

Enterprise Edition | Dynamic

Table 313. dbms.security.oidc.<provider>.config

Description	The accepted values (all optional) are: • `principal`: in which JWT claim the user's email address is specified, email is the default. This is the value that will be shown in browser. • `code_challenge_method`: default is `S256` and it's the only supported method at this moment. This setting applies only for pkce auth flow • `token_type_principal`: the options are almost always either `access_token`, which is the default, or `id_token`. • `token_type_authentication`: the options are almost always either `access_token`, which is the default, or `id_token`. • `implicit_flow_requires_nonce`: `true` or `false`. Defaults to `false`.
Valid values	A simple key-value map pattern `k1=v1;k2=v2`. Valid key options are: `[implicit_flow_requires_nonce, token_type_authentication, token_type_principal, principal, code_challenge_method]`.
Default value	`{}`

`dbms.security.logs.oidc.jwt_claims_at_debug_level_enabled`

Table 314. dbms.security.logs.oidc.jwt_claims_at_debug_level_enabled

Description	When set to `true`, it logs the claims from the JWT. This will only take effect when the security log level is set to `DEBUG`. **Warning:** It is strongly advised that this is set to `false` when running in a production environment in order to prevent logging of sensitive information. Also note that the contents of the JWT claims set can change over time because they are dependent entirely upon the ID provider.
Valid values	A boolean.
Default value	`false`

`dbms.security.oidc.<provider>.display_name`

Enterprise Edition

Table 315. dbms.security.oidc.<provider>.display_name

Description	The user-facing name of the provider as provided by the discovery endpoint to clients (Bloom, Browser etc.).
Valid values	A string.

`dbms.security.oidc.<provider>.get_groups_from_user_info`

Enterprise Edition Dynamic

Table 316. dbms.security.oidc.<provider>.get_groups_from_user_info

Description	When turned on, Neo4j gets the groups from the provider user info endpoint.
Valid values	A boolean.
Default value	`false`

`dbms.security.oidc.<provider>.get_username_from_user_info`

Enterprise Edition Dynamic

Table 317. dbms.security.oidc.<provider>.get_username_from_user_info

Description	When turned on, Neo4j gets the username from the provider user info endpoint.
Valid values	A boolean.
Default value	`false`

`dbms.security.oidc.<provider>.issuer`

Enterprise Edition Dynamic

Table 318. dbms.security.oidc.<provider>.issuer

Description	The expected value of the iss claim in the id token. If this is not supplied Neo4j will attempt to discover it from the well_known_discovery_uri.
Valid values	A string.

`dbms.security.oidc.<provider>.jwks_uri`

Enterprise Edition Dynamic

Table 319. dbms.security.oidc.<provider>.jwks_uri

Description	The location of the JWK public key set for the identity provider. If this is not supplied Neo4j will attempt to discover it from the well_known_discovery_uri.
Valid values	a URI

`dbms.security.oidc.<provider>.params`

Enterprise Edition Dynamic

Table 320. dbms.security.oidc.<provider>.params

Description	The map is a semicolon separated list of key-value pairs. For example: `k1=v1;k2=v2`. The user should at least provide:
	`client_id: the SSO Idp client idenfifier.`
	`response_type: code if auth_flow is pkce or token for implicit auth_flow.`
	`scope: often containing a subset of 'email profile openid groups'.`
	For example: `client_id=my-client-id;response_type=code;scope=openid profile email`.

Valid values	A simple key-value map pattern `k1=v1;k2=v2`. Required key options are: `[scope, client_id, response_type]`.
Default value	`{}`

dbms.security.oidc.<provider>.token_endpoint

Enterprise Edition | Dynamic

Table 321. dbms.security.oidc.<provider>.token_endpoint

Description	The OIDC token endpoint. If this is not supplied Neo4j will attempt to discover it from the well_known_discovery_uri.
Valid values	a URI

dbms.security.oidc.<provider>.token_params

Enterprise Edition | Dynamic

Table 322. dbms.security.oidc.<provider>.token_params

Description	Optional query parameters that the token endpoint requires. The map is a semicolon separated list of key-value pairs. For example: `k1=v1;k2=v2`.If the token endpoint requires a client_secret then this parameter should contain `client_secret=super-secret`
Valid values	A simple key value map pattern `k1=v1;k2=v2`.
Default value	`{}`

dbms.security.oidc.<provider>.user_info_uri

Enterprise Edition | Dynamic

Table 323. dbms.security.oidc.<provider>.user_info_uri

Description	The identity providers user info uri.
Valid values	a URI

dbms.security.oidc.<provider>.well_known_discovery_uri

Enterprise Edition | Dynamic

Table 324. dbms.security.oidc.<provider>.well_known_discovery_uri

Description	OpenID Connect Discovery endpoint used to fetch identity provider settings. If not provided, `issuer`, `jwks_uri`, `auth_endpoint` should be present. If the auth_flow is pkce, `token_endpoint` should also be provided.
Valid values	a URI

`dbms.security.procedures.allowlist`

Table 325. dbms.security.procedures.allowlist

Description	A list of procedures (comma separated) that are to be loaded. The list may contain both fully-qualified procedure names, and partial names with the wildcard `*`. The default (`*`) loads all procedures. If no value is specified, no procedures will be loaded.
Valid values	A comma-separated list where each element is a string.
Default value	*

`dbms.security.procedures.unrestricted`

Table 326. dbms.security.procedures.unrestricted

Description	A list of procedures and user-defined functions (comma separated) that are allowed full access to the database. The list may contain both fully-qualified procedure names, and partial names with the wildcard `*`. Note that this enables these procedures to bypass security. Use with caution.
Valid values	A comma-separated list where each element is a string.
Default value	

`dbms.security.require_local_user`

Enterprise Edition | Introduced in 5.24

Table 327. dbms.security.require_local_user

Description	This controls if a local user has to be created for external authentication. If set to the default (`false`), no user has to be created to authenticate with an external authentication provider. If set to `true`, a user representing the external user must be created before they can authenticate successfully.
Valid values	A boolean.
Default value	`false`

`dbms.netty.ssl.provider`

Table 328. dbms.netty.ssl.provider

Description	Netty SSL provider.
Valid values	One of [JDK, OPENSSL, OPENSSL_REFCNT].

Default value	JDK

Server directories settings

The server directories settings can be used to change the default locations of your Neo4j files. For more information, see Default file locations.

`server.directories.cluster_state`

Enterprise Edition

Table 329. server.directories.cluster_state

Description	Directory to hold cluster state including Raft log.
Valid values	A path. If relative, it is resolved from server.directories.data.
Default value	`cluster-state`

`server.directories.data`

Table 330. server.directories.data

Description	Path of the data directory. You must not configure more than one Neo4j installation to use the same data directory.
Valid values	A path. If relative, it is resolved from server.directories.neo4j_home.
Default value	`data`

`server.directories.dumps.root`

Table 331. server.directories.dumps.root

Description	Root location where Neo4j will store database dumps optionally produced when dropping said databases.
Valid values	A path. If relative, it is resolved from server.directories.data.
Default value	`dumps`

`server.directories.import`

Table 332. server.directories.import

Description	Sets the root directory for file URLs used with the Cypher `LOAD CSV` clause. This should be set to a directory relative to the Neo4j installation path, restricting access to only those files within that directory and its subdirectories. For example the value "import" will only enable access to files within the 'import' folder. Removing this setting will disable the security feature, allowing all files in the local system to be imported. Setting this to an empty field will allow access to all files within the Neo4j installation folder.
Valid values	A path. If relative, it is resolved from server.directories.neo4j_home.

Default value	

`server.directories.lib`

Table 333. server.directories.lib

Description	Path of the lib directory.
Valid values	A path. If relative, it is resolved from server.directories.neo4j_home.
Default value	`lib`

`server.directories.licenses`

Table 334. server.directories.licenses

Description	Path of the licenses directory.
Valid values	A path. If relative, it is resolved from server.directories.neo4j_home.
Default value	`licenses`

`server.directories.logs`

Table 335. server.directories.logs

Description	Path of the logs directory.
Valid values	A path. If relative, it is resolved from server.directories.neo4j_home.
Default value	`logs`

`server.directories.metrics`

Enterprise Edition

Table 336. server.directories.metrics

Description	The target location of the CSV files: a path to a directory wherein a CSV file per reported field will be written.
Valid values	A path. If relative, it is resolved from server.directories.neo4j_home.
Default value	`metrics`

`server.directories.neo4j_home`

Table 337. server.directories.neo4j_home

Description	Root relative to which directory settings are resolved. Calculated and set by the server on startup. Defaults to the current working directory.
Valid values	A path that is absolute.

Default value	

`server.directories.plugins`

Table 338. server.directories.plugins

Description	Location of the database plugin directory. Compiled Java JAR files that contain database procedures will be loaded if they are placed in this directory.
Valid values	A path. If relative, it is resolved from server.directories.neo4j_home.
Default value	`plugins`

`server.directories.run`

Table 339. server.directories.run

Description	Path of the run directory. This directory holds Neo4j's runtime state, such as a pidfile when it is running in the background. The pidfile is created when starting neo4j and removed when stopping it. It may be placed on an in-memory filesystem such as tmpfs.
Valid values	A path. If relative, it is resolved from server.directories.neo4j_home.
Default value	`run`

`server.directories.script.root`

Table 340. server.directories.script.root

Description	Root location where Neo4j will store scripts for configured databases.
Valid values	A path. If relative, it is resolved from server.directories.data.
Default value	`scripts`

`server.directories.transaction.logs.root`

Table 341. server.directories.transaction.logs.root

Description	Root location where Neo4j will store transaction logs for configured databases.
Valid values	A path. If relative, it is resolved from server.directories.data.
Default value	`transactions`

Server settings

Server settings apply only to the specific server and can be varied between configuration files across a cluster/DBMS.

`server.backup.enabled`

Enterprise Edition

Table 342. server.backup.enabled

Description	Enable support for running online backups.
Valid values	A boolean.
Default value	`true`

`server.backup.exec_connector.command`

Enterprise Edition

Table 343. server.backup.exec_connector.command

Description	Command to execute for ExecDataConnector list
Valid values	A string.
Default value	

`server.backup.exec_connector.scheme`

Enterprise Edition

Table 344. server.backup.exec_connector.scheme

Description	Schemes ExecDataConnector will match on
Valid values	A comma-separated list where each element is a string.
Default value	

`server.backup.listen_address`

Enterprise Edition

Table 345. server.backup.listen_address

Description	Network interface and port for the backup server to listen on.
Valid values	A socket address in the format of `hostname:port`, `hostname`, or `:port`.
Default value	`127.0.0.1:6362`

`server.backup.store_copy_max_retry_time_per_request`

Enterprise Edition

Table 346. server.backup.store_copy_max_retry_time_per_request

Description	Maximum retry time per request during store copy. Regular store files and indexes are downloaded in separate requests during store copy. This configures the maximum time failed requests are allowed to resend.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`20m`

`server.config.strict_validation.enabled`

Table 347. server.config.strict_validation.enabled

Description	A strict configuration validation will prevent the database from starting up if unknown configuration options are specified in the neo4j settings namespace (such as dbms., cypher., etc) or if settings are declared multiple times.
Valid values	A boolean.
Default value	`true`

`server.databases.default_to_read_only`

Dynamic

Table 348. server.databases.default_to_read_only

Description	Whether or not any database on this instance is read_only by default. If `false`, individual databases may be marked as read_only using server.database.read_only. If `true`, individual databases may be marked as writable using `server.databases.writable`.
Valid values	A boolean.
Default value	`false`

`server.databases.read_only`

Dynamic

Table 349. server.databases.read_only

Description	List of databases for which to prevent write queries. Databases not included in this list maybe read_only anyway depending upon the value of `server.databases.default_to_read_only`.
Valid values	A comma-separated set where each element is a valid database name containing only alphabetic characters, numbers, dots, and dashes with a length between 3 and 63 characters, starting with an alphabetic character but not with the name system.
Default value	

`server.databases.writable`

Dynamic

Table 350. server.databases.writable

Description	List of databases for which to allow write queries. Databases not included in this list will allow write queries anyway, unless `server.databases.default_to_read_only` is set to `true`.
Valid values	A comma-separated set where each element is a valid database name containing only alphabetic characters, numbers, dots, and dashes with a length between 3 and 63 characters, starting with an alphabetic character but not with the name system.
Default value	

`server.dynamic.setting.allowlist`

Enterprise Edition

Table 351. server.dynamic.setting.allowlist

Description	A list of setting name patterns (comma separated) that are allowed to be dynamically changed. The list may contain both full setting names, and partial names with the wildcard `*`. If this setting is left empty all dynamic settings updates will be blocked.
Valid values	A comma-separated list where each element is a string.
Default value	*

`server.jvm.additional`

Table 352. server.jvm.additional

Description	Additional JVM arguments. Argument order can be significant. To use a Java commercial feature, the argument to unlock commercial features must precede the argument to enable the specific feature in the config value string.
Valid values	One or more jvm arguments.
Default value	

For details about the default values of `server.jvm.additional`, see The *neo4j.conf* file → Default values of `server.jvm.additional`.

`server.max_databases`

Enterprise Edition Deprecated in 5.6

Table 353. server.max_databases

Description	The maximum number of databases.
Valid values	A long that is minimum `2`.
Default value	`100`
Replaced by	`dbms.max_databases`

`server.panic.shutdown_on_panic`

Enterprise Edition

Table 354. server.panic.shutdown_on_panic

Description	If there is a Database Management System Panic (an irrecoverable error) should the neo4j process shut down or continue running. Following a DbMS panic it is likely that a significant amount of functionality will be lost. Recovering full functionality will require a Neo4j restart. Default is `false` except for Neo4j Enterprise Edition deployments running on Kubernetes where it is `true`.
Valid values	A boolean.
Default value	`false`

`server.threads.worker_count`

Table 355. server.threads.worker_count

Description	Number of Neo4j worker threads. This setting is only valid for REST and does not influence bolt-server. It sets the number of worker threads for the Jetty server used by neo4j-server. This option can be tuned when you plan to execute multiple, concurrent REST requests, to get more throughput from the database. Your OS might enforce a lower limit than the maximum value specified here. Number of available processors, or 500 for machines that have more than 500 processors.
Valid values	An integer that is in the range `1` to `44738`.
Default value	

`server.unmanaged_extension_classes`

Table 356. server.unmanaged_extension_classes

Description	Comma-separated list of <classname>=<mount point> for unmanaged extensions.
Valid values	A comma-separated list where each element is `<classname>=<mount point>` string.
Default value	

`server.windows_service_name`

Table 357. server.windows_service_name

Description	Name of the Windows Service managing Neo4j when installed using `neo4j install-service`. Only applicable on Windows OS. NOTE: This must be unique for each installation.
Valid values	A string.
Default value	`neo4j`

Transaction settings

The transaction settings helps you manage the transactions in your database, for example, the transaction timeout, the lock acquisition timeout, the maximum number of concurrently running transactions, etc. For more information, see Manage transactions and Concurrent data access.

`db.lock.acquisition.timeout`

Dynamic

Table 358. db.lock.acquisition.timeout

Description	The maximum time interval within which lock should be acquired. Zero (default) means the timeout is disabled.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`0s`

`db.shutdown_transaction_end_timeout`

Table 359. db.shutdown_transaction_end_timeout

Description	The maximum amount of time to wait for running transactions to complete before allowing initiated database shutdown to continue.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`10s`

`db.transaction.bookmark_ready_timeout`

Dynamic

Table 360. db.transaction.bookmark_ready_timeout

Description	The maximum amount of time to wait for the database state represented by the bookmark.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) that is minimum `1s`.
Default value	`30s`

`db.transaction.concurrent.maximum`

Dynamic

Table 361. db.transaction.concurrent.maximum

Description	The maximum number of concurrently running transactions. If set to 0, the limit is disabled.
Valid values	An integer.
Default value	`1000`

db.transaction.monitor.check.interval

Table 362. db.transaction.monitor.check.interval

Description	Configures the time interval between transaction monitor checks. Determines how often the monitor thread will check a transaction for timeout.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`2s`

db.transaction.sampling.percentage

Dynamic

Table 363. db.transaction.sampling.percentage

Description	Transaction sampling percentage.
Valid values	An integer that is in the range `1` to `100`.
Default value	`5`

db.transaction.timeout

Dynamic

Table 364. db.transaction.timeout

Description	The maximum time interval of a transaction within which it should be completed.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`0s`

db.transaction.tracing.level

Dynamic

Table 365. db.transaction.tracing.level

Description	Transaction creation tracing level.
Valid values	One of [DISABLED, SAMPLE, ALL].
Default value	`DISABLED`

server.http.transaction_idle_timeout

Introduced in 5.20

Table 366. server.http.transaction_idle_timeout

Description	Timeout for idle transactions in the HTTP Server. NOTE: This is different from 'db.transaction.timeout' which will timeout the underlying transaction.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`30s`

`server.queryapi.transaction_idle_timeout`

Introduced in 5.26

Table 367. server.queryapi.transaction_idle_timeout

Description	Timeout for idle transactions in the Query API. Note: this is different from 'db.transaction.timeout' which will timeout the underlying transaction.
Valid values	A duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`).
Default value	`1m`

Transaction log settings

Transaction logs keep the list of transactions that have not yet been applied to the store files. This is necessary for recovery. The following settings configure the number of transaction logs left after a pruning operation and the size of the transaction log files.

See also Checkpoint settings.

`db.recovery.fail_on_missing_files`

Table 368. db.recovery.fail_on_missing_files

Description	If `true`, Neo4j will abort recovery if transaction log files are missing. Setting this to `false` will allow Neo4j to create new empty missing files for the already existing database, but the integrity of the database might be compromised.
Valid values	A boolean.
Default value	`true`

`db.tx_log.buffer.size`

Table 369. db.tx_log.buffer.size

Description	On serialization of transaction logs, they will be temporary stored in the byte buffer that will be flushed at the end of the transaction or at any moment when the buffer will be full. By default, the size of the byte buffer is based on the number of available CPU's with a minimal buffer size of 512KB. Every other 4 CPU's will add another 512KB into the buffer size. The maximal buffer size in this default scheme is 4MB taking into account that you can have one transaction log writer per database in multi-database env. For example, runtime with 4 CPUs will have the buffer size of 1MB; runtime with 8 CPUs will have the buffer size of 1MB 512KB; runtime with 12 CPUs will have the buffer size of 2MB.

Valid values	A long that is minimum `131072`.
Default value	

db.tx_log.preallocate

Dynamic

Table 370. db.tx_log.preallocate

Description	Specify if Neo4j should try to preallocate the logical log file in advance. It optimizes file system by ensuring there is room to accommodate newly generated files and avoid file-level fragmentation.
Valid values	A boolean.
Default value	`true`

db.tx_log.rotation.retention_policy

Changed in 5.13 Dynamic

Table 371. db.tx_log.rotation.retention_policy

Description	Specify how long Neo4j should keep logical transaction logs to backup the database. For example, `10 days` prunes logical logs that only contain transactions older than 10 days. Alternatively, `100k txs` keeps the 100k latest transactions from each database and prunes any older transactions. From Neo4j 5.9 onwards, you can optionally add a period-based restriction to the size of logs to keep. For example, `2 days 1G` prunes logical logs that only contain transactions older than 2 days or are larger than 1G.
Valid values	A string that matches the pattern `^(true|keep_all|false|keep_none|(\d+[KkMmGg]?( (files|size|txs|entries|hours( \d+[KkMmGg]?)?|days( \d+[KkMmGg]?)?))))$` (Must be `true` or `keep_all`, `false` or `keep_none`, or of format `<number><optional unit> <type> <optional space restriction>`. Valid units are `K`, `M` and `G`. Valid types are `files`, `size`, `txs`, `entries`, `hours` and `days`. Valid optional space restriction is a logical log space restriction like 100M. For example, `100M size` will limit logical log space on disk to 100MiB per database, and `200K txs` will limit the number of transactions kept to 200 000 per database.).
Default value	`2 days 2G`

db.tx_log.rotation.size

Dynamic

Table 372. db.tx_log.rotation.size

Description	Specifies at which file size the logical log will auto-rotate. The minimum accepted value is 128 KiB.
Valid values	A byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `T` `PiB`, `PB`, `EiB`, `EB`) that is minimum `128.00KiB`.
Default value	`256.00MiB`

[1] To view *neo4j.log* in Docker, use `docker logs <containerID/name>`.
[2] To view the neo4j.log for Debian and RPM, use `journalctl --unit=neo4j`.
[3] When Neo4j is used in embedded mode, the default value is `false`.
[4] The default value for `server.https.enabled` is `false`.
[5] See Neo4j Admin and Neo4j CLI → Configuration for details.
[6] You can also get basic access, without a license key, to Neo4j Aura console.
[7] Graph Data Science is available in both Neo4j CE and EE editions. The Enterprise Edition includes additional features and requires a license key.

Database administration

Neo4j is a Database Management System, or DBMS, capable of managing multiple databases. The DBMS can manage a standalone server, or a group of servers in a cluster.

A database is an administrative partition of a DBMS. In practical terms, it is a physical structure of files organized within a directory or folder, that has the same name of the database. This chapter describes how to manage local and remote standard databases, composite databases, and database aliases.

Standard databases

In Neo4j 5, each standard database contains a single graph. Many administrative commands refer to a specific graph by using the database name.

A database defines a *transaction domain* (a collection of graphs that can be updated within the context of a single transaction) and an *execution context* (a runtime environment for the execution of a request). This means that a transaction cannot span across multiple databases. Similarly, a procedure is called within a database, although its logic may access data that is stored in other databases.

Standard databases per Neo4j edition

The edition of Neo4j determines the number of possible databases:

- Installations of Community Edition can have exactly **one** standard database.
- Installations of Enterprise Edition can have any number of standard databases.

Default database

A default installation of Neo4j 5 contains one standard database, named `neo4j`, which is the default database for the DBMS. A different name can be configured before starting Neo4j for the first time. For details, see Configuration parameters.

In the Enterprise Edition, the default standard database `neo4j` can be deleted. To delete it, you must first set another database as the default by calling the procedure dbms.setDefaultDatabase against the `system` database. For more information, see the Change the default database.

The following image illustrates an installation of Neo4j containing the three standard databases, named `marketing`, `sales`, and `hr`, and the `system` database. The default database is `sales`:

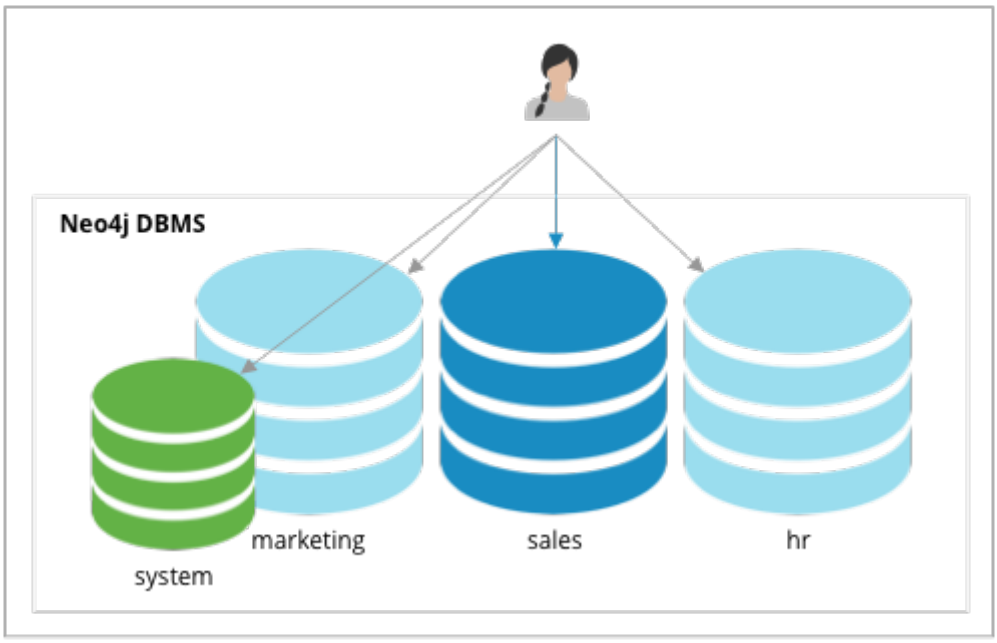


Figure 2. A multiple database Neo4j installation, with a default database.

> **Note:**
>
> Be aware that the automatically created *initial* default database may have a different topology to the default configuration values. See Default database in a cluster for more information.

Per-user home databases

The home database is the database that you connect to by default when no database is specified. It is different from the default database, which is the database that the server uses when no home database is specified.

Per-user home databases are controlled via the Cypher administration commands.

To set a home database for a user, this user must exist as a record in Neo4j. Therefore, for deployments using auth providers other than native, you create a native user with a matching username and then set a home database for that user. For more information on creating native users and configuring a home database for a user, see Manage users.

The `system` database

All installations include a built-in database named `system`, which contains metadata on the DBMS and security configuration.

The `system` database behaves differently than all other databases. In particular, when connected to this database you can only perform a specific set of administrative tasks, such as managing databases, aliases, servers, and access control.

The `system` database cannot be deleted.

Most of the available administrative commands are restricted to users with specific administrative privileges. An example of configuring security privileges is described in Fine-grained access control.

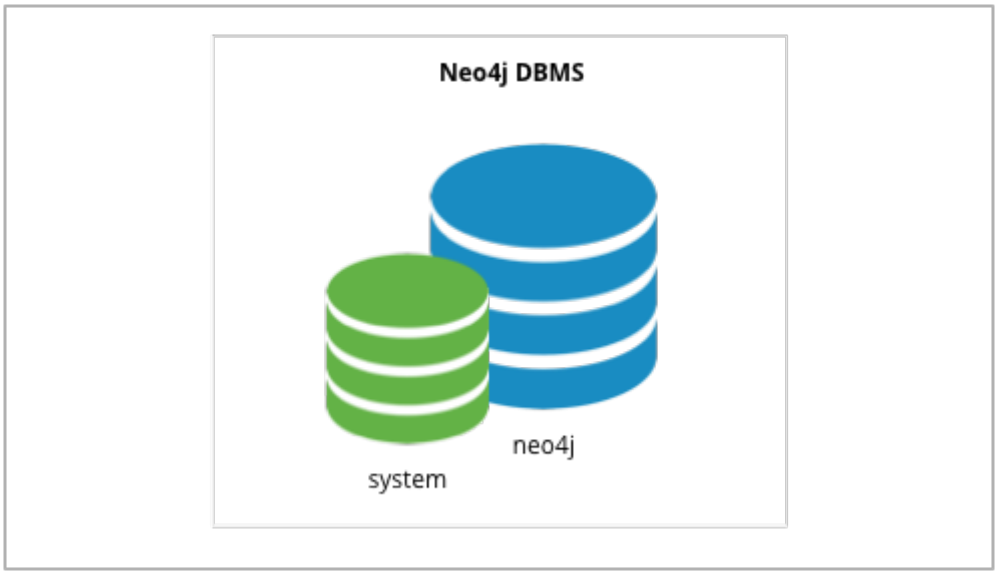


Figure 3. A default Neo4j installation.

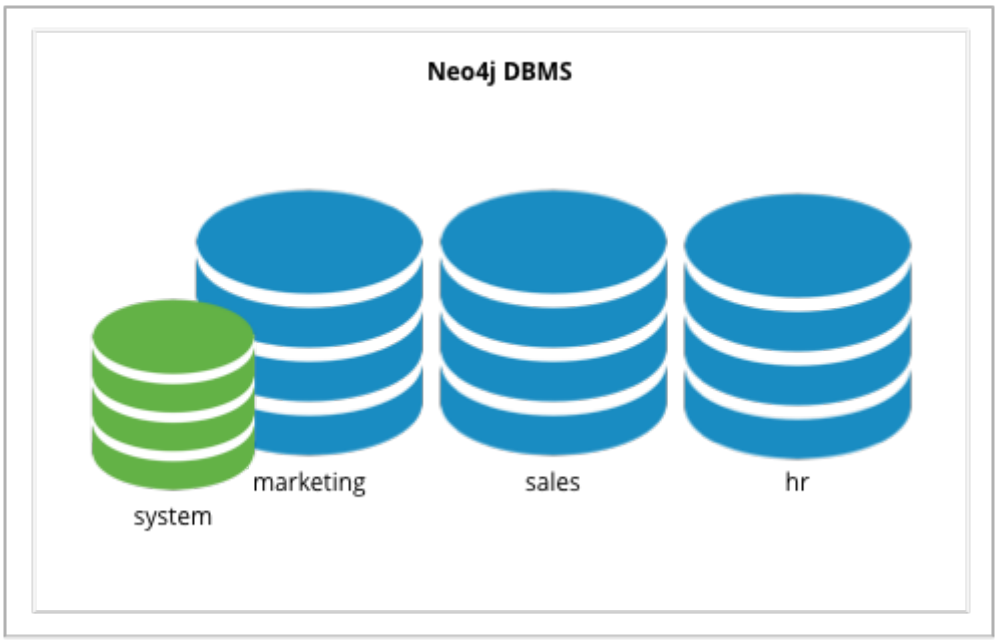


Figure 4. A multiple database Neo4j installation.

For details about the `system` database in a clustered environment, refer to Managing databases in a cluster → The `system` database.

Composite databases

A Composite database is a logical grouping of multiple graphs contained in other, standard databases. A Composite database defines an *execution context* and a (limited) *transaction domain*. For more information, see Composite databases.

Database management command syntax

Almost all administration commands have variations. The most common are parts of the command that are optional or that can have multiple values.

Reading the administration commands syntax

Table 373. Special characters in syntax summaries

Character	Meaning	Example
`|`	Used to indicate alternative parts of a command (i.e. `or`). Needs to be part of a grouping.	If the syntax needs to specify either a name or `*`, this can be indicated with `* | name`.
`{` and `}`	Used to group parts of the command. Commonly found together with `|`.	In order to use the `or` in the syntax summary, it needs to be in a group: `{* | name}`.
`[` and `]`	Used to indicate an optional part of the command. It also groups alternatives together, when there can be either of the alternatives or nothing.	If a keyword in the syntax can either be in singular or plural, you can indicate that the `S` is optional with `GRAPH[S]`.
`...`	Repeated pattern. Related to the command part immediately before this is repeated.	A comma separated list of names would be `name[, ...]`.
`"`	When a special character is part of the syntax itself, surround it with `"` to indicate this.	To include `{` in the syntax use `"{" { * | name } "}"`. In this case, you will get either `{ * }` or `{ name }`.

The special characters in the table above are the only ones that need to be escaped using `"` in the syntax summaries.

Here is an example that uses all the special characters. It grants the `READ` privilege:

```
GRANT READ
  "{" { * | property[, ...] } "}"
  ON {HOME GRAPH | GRAPH[S] { * | name[, ...] }}
    [ ELEMENT[S] { * | label-or-rel-type[, ...] }
    | NODE[S] { * | label[, ...] }
    | RELATIONSHIP[S] { * | rel-type[, ...] }]
  TO role[, ...]
```

Note that this command includes `{` and `}` in the syntax, and between them there can be a grouping of properties or the character `*`. It also has multiple optional parts, including the entity part of the command which is the grouping following the graph name. For details about the graph privilege commands syntax, see Components of the graph privilege commands.

However, there is no need to escape any characters when creating a constraint for a node property. This is because `(` and `)` are not special characters, and `[` and `]` indicate that the constraint name and the `IF NOT EXISTS` parts are optional, and therefore not part of the command.

```
CREATE CONSTRAINT [constraint_name] [IF NOT EXISTS]
FOR (n:LabelName)
REQUIRE n.propertyName IS NOT NULL
```

Database management command syntax

The database management commands are used to manage standard or composite databases.

Show databases

Command	Syntax

SHOW DATABASE	`SHOW { DATABASE[S] name | DATABASE[S] | DEFAULT DATABASE | HOME DATABASE }`
	`[WHERE expression]`
	
	`SHOW { DATABASE[S] name | DATABASE[S] | DEFAULT DATABASE | HOME DATABASE }`
	`YIELD { * | field[, ...] } [ORDER BY field[, ...]] [SKIP n] [LIMIT n]`
	`[WHERE expression]`
	`[RETURN field[, ...] [ORDER BY field[, ...]] [SKIP n] [LIMIT n]]`

Create a database

Command	Syntax

CREATE DATABASE	`CREATE DATABASE name [IF NOT EXISTS]`
	`[TOPOLOGY n PRIMAR{Y|IES} [m SECONDAR{Y|IES}]]`
	`[OPTIONS "{" option: value[, ...] "}"]`
	`[WAIT [n [SEC[OND[S]]]]|NOWAIT]`
	
	`CREATE OR REPLACE DATABASE name`
	`[TOPOLOGY n PRIMAR{Y|IES} [m SECONDAR{Y|IES}]]`
	`[OPTIONS "{" option: value[, ...] "}"]`
	`[WAIT [n [SEC[OND[S]]]]|NOWAIT]`

Create a composite database

Command	Syntax

CREATE COMPOSITE DATABASE	`CREATE COMPOSITE DATABASE name [IF NOT EXISTS]`
	`[OPTIONS "{" "}"]`
	`[WAIT [n [SEC[OND[S]]]]|NOWAIT]`
	
	`CREATE OR REPLACE COMPOSITE DATABASE name`
	`[OPTIONS "{" "}"]`
	`[WAIT [n [SEC[OND[S]]]]|NOWAIT]`

Alter a database

Command	Syntax

ALTER DATABASE	`ALTER DATABASE name [IF EXISTS]`
	`{`
	`SET ACCESS {READ ONLY | READ WRITE} |`
	`SET TOPOLOGY n PRIMAR{Y|IES} [m SECONDAR{Y|IES}] |`
	`SET OPTION option value`
	`}`
	`[WAIT [n [SEC[OND[S]]]]|NOWAIT]`
	
	`ALTER DATABASE name [IF EXISTS]`
	`REMOVE OPTION option`
	`[WAIT [n [SEC[OND[S]]]]|NOWAIT]`

Note:

There can be multiple `SET OPTION` or `REMOVE OPTION` clauses for different option keys.

Stop a database

Command	Syntax
STOP DATABASE	`STOP DATABASE name [WAIT [n [SEC[OND[S]]]]|NOWAIT]`

Start a database

Command	Syntax
START DATABASE	`START DATABASE name [WAIT [n [SEC[OND[S]]]]|NOWAIT]`

Delete a database

Command	Syntax
DROP DATABASE	`DROP [COMPOSITE] DATABASE name [IF EXISTS] [RESTRICT | CASCADE ALIAS[ES]] [{DUMP|DESTROY} [DATA]]` `[WAIT [n [SEC[OND[S]]]]|NOWAIT]`

Database alias management command syntax

The database alias management commands are used to manage local or remote database aliases.

Show aliases

Command	Syntax

Show Database Alias	`SHOW ALIAS[ES] [name] FOR DATABASE[S]`
	`[WHERE expression]`
	
	`SHOW ALIAS[ES] [name] FOR DATABASE[S]`
	`YIELD { * | field[, ...] } [ORDER BY field[, ...]] [SKIP n] [LIMIT n]`
	`[WHERE expression]`
	`[RETURN field[, ...] [ORDER BY field[, ...]] [SKIP n] [LIMIT n]]`
	
	Lists both local and remote database aliases, optionally filtered on the alias name.

Create a local alias

Command	Syntax

CREATE ALIAS	`CREATE ALIAS name [IF NOT EXISTS] FOR DATABASE targetName`
	`[PROPERTIES "{" key: value[, ...] "}"]`
	
	`CREATE OR REPLACE ALIAS name FOR DATABASE targetName`
	`[PROPERTIES "{" key: value[, ...] "}"]`

Create a remote alias

Command	Syntax

CREATE ALIAS	`CREATE ALIAS name [IF NOT EXISTS] FOR DATABASE targetName`
	`AT 'url' USER username PASSWORD 'password'`
	`[DRIVER "{" setting: value[, ...] "}"]`
	`[PROPERTIES "{" key: value[, ...] "}"]`
	
	`CREATE OR REPLACE ALIAS name FOR DATABASE targetName`
	`AT 'url' USER username PASSWORD 'password'`
	`[DRIVER "{" setting: value[, ...] "}"]`
	`[PROPERTIES "{" key: value[, ...] "}"]`

Alter a local alias

Command	Syntax

ALTER ALIAS	`ALTER ALIAS name [IF EXISTS] SET DATABASE`
	`[TARGET targetName]`
	`[PROPERTIES "{" key: value[, ...] "}"]`

Alter a remote alias

Command	Syntax
ALTER ALIAS	`ALTER ALIAS name [IF EXISTS] SET DATABASE [TARGET targetName AT 'url'] [USER username] [PASSWORD 'password'] [DRIVER "{" setting: value[, ...] "}"] [PROPERTIES "{" key: value[, ...] "}"]`

Delete an alias

Command	Syntax
DROP ALIAS	`DROP ALIAS name [IF EXISTS] FOR DATABASE` Drop either a local or remote database alias.

Standard databases

Naming rules for databases

Enterprise Edition

Database names are subject to the standard Cypher restrictions on valid identifiers. See Cypher Manual → Naming rules and recommendations.

Naming rules for databases are as follows:

- Length must be between 3 and 63 characters.
- The first character of a name must be an ASCII alphabetic character.
- Subsequent characters must be ASCII alphabetic or numeric characters, dots or dashes; `[a..z][0..9].-`.
- Names cannot end with dots or dashes.
- Names are case-insensitive and normalized to lowercase.
- Names that begin with an underscore and with the prefix `system` are reserved for internal use.

> **Note:**
>
> The `-` (dash) and `.` (dot) characters are not legal in Cypher variables. Names with a `-` in them must be enclosed within backticks. For example, `CREATE DATABASE \`main-db\`` is a valid database name. Using dots in database names is not recommended, as it makes it difficult to determine if a dot is part of the database name or a delimiter for a database alias in a composite database.

It is possible to create an alias to refer to an existing database to avoid these restrictions. For more information, see Creating database aliases.

Create, start, and stop databases

Enterprise Edition Not available on Aura

Neo4j supports the management of multiple databases within the same DBMS. The metadata for these databases, including the associated security model, is maintained in a special database called the `system` database. All multi-database administrative commands must be run against the `system` database. These administrative commands are automatically routed to the `system` database when connected to the DBMS over Bolt.

> **Note:**
>
> Administrative commands should not be used during a rolling upgrade. For more information, see Upgrade and Migration Guide → Upgrade a cluster.

Create databases

You can create a database using the Cypher command `CREATE DATABASE`.

> **Note:**
>
> Database names are subject to the rules specified in Database names section. Having dots (`.`) in the database names is not recommended. This is due to the difficulty of determining if a dot is part of the database name or a delimiter for a database alias in a composite database.

In versions previous to Neo4j 5.22, the default store format for all new databases is `aligned`. From Neo4j 5.22, `block` is the default format for all newly-created databases as long as they do not have the `db.format` setting specified.
If you want to change it, you can set a new value for the `db.format` configuration in the *neo4j.conf* file. Alternatively, you can set the store format of new databases using the `CREATE DATABASE databasename OPTIONS {storeFormat: 'the-new-format'}` command. However, if the store is seeded with `seedURI`, `existingDataSeedServer` or `existingDataSeedInstance`, or if the command is being used to mount pre-existing store files already present on the disk, they will use their current store format without any alterations.

See Store formats, for more details about available database store formats in Neo4j.

Syntax

Command	Syntax

`CREATE DATABASE`	`CREATE DATABASE name [IF NOT EXISTS]`
	`[TOPOLOGY n PRIMAR{Y|IES} [m SECONDAR{Y|IES}]]`
	`[OPTIONS "{" option: value[, ...] "}"]`
	`[WAIT [n [SEC[OND[S]]]]|NOWAIT]`
	
	`CREATE OR REPLACE DATABASE name`
	`[TOPOLOGY n PRIMAR{Y|IES} [m SECONDAR{Y|IES}]]`
	`[OPTIONS "{" option: value[, ...] "}"]`
	`[WAIT [n [SEC[OND[S]]]]|NOWAIT]`

Options

The `CREATE DATABASE` command can have a map of options, e.g. `OPTIONS {key: 'value'}`.

Key	Value	Description
`existingData`	`use`	Controls how the system handles existing data on disk when creating the database. Currently, this is only supported with `existingDataSeedInstance`, `existingDataSeedServer` and `seedURI`, and must be set to `use`, which indicates the existing data files should be used for the new database.
`existingDataSeedServer` new in 5.25 `existingDataSeedInstance` deprecated in 5.25	ID of the cluster server	Defines which server is used for seeding the data of the created database. The server ID can be found in the `serverId` column after running `SHOW SERVERS`. `existingDataSeedInstance` is replaced by `existingDataSeedServer` in Neo4j 5.25.
`seedURI`	URI to a backup or a dump from an existing database.	Defines an identical seed from an external source which will be used to seed all servers. For more information, see Seed from a URI.
`seedConfig`	Comma-separated list of configuration values.	
`seedCredentials` Deprecated in 5.26	credentials	Defines credentials that need to be passed into certain seed providers. From 5.26, it is recommended to use the `CloudSeedProvider` seed provider, which does not require this configuration when seeding from cloud storage. For more information see CloudSeedProvider.
`txLogEnrichment`	`FULL` | `DIFF` | `OFF`	Defines the level of enrichment applied to transaction logs for Change Data Capture (CDC) purposes. For details about enrichment mode, see Change Data Capture Manual → Enable CDC on self-managed instances → Modify a database's CDC mode.

Key	Value	Description
`storeFormat`	`aligned` | `standard` | `high_limit` | `block`	Defines the store format if the database created is new. `high_limit` and `standard` formats are deprecated from 5.23. For more information on store formats, see Store formats. If the store is seeded with `seedURI`, `existingDataSeedInstance` or `existingDataSeedServer`, or if the command is used to mount pre-existing store files already present on the disk, they will retain their current store format without any modifications.

Note:

The `existingData`, `existingDataSeedInstance`, `existingDataSeedServer`, `seedURI`, `seedConfig`, and `seedCredentials` options cannot be combined with the `OR REPLACE` part of this command. More details about seeding options can be found in Seed a cluster.

Examples

Create a database

To create a database named `customers`, use the command `CREATE DATABASE` followed by the name of this database.

```
CREATE DATABASE customers
```

When you create a database, it shows up in the listing provided by the command `SHOW DATABASES`:

```
SHOW DATABASES YIELD name
```

Result

```
+-------------+
| name        |
+-------------+
| "customers" |
| "movies"    |
| "neo4j"     |
| "system"    |
+-------------+
```

Create a database with `WAIT`

Sub-clause `WAIT` allows you to specify a time limit in which the command must complete and return.

```
CREATE DATABASE slow WAIT 5 SECONDS
```

Result

```
+-----------------------------------------------------+
| address          | state      | message     | success |
+-----------------------------------------------------+
| "localhost:7687" | "CaughtUp" | "caught up" | TRUE    |
+-----------------------------------------------------+
```

The `success` column provides an aggregate status of whether or not the command is considered successful. Thus, every row has the same value, determined on a successful completion without a timeout.

Create databases with `IF NOT EXISTS` or `OR REPLACE`

The `CREATE DATABASE` command is optionally idempotent, with the default behavior to fail with an error if the database already exists. There are two ways to circumvent this behavior.

First, appending `IF NOT EXISTS` to the command ensures that no error is returned and that nothing happens if the database already exists.

```
CREATE DATABASE customers IF NOT EXISTS
```

Second, adding `OR REPLACE` to the command deletes any existing database and creates a new one.

```
CREATE OR REPLACE DATABASE customers
```

This is equivalent to running `DROP DATABASE customers IF EXISTS` followed by `CREATE DATABASE customers`.

Keep in mind that using `CREATE OR REPLACE DATABASE` also removes indexes and constraints. To preserve them, run the following Cypher commands before the `CREATE OR REPLACE DATABASE` and save their outputs:

```
SHOW CONSTRAINTS YIELD createStatement AS statement
```

```
SHOW INDEXES YIELD createStatement, owningConstraint
WHERE owningConstraint IS NULL
RETURN createStatement AS statement
```

The behavior of `IF NOT EXISTS` and `OR REPLACE` apply to both standard and composite databases (e.g. a composite database may replace a standard database or another composite database).

Note:

The `IF NOT EXISTS` and `OR REPLACE` parts of these commands cannot be used together.

Start databases

Databases can be started using the command `START DATABASE`.

> **Note:**
>
> Both standard databases and composite databases can be started using this command.

Syntax

Command	Syntax
START DATABASE	`START DATABASE name [WAIT [n [SEC[OND[S]]]]|NOWAIT]`

Examples

Start a database

Starting a database is a straightforward operation. Suppose you have a database named `customers`. To start it, use the following command:

```
START DATABASE customers
```

You can see the status of the started database by running the command `SHOW DATABASE name`.

```
SHOW DATABASE customers YIELD name, requestedStatus, currentStatus
```

Result

```
+-----------------------------------------------+
| name        | requestedStatus | currentStatus |
+-----------------------------------------------+
| "customers" | "online"        | "online"      |
+-----------------------------------------------+
```

Start a database with `WAIT`

You can start your database using `WAIT` sub-clause to ensure that the command waits for a specified amount of time until the database is started.

```
START DATABASE customers WAIT 5 SECONDS
```

Stop databases

Databases can be stopped using the command `STOP DATABASE`.

Syntax

Command	Syntax
STOP DATABASE	`STOP DATABASE name [WAIT [n [SEC[OND[S]]]]|NOWAIT]`

Examples

Stop a database

To stop a database, use the following command:

```
STOP DATABASE customers
```

> **Note:**
>
> Both standard databases and composite databases can be stopped using this command.

The status of the stopped database can be seen using the command `SHOW DATABASE name`:

```
SHOW DATABASE customers YIELD name, requestedStatus, currentStatus
```

Result

```
+-----------------------------------------------+
| name        | requestedStatus | currentStatus |
+-----------------------------------------------+
| "customers" | "offline"       | "offline"     |
+-----------------------------------------------+
```

Stop a database with `WAIT`

You can also stop your database using the `WAIT` sub-clause, which allows you to specify the amount of time that the system should wait for the database to stop.

```
STOP DATABASE customers WAIT 10 SECONDS
```

> **Note:**
>
> Databases that are stopped with the `STOP` command are completely shut down and may be started again through the `START` command. In a cluster, as long as a database is in a shutdown state, it can not be considered available to other members of the cluster. It is not possible to do online backups against shutdown databases and they need to be taken into special consideration during disaster recovery, as they do not have a running Raft machine while shutdown. Unlike stopped databases, dropped databases are completely removed and are not intended to be used again at all.

Create a database from a URI

This method seeds all databases with an identical seed from an external source, specified by a URI.

You specify the seed URI as an argument of the `CREATE DATABASE` command:

```
CREATE DATABASE foo OPTIONS {existingData: 'use', seedURI:'s3://myBucket/myBackup.backup'}
```

Download and validation of the seed is only performed as the new database is started. If it fails, the database is not available and it has the `statusMessage`: `Unable to start database` of the `SHOW DATABASES` command.

```
neo4j@neo4j> SHOW DATABASES;
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
--+
| name    | type       | aliases | access       | address          | role      | writer | requestedStatus
| currentStatus | statusMessage                                                | default | home  |
constituents |
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
--+
| "seed3" | "standard" | []      | "read-write" | "localhost:7682" | "unknown" | FALSE  | "online"
| "offline"     | "Unable to start database `DatabaseId{3fe1a59b[seed3]}`" | FALSE   | FALSE | []
|
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
--+
```

To determine the cause of the problem, it is recommended to look at the `debug.log`.

Seed providers in Neo4j

The seed can be a full backup, a differential backup (see `CloudSeedProvider`, starting from Neo4j 5.26), or a dump from an existing database. When using `CloudSeedProvider`, the URI can point also to a folder which contains a backup chain. The sources of seeds are called *seed providers*.

Backups have a `.backup` extension, while the dump file can have whatever extension the user chooses. For example, the backup file can be called `myBackup.backup`, while the dump can be called `myDump.dump` or also `myBackup.backup`, or any other name.

The mechanism is pluggable, allowing new sources of seeds to be supported (see Java Reference → Implement custom seed providers for more information).

The product has built-in support for seed from a mounted file system (file), FTP server, HTTP/HTTPS server, Amazon S3, Google Cloud Storage (from Neo4j 5.25), and Azure Cloud Storage (from Neo4j 5.25).

Note:

Amazon S3 is supported by default. Starting from Neo4j 5.26, the default support extends also to Google Cloud Storage and Azure Cloud Storage.

To enable other providers, you need to configure `dbms.databases.seed_from_uri_providers`.

FileSeedProvider

Introduced in 5.26

The `FileSeedProvider` supports:

- `file:`

URLConnectionSeedProvider

The `URLConnectionSeedProvider` supports the following:

- `file:` Deprecated in 5.26
- `ftp:`
- `http:`
- `https:`

CloudSeedProvider

Introduced in 5.25

The `CloudSeedProvider` supports:

- `s3:`
- `gs:`
- `azb:`

Starting from Neo4j 5.26, the `CloudSeedProvider` supports using differential backup files as seeds. With the provided differential backup file, the `CloudSeedProvider` searches the directory containing differential backup files for a backup chain ending at the specified differential backup, and then seeds using this backup chain.

> **Note:**
>
> Neo4j uses the AWS SDK v2 to call the APIs on AWS using AWS URLs. Alternatively, you can override the endpoints so that the AWS SDK can communicate with alternative storage systems, such as Ceph, Minio, or LocalStack, using the system variables `aws.endpointUrls3`, `aws.endpointUrlS3`, or `aws.endpointUrl`, or the environments variables `AWS_ENDPOINT_URL_S3` or `AWS_ENDPOINT_URL`.

1. Install the AWS CLI by following the instructions in the AWS official documentation — Install the AWS CLI version 2.
2. Create an S3 bucket and a directory to store the backup files using the AWS CLI:

```
aws s3 mb --region=us-east-1 s3://myBucket
aws s3api put-object --bucket myBucket --key myDirectory/
```

For more information on how to create a bucket and use the AWS CLI, see the AWS official

documentation — Use Amazon S3 with the AWS CLI and Use high-level (s3) commands with the AWS CLI.

3. Verify that the `~/.aws/config` file is correct by running the following command:

```
cat ~/.aws/config
```

The output should look like this:

```
[default]
region=us-east-1
```

4. Configure the access to your AWS S3 bucket by setting the `aws_access_key_id` and `aws_secret_access_key` in the `~/.aws/credentials` file and, if needed, using a bucket policy. For example:

   a. Use `aws configure set aws_access_key_id aws_secret_access_key` command to set your IAM credentials from AWS and verify that the `~/.aws/credentials` is correct:

```
cat ~/.aws/credentials
```

The output should look like this:

```
[default]
aws_access_key_id=this.is.secret
aws_secret_access_key=this.is.super.secret
```

   b. Additionally, you can use a resource-based policy to grant access permissions to your S3 bucket and the objects in it. Create a policy document with the following content and attach it to the bucket. Note that both resource entries are important to be able to download and upload files.

```
{
    "Version": "2012-10-17",
    "Id": "Neo4jBackupAggregatePolicy",
    "Statement": [
        {
            "Sid": "Neo4jBackupAggregateStatement",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetObject",
                "s3:PutObject",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::myBucket/*",
                "arn:aws:s3:::myBucket"
            ]
        }
    ]
}
```

5. Create database from `myBackup.backup`. This backup can be a full backup, a differential backup, or a dump file.

```
CREATE DATABASE foo OPTIONS { existingData: 'use', seedURI: 's3://myBucket/myBackup.backup'
}
```

1. Ensure you have a Google account and a project created in the Google Cloud Platform (GCP).
   a. Install the `gcloud` CLI by following the instructions in the Google official documentation — Install the gcloud CLI.
   b. Create a service account and a service account key using Google official documentation — Create service accounts and Creating and managing service account keys.
   c. Download the JSON key file for the service account.
   d. Set the `GOOGLE_APPLICATION_CREDENTIALS` and `GOOGLE_CLOUD_PROJECT` environment variables to the path of the JSON key file and the project ID, respectively:

   ```
   export GOOGLE_APPLICATION_CREDENTIALS="/path/to/keyfile.json"
   export GOOGLE_CLOUD_PROJECT=YOUR_PROJECT_ID
   ```

   e. Authenticate the `gcloud` CLI with the e-mail address of the service account you have created, the path to the JSON key file, and the project ID:

   ```
   gcloud auth activate-service-account service-account@example.com --key-file
   =$GOOGLE_APPLICATION_CREDENTIALS --project=$GOOGLE_CLOUD_PROJECT
   ```

   For more information, see the Google official documentation — gcloud auth activate-service-account.

   f. Create a bucket in the Google Cloud Storage using Google official documentation — Create buckets.
   g. Verify that the bucket is created by running the following command:

   ```
   gcloud storage ls
   ```

   The output should list the created bucket.

2. Create database from `myBackup.backup`. This backup can be a full backup, a differential backup, or a dump file.

```
CREATE DATABASE foo OPTIONS { existingData: 'use', seedURI: 'gs://myBucket/myBackup.backup'
}
```

1. Ensure you have an Azure account, an Azure storage account, and a blob container.
   a. You can create a storage account using the Azure portal.
   For more information, see the Azure official documentation on Create a storage account.

b. Create a blob container in the Azure portal.
   For more information, see the Azure official documentation on Quickstart: Upload, download, and list blobs with the Azure portal.

2. Install the Azure CLI by following the instructions in the Azure official documentation — Azure official documentation.
3. Authenticate the neo4j or neo4j-admin process against Azure using the default Azure credentials.
   See the Azure official documentation on default Azure credentials for more information.

```
az login
```

   Then you should be ready to use Azure URLs in either neo4j or neo4j-admin.

4. To validate that you have access to the container with your login credentials, run the following commands:

```
# Upload a file:
az storage blob upload --file someLocalFile  --account-name accountName - --container
someContainer --name remoteFileName  --auth-mode login

# Download the file
az storage blob download  --account-name accountName --container someContainer --name
remoteFileName --file downloadedFile --auth-mode login

# List container files
az storage blob list  --account-name someContainer --container someContainer  --auth-mode
login
```

5. Create database from `myBackup.backup`. This backup can be a full backup, a differential backup, or a dump file.

```
CREATE DATABASE foo OPTIONS { existingData: 'use', seedURI:
'azb://myStorageAccount/myContainer/myBackup.backup' }
```

S3SeedProvider

The `S3SeedProvider` supports:

- `s3:` Deprecated in 5.26

> **Note:**
>
> Neo4j 5 comes bundled with necessary libraries for AWS S3 connectivity. Therefore, if you use `S3SeedProvider`, `aws cli` is not required but can be used with the `CloudSeedProvider`.

The `S3SeedProvider` requires additional configuration. This is specified with the `seedConfig` option. This option expects a comma-separated list of configurations. Each configuration value is specified as a name followed by `=` and the value, as such:

```
CREATE DATABASE foo OPTIONS { existingData: 'use', seedURI: 's3://myBucket/myBackup.backup', seedConfig:
'region=eu-west-1' }
```

`S3SeedProvider` also requires passing in credentials. These are specified with the `seedCredentials` option. Seed credentials are securely passed from the Cypher command to each server hosting the database. For this to work, Neo4j on each server in the cluster must be configured with identical keystores. This is identical to the configuration required by remote aliases, see Configuration of DBMS with remote database alias. If this configuration is not performed, the `seedCredentials` option fails.

```
CREATE DATABASE foo OPTIONS { existingData: 'use', seedURI: 's3://myBucket/myBackup.backup', seedConfig:
'region=eu-west-1', seedCredentials: [accessKey];[secretKey] }
```

Where `accessKey` and `secretKey` are provided by AWS.

Seed provider reference

URL scheme	Seed provider	URI example
`file:`	`URLConnectionSeedProvider` Deprecated in 5.26, `FileSeedProvider` Introduced in 5.26	`file:/tmp/backup1.backup` `file:/tmp/backup1.dump`
`ftp:`	`URLConnectionSeedProvider`	`ftp://myftp.com/backups/backup1.backup` `ftp://myftp.com/backups/backup1.dump`
`http:`	`URLConnectionSeedProvider`	`http://myhttp.com/backups/backup1.backup` `http://myhttp.com/backups/backup1.dump`
`https:`	`URLConnectionSeedProvider`	`https://myhttp.com/backups/backup1.backup` `https://myhttp.com/backups/backup1.dump`
`s3:`	`S3SeedProvider` Deprecated in 5.26, `CloudSeedProvider` Introduced in 5.25	`s3://mybucket/backups/backup1.backup` `s3://mybucket/backups/backup1.dump` `s3://mybucket/backups/` (folder containing a backup chain, only used by `CloudSeedProvider`)
`gs:`	`CloudSeedProvider` Introduced in 5.25	`gs://mybucket/backups/backup1.backup` `gs://mybucket/backups/backup1.dump` `gs://mybucket/backups/` (folder containing a backup chain)
`azb:`	`CloudSeedProvider` Introduced in 5.25	`azb://mystorageaccount.blob/backupscontainer/backup1.backup` `azb://mystorageaccount.blob/backupscontainer/backup1.dump` `azb://mystorageaccount.blob/backupscontainer/` (folder containing a backup chain)

List databases

Enterprise Edition

You can list your databases using the Cypher command `SHOW DATABASES`.

Syntax

Command	Syntax

`SHOW DATABASE`	`SHOW { DATABASE[S] name | DATABASE[S] | DEFAULT DATABASE | HOME DATABASE }`
	`[WHERE expression]`
	
	`SHOW { DATABASE[S] name | DATABASE[S] | DEFAULT DATABASE | HOME DATABASE }`
	`YIELD { * | field[, ...] } [ORDER BY field[, ...]] [SKIP n] [LIMIT n]`
	`[WHERE expression]`
	`[RETURN field[, ...] [ORDER BY field[, ...]] [SKIP n] [LIMIT n]]`

`SHOW DATABASES` output

Depending on what you want to see, you can list:

- All databases.
- A particular database.
- The DBMS default database.
- The home database.

These commands return the following columns:

Table 374. Listing databases output

Column	Description	Type
`name`	The name of the database. Default Output	`STRING`
`type`	The type of the database: `system`, `standard`, or `composite`. Default Output	`STRING`
`aliases`	The names of any aliases the database may have. Default Output	`LIST<STRING>`
`access`	The database access mode, either `read-write` or `read-only`. Default Output A database may be described as read-only when using `ALTER DATABASE … SET ACCESS READ ONLY`.	`STRING`
`databaseID`	The database unique ID. A database must be `online` or `deallocating` for this value to be available. For other database states the value will be `NULL`.	`STRING`
`serverID`	The server instance ID.	`STRING`

Column	Description	Type
`address`	Instance address in a clustered DBMS. The default for a standalone database is `neo4j://localhost:7687`. Default Output	`STRING`
`role`	The current role of the database (`primary`, `secondary`, `unknown`). Default Output The value for composite databases is `NULL` because it does not apply to them.	`STRING`
`writer`	`true` for the instance that accepts writes for this database (this instance is the leader for this database in a cluster or this is a standalone instance). Default Output	`BOOLEAN`
`requestedStatus`	The expected status of the database. The value can be either `online` or `offline`. Default Output	`STRING`
`currentStatus`	The actual status of the database. Default Output The possible statuses are: • `online` • `offline` • `starting` • `stopping` • `store copying` • `initial` • `deallocating` • `dirty` • `quarantined` • `unknown` See Database states for more information.	`STRING`
`statusMessage`	A message explaining the status of the database, often explaining why it is not in the correct state. Default Output	`STRING`
`default`	`true` if this is the default database for the DBMS. Default Output Not returned by `SHOW HOME DATABASE` or `SHOW DEFAULT DATABASE`.	`BOOLEAN`
`home`	`true` if this is the home database for the current user. Default Output Not returned by `SHOW HOME DATABASE` or `SHOW DEFAULT DATABASE`.	`BOOLEAN`

Column	Description	Type
`currentPrimariesCount`	Number of primaries for this database reported as running currently. It is the same as the number of rows where `role=primary` and `name=this database`. The value for composite databases is `NULL` because it does not apply to them.[1]	`INTEGER`
`currentSecondariesCount`	Number of secondaries for this database reported as running currently. It is the same as the number of rows where `role=secondary` and `name=this database`. The value for composite databases is `NULL` because it does not apply to them.[1]	`INTEGER`
`requestedPrimariesCount`	The requested number of primaries for this database. May be lower than current if the DBMS is currently reducing the number of copies of the database, or higher if it is currently increasing the number of copies. The value for composite databases is `NULL` because it does not apply to them.	`INTEGER`
`requestedSecondariesCount`	The requested number of secondaries for this database. May be lower than current if the DBMS is currently reducing the number of copies of the database, or higher if it is currently increasing the number of copies. The value for composite databases is `NULL` because it does not apply to them.	`INTEGER`
`creationTime`	The date and time at which the database was created.	`ZONED DATETIME`
`lastStartTime`	The date and time at which the database was last started.	`ZONED DATETIME`
`lastStopTime`	The date and time at which the database was last stopped.	`ZONED DATETIME`
`store`	Information about the storage engine and the store format. The value is a string formatted as `{storage engine}-{store format}-{major version}.{minor version}`. A database must be `online` or `deallocating` for this value to be available. For other database states the value will be `NULL`. The value for composite databases is `NULL` because it does not apply to them.[1]	`STRING`

Column	Description	Type
`lastCommittedTxn`	The ID of the last transaction received. A database must be `online` or `deallocating` for this value to be available. For other database states the value will be `NULL`.	`INTEGER`
`replicationLag`	Number of transactions the current database is behind compared to the database on the primary instance. The lag is expressed in negative integers. In standalone environments, the value is always `0`. A database must be `online` or `deallocating` for this value to be available. For other database states the value will be `NULL`.	`INTEGER`
`constituents`	The names of any constituents the database may have. Applicable only for composite databases. Default Output	`LIST<STRING>`
`options`	The map of options applied to the database. The value for composite databases is `NULL` because it does not apply to them.	`MAP`

The results of the `SHOW DATABASES` command are filtered according to the `ACCESS` privileges of the user. However, some privileges enable users to see additional databases regardless of their `ACCESS` privileges:

- Users with `CREATE/DROP/ALTER DATABASE` or `SET DATABASE ACCESS` privileges can see all standard databases.
- Users with `CREATE/DROP COMPOSITE DATABASE` or `COMPOSITE DATABASE MANAGEMENT` privileges can see all composite databases.
- Users with `DATABASE MANAGEMENT` privilege can see all databases.

If a user has not been granted `ACCESS` privilege to any databases nor any of the above special cases, the command can still be executed but it will only return the `system` database, which is always visible.

Database states

A database's `currentStatus` can be one of the following:

State	Description
`online`	The database is running.
`offline`	The database is not running. If the `statusMessage` column is filled, the database is not running because of a problem.
`starting`	The database is not running, but is about to.

State	Description
`stopping`	The database is not running anymore, but still has not stopped completely. No offline operations (e.g. `load` / `dump`) can be performed yet.
`store copying`	The database is currently being updated from another instance of Neo4j.
`initial`	The database has not yet been created.
`deallocating`	Only applies to databases in a cluster. The database is still online but will eventually be offline due to a transfer of its role in the cluster to a different member. The status is `deallocting` until the transfer is complete, which can take anything from a second to a day or more.
`dirty`	This state implies an error has occurred. The database's underlying store files may be invalid. For more information, consult the `statusMessage` column or the server's logs.
`quarantined`	The database is effectively stopped and its state may not be changed until no longer quarantined. For more information, consult the `statusMessage` column or the server's logs.
`unknown`	This instance of Neo4j does not know the state of this database.

Note:

Note that for failed databases, the `currentStatus` and `requestedStatus` are different. This often implies an error, but **that is not always the case**. For example, a database may take a while to transition from `offline` to `online` due to a performing recovery. Or, during normal operation, a database's `currentStatus` may be transiently different from its `requestedStatus` due to a necessary automatic process, such as one Neo4j instance copying store files from another.

Show all available databases

A summary of all available databases can be displayed using the command `SHOW DATABASES`.

Query

```
SHOW DATABASES
```

Result

```
+-----------------------------------------------------------------------------------------------------------------------------------------------------------+
| name     | type       | aliases                     | access       | address          | role      | writer | requestedStatus | currentStatus | statusMessage | default | home  | constituents |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------+
| "movies" | "standard" | ["films", "motion pictures"] | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"        | "online"      | ""            | FALSE   | FALSE | []           |
| "neo4j"  | "standard" | []                          | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"        | "online"      | ""            | TRUE    | TRUE  | []           |
```

```
| "system" | "system"   | []                          | "read-write" | "localhost:7687" | "primary" |
TRUE   | "online"       | "online"      | ""            | FALSE   | FALSE | []          |
+-------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------+
```

> **Note:**
>
> As of Neo4j 5.3, databases hosted on servers that are offline are also returned by the `SHOW DATABASES` command. For such databases, the `address` column displays `NULL`, the `currentStatus` column displays `unknown`, and the `statusMessage` displays `Server is unavailable`.

Show detailed information for a particular database

In this example, the detailed information for a particular database can be displayed using the command `SHOW DATABASE name YIELD *`. When a `YIELD` clause is provided, the full set of columns is returned.

Query

```
SHOW DATABASE movies YIELD *
```

Result

```
+-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------
---------------+
| name     | type       | aliases                     | access       | databaseID
| serverID                              | address          | role      | writer | requestedStatus |
currentStatus | statusMessage | default | home  | currentPrimariesCount | currentSecondariesCount |
requestedPrimariesCount | requestedSecondariesCount | creationTime              | lastStartTime
| lastStopTime | store                | lastCommittedTxn | replicationLag | constituents | options |
+-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------
---------------+
| "movies" | "standard" | ["films", "motion pictures"] | "read-write" |
"C066801F54B44EA1520F0FE392B4005AABF42D8DD0A5FD09969B955575D287D5" | "e3063985-e2f4-4728-824b-
a7d53779667a" | "localhost:7687" | "primary" | TRUE   | "online"        | "online"      | ""            |
FALSE   | FALSE | 1                     | 0                        | 1                         | 0
| 2023-08-14T10:01:29.074Z | 2023-08-14T10:01:29.074Z | NULL         | "record-aligned-1.1" | 3
| 0              | []           | {}      |
+-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------
---------------+
```

Show the number of databases

The number of distinct databases can be seen using `YIELD` and a `count()` function in the `RETURN` clause.

Query

```
SHOW DATABASES YIELD name
```

```
RETURN count(DISTINCT name) AS count
```

Result

```
+-------+
| count |
+-------+
| 3     |
+-------+
```

By specifying the `name` column and sorting the results by distinct name, only the number of distinct databases are counted, **not** the number of allocations of databases in a clustered environment.

Show the default database

The default database can be seen using the command `SHOW DEFAULT DATABASE`.

Query

```
SHOW DEFAULT DATABASE
```

Result

```
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------+
| name    | type       | aliases | access       | address          | role      | writer | requestedStatus
| currentStatus | statusMessage | constituents |
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------+
| "neo4j" | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | []           |
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------+
```

Show the home database

The home database for the current user can be seen using the command `SHOW HOME DATABASE`.

Query

```
SHOW HOME DATABASE
```

Result

```
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------+
| name    | type       | aliases | access       | address          | role      | writer | requestedStatus
| currentStatus | statusMessage | constituents |
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------+
| "neo4j" | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | []           |
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------+
```

Filter the listed databases

It is also possible to filter and sort the results by using `YIELD`, `ORDER BY`, and `WHERE`.

Query

```
SHOW DATABASES YIELD name, currentStatus, requestedStatus
ORDER BY currentStatus
WHERE name CONTAINS 'o'
```

In this example:

- The number of columns returned has been reduced with the `YIELD` clause.
- The order of the returned columns has been changed.
- The results are ordered by the `currentStatus` column using `ORDER BY`.
- The results have been filtered to only show database names containing `'o'`.

It is also possible to use `SKIP` and `LIMIT` to paginate the results.

Result

```
+-----------------------------------------+
| name     | currentStatus | requestedStatus |
+-----------------------------------------+
| "movies" | "online"      | "online"        |
| "neo4j"  | "online"      | "online"        |
+-----------------------------------------+
```

Alter databases

Enterprise Edition | Not available on Aura

You can modify standard databases using the Cypher command `ALTER DATABASE`.

Syntax

Command	Syntax

`ALTER DATABASE`	`ALTER DATABASE name [IF EXISTS]`
	`{`
	`SET ACCESS {READ ONLY | READ WRITE} |`
	`SET TOPOLOGY n PRIMAR{Y|IES} [m SECONDAR{Y|IES}] |`
	`SET OPTION option value`
	`}`
	`[WAIT [n [SEC[OND[S]]]]|NOWAIT]`
	
	`ALTER DATABASE name [IF EXISTS]`
	`REMOVE OPTION option`
	`[WAIT [n [SEC[OND[S]]]]|NOWAIT]`
	
	Note: There can be multiple `SET OPTION` or `REMOVE OPTION` clauses for different option keys.

Alter database access mode

Not available on Aura

By default, a database has read-write access mode on creation. The database can be limited to read-only mode on creation using the configuration settings `server.databases.default_to_read_only`, `server.databases.read_only`, and `server.database.writable`. For details, see the section on <<, Configuration parameters>>.

A database that was created with read-write access mode can be changed to read-only. To change it to read-only, you can use the `ALTER DATABASE` command with the sub-clause `SET ACCESS READ ONLY`. Subsequently, the database access mode can be switched back to read-write using the sub-clause `SET ACCESS READ WRITE`. Altering the database access mode is allowed at all times, whether a database is online or offline.

If conflicting modes are set by the `ALTER DATABASE` command and the configuration parameters, i.e. one says read-write and the other read-only, the database will be read-only and prevent write queries.

The `WAIT` *sub-clause was added as an option to the* `ALTER DATABASE` *command in Neo4j 5.7.*

> **Note:**
>
> Modifying access mode is only available to standard databases and not composite databases.

Alter database access mode to read-only

To modify the database access mode, use the following command where `customers` is the database name:

```
ALTER DATABASE customers SET ACCESS READ ONLY
```

The database access mode can be seen in the `access` output column of the command `SHOW DATABASES`:

```
SHOW DATABASES yield name, access
```

Result

```
+----------------------------+
| name        | access       |
+----------------------------+
| "customers" | "read-only"  |
| "movies"    | "read-write" |
| "neo4j"     | "read-write" |
| "system"    | "read-write" |
+----------------------------+
```

Alter database access using `IF EXISTS`

`ALTER DATABASE` commands are optionally idempotent, with the default behavior to fail with an error if the database does not exist. Appending `IF EXISTS` to the command ensures that no error is returned and nothing happens if the database does not exist.

```
ALTER DATABASE nonExisting IF EXISTS
SET ACCESS READ WRITE
```

Alter database topology

In a cluster environment, you can use the `ALTER DATABASE` command to change the number of servers hosting a database. For more information, see Managing databases in a cluster.

`ALTER DATABASE` options

The `ALTER DATABASE` command can be used to set or remove specific options for a database.

Table 375. Available options

Key	Value	Description
`txLogEnrichment`	`FULL|DIFF|OFF`	Defines the level of enrichment applied to transaction logs for Change Data Capture (CDC) purposes. For details about enrichment mode, see Change Data Capture Manual → Modify a database's CDC mode.

> **Note:**
>
> The `ALTER DATABASE` command cannot be used to modify the store format of a database. For details about how to change the store format of a database, see Changing the store format of existing databases.

Modify the options set for a database

```
ALTER DATABASE `movies`
SET OPTION txLogEnrichment 'FULL'
```

The database set options can be seen in the `options` output column of the command `SHOW DATABASES`.

```
SHOW DATABASES yield name, options
```

Table 376. Result

name	options
`"customers"`	`{}`
`"movies"`	`{txLogEnrichment: "FULL"}`
`"neo4j"`	`{}`
`"system"`	`{}`
Rows: 4	

Remove the options set for a database

```
ALTER DATABASE `movies`
REMOVE OPTION txLogEnrichment
```

The `REMOVE OPTION` clause removes the specified option from the database using the `ALTER DATABASE` command.

```
SHOW DATABASES YIELD name, options
```

Table 377. Result

name	options
"customers"	{}
"movies"	{}
"neo4j"	{}
"system"	{}
Rows: 4	

Recreate a database

Neo4j 5.24 introduces the `dbms.cluster.recreateDatabase()` procedure, which allows you:

- Change the database store to a specified backup, while keeping all the associated privileges for the database.
- Make your database write-available again after it has been lost (for example, due to a disaster). See Disaster recovery for more information.

Caution:

The recreate procedure works only for real user databases and not for composite databases, or the `system` database.

Remember that the recreate procedure results in downtime while the stores get updated. The time is unbounded and may depend on different factors — for example, the size of the store, network speed, etc.

Syntax

```
dbms.cluster.recreateDatabase(database :: STRING, options = {} :: MAP)
```

Input arguments are the database name and list of the seeding and topology options used for recreating a database.

Table 378. Seeding and topology options of the recreate procedure

Option	Description
seedingServers	A list of possible seeding servers. You can define available servers or provide an empty list. For details, see Use available servers as a seed.
seedURI	External source specified by URI.
primaries	Number of primary allocations for the recreated database. If you set number of primaries without secondaries, then secondaries is set to `0`. For more details, see Change the topology.
secondaries	Number of secondary allocations for the recreated database. You cannot set secondaries without primaries.

Prerequisites and considerations

The database in question can be in an `online` or `offline` state when it is recreated, but a successful operation starts the database regardless of its previous state.

If your database has Change Data Capture (CDC) enabled, the CDC chain will stop when the database is recreated, even though CDC remains enabled in the recreated database. To restore CDC functionality, follow the guide on how to initialize CDC applications from an existing database.

Before recreating a database, any eventual quarantined states need to be addressed. For more information, see Standard databases → Error handling.

You need the `CREATE DATABASE` and `DROP DATABASE` privileges to run the recreate procedure.

Additionally, in a cluster deployment, you have the option to modify the topology during the recreation process. However, note that the store format, access, and enrichment cannot be altered during recreation.

To check if the recreation is successful, use the `SHOW DATABASES` command and verify that all allocations have been started.

Seeding options

The store to be used during the recreation of a database can be defined in different ways. One method uses a backup, while others use available allocations in the cluster.

You can use either `seedURI` or `seedingServers` to specify the source from which the database should be recreated.

- If you define neither, an error is thrown.
- If you define both of them, then `seedingServers` must be an empty list. See Undefined servers with fallback backup for more details.
- If `seedingServers` is not empty and `seedURI` is also defined, an error will occur.

Use backup as a seed

If you provide a URI to a backup or a dump, the stores on all allocations will be replaced by the backup or the dump at the given URI. The new allocations can be put on any `ENABLED` server in the cluster. See Seed from URI for more details.

```
CALL dbms.cluster.recreateDatabase("neo4j", {seedURI: "s3://myBucket/myBackup.backup"});
```

Use available servers as a seed

After the recreation is complete, the database will have the latest data store from the seeding servers.

> **Caution:**
>
> Recreation is based on available remaining stores or specific stores explicitly defined by the user. Stores that are lost or not explicitly specified are excluded from the recreation process. Therefore, if the excluded stores contained more recent data than those used, data loss may occur.

Specified servers

You can specify a set of available servers. The stores on all allocations will be synchronized to the most up-to-date store from the defined servers. The number of defined servers cannot exceed the number of total allocations in the desired topology.

```
CALL dbms.cluster.recreateDatabase("neo4j", {seedingServers: ["serverId1", "serverId2", "serverId3"]});
```

Undefined servers

If you provide an empty list of seeding servers and do not specify a `seedURI`, Neo4j will select all allocations of the database (primaries and secondaries) on currently enabled and non-cordoned servers to act as seeders.

Before running the procedure, ensure that all unavailable servers are cordoned; otherwise, the procedure will fail.

To determine where the database is allocated, use the `SHOW DATABASES` command. To identify all enabled and available servers hosting the required database, run the `SHOW SERVERS` command. Servers have to show `health = Available` and `status = Enabled`. Cordon all unreachable servers by running the `dbms.cluster.cordonServer()` procedure.

Then Neo4j will select the most up-to-date seeder available in the cluster to recreate the database.

```
CALL dbms.cluster.recreateDatabase("neo4j", {seedingServers: []});
```

Undefined servers with fallback backup

If both an empty list of seeding servers and a `seedURI` are provided, Neo4j finds all allocations of the database and use those as seeders. Unavailable servers must be cordoned.

However, if no available servers can be found, the database is recreated based on the backup or the dump specified by the URI. This means the store is replaced by the most up-to-date seeder if available;

otherwise, the backup is used.

```
CALL dbms.cluster.recreateDatabase("neo4j", {seedingServers: [], seedURI:
"s3://myBucket/myBackup.backup"});
```

Change the topology

In a cluster deployment, there is an option to define a new topology when recreating a database. This can be beneficial during a disaster, if enough servers are not available to recreate the database with the original topology. When altering the total number of allocations down during a recreation, it is important to remember that the number of seeding servers cannot exceed the number of total allocations of the database. This also holds true when using recreate with an empty list of seeders. If there are more available servers in the cluster hosting the database than the number of new allocations, the recreation will fail.

```
CALL dbms.cluster.recreateDatabase("neo4j", {seedingServers: [], primaries: 3, secondaries: 0});
```

Delete databases

Enterprise Edition | Not available on Aura

Databases can be deleted by using the command `DROP DATABASE`. Note that all database aliases must be dropped before dropping a database.

Syntax

Command	Syntax
`DROP DATABASE`	`DROP [COMPOSITE] DATABASE name [IF EXISTS] [RESTRICT | CASCADE ALIAS[ES]] [{DUMP|DESTROY} [DATA]] [WAIT [n [SEC[OND[S]]]]|NOWAIT]`

Examples

Delete a database

To delete the database `customers`, run the following command:

```
DROP DATABASE customers
```

> **Note:**
>
> Both standard databases and composite databases can be deleted using this command.

The `DROP DATABASE` command removes a database entirely. Therefore, it no longer shows up in the listing provided by the command `SHOW DATABASES`:

```
SHOW DATABASES YIELD name
```

Result

```
+--------------------+
| name               |
+--------------------+
| "movies"           |
| "neo4j"            |
| "system"           |
+--------------------+
```

Delete a database with `IF EXISTS`

The `DROP DATABASE` command is optionally idempotent, with the default behavior to fail with an error if the database does not exist.

Appending `IF EXISTS` to the command ensures that no error is returned and nothing happens if the database does not exist.

It always returns an error if there is an existing alias that targets the database. In that case, the alias needs to be dropped before dropping the database.

```
DROP DATABASE customers IF EXISTS
```

Delete a database with `DUMP DATA` or `DESTROY DATA`

By appending `DUMP DATA` to the command `DROP DATABASE`, you can create a dump of the store files before deleting the database:

```
DROP DATABASE movies DUMP DATA
```

In Neo4j, dumps can be stored in the directory specified by the `server.directories.dumps.root` setting (by default, the path for storing dumps is `<neo4j-home>/data/dumps`). You can use dumps to create databases using the seed from a URI approach.

The option `DESTROY DATA` explicitly requests the default behavior of the command.

> **Note:**
>
> The dumps produced by `DUMP DATA` are equivalent to those produced by `neo4j-admin database dump`. You can also restore them using the `neo4j-admin database load` command.

Delete a database with `IF EXISTS` and `DUMP DATA`/`DESTROY DATA`

The options `IF EXISTS` and `DUMP DATA`/`DESTROY DATA` can also be combined.

An example could look like this:

```
DROP DATABASE customers IF EXISTS DUMP DATA
```

Delete a database with local database aliases targeting it

There are two ways of dropping a database that is the target of local database aliases:

- Drop the local database aliases first, then use `DROP DATABASE name` to drop the database. Remote database aliases targeting the database do not affect the deletion of the database and therefore dos not need to be dropped beforehand.
- Use `DROP DATABASE name CASCADE ALIASES` to also drop the local database aliases targeting it while dropping the database. If any of the dropped database aliases are constituents of composite databases, those composite databases will not be dropped. This command does not affect the remote database aliases targeting the database being dropped. They will simply no longer resolve their targets as if they were created targeting a nonexistent database.

Using `CASCADE ALIASES` requires the `DROP ALIAS` privilege. For more information about the privilege, see ALIAS MANAGEMENT privileges.

Example 32. Drop a database and the local database alias targeting it

The following example creates a database `movies` and a local database alias `films` targeting it:

```
CREATE DATABASE movies
CREATE ALIAS films FOR DATABASE movies
```

Then, the database `movies` and the local database alias `films` can be dropped using the following command:

```
DROP DATABASE movies CASCADE ALIASES
```

The option `RESTRICT` explicitly requests the default behavior of the command.

Note:

For standard databases, the aliases that are dropped when using the `CASCADE ALIASES` option can be found in the `aliases` column of `SHOW DATABASE`.

Delete a database with `RESTRICT`/`CASCADE ALIASES` and other command parts

The options `RESTRICT`/`CASCADE ALIASES` can also be combined with `IF EXISTS` and `DUMP DATA`/`DESTROY DATA`. For example:

```
DROP DATABASE movies IF EXISTS CASCADE ALIASES DUMP DATA
```

`WAIT` options

Enterprise Edition | Not available on Aura

Aside from `SHOW DATABASES`, all database management commands accept an optional `WAIT`/`NOWAIT` sub-clause. The `WAIT`/`NOWAIT` sub-clause allows you to specify a time limit in which the command must complete and return.

The options are:

- `WAIT n SECONDS` - Returns once completed or when the specified time limit of `n` seconds is up.
- `WAIT` - Returns once completed or when the default time limit of 300 seconds is up.
- `NOWAIT` - Returns immediately.

A command using a `WAIT` sub-clause automatically commits the current transaction when it executes successfully, as the command needs to run immediately for it to be possible to `WAIT` for it to complete. Any subsequent commands executed are therefore performed in a new transaction. This is different from the usual transactional behavior, and for this reason, it is recommended that these commands be run in their own transaction. The default behavior is `NOWAIT`, so if no clause is specified the transaction behaves normally and the action is performed in the background post-commit.

The `WAIT` sub-clause was added as an option to the `ALTER DATABASE` command in Neo4j 5.7.

Note:

A command with a `WAIT` clause may be interrupted whilst it is waiting to complete. In this event, the command will continue to execute in the background and will not be aborted.

Configuration parameters

Not available on Aura

Configuration parameters are defined in the neo4j.conf file.

The following configuration parameters are applicable for managing databases:

Parameter name	Description
`initial.dbms.default_database`	Name of the default database for the Neo4j instance. The database is created if it does not exist when the instance starts. **Default value:** `neo4j` **Note:** In a clustered setup, the value of `initial.dbms.default_database` is only used to set the initial default database. To change the default database at a later point, see Change the default database. **Note:** Be aware that the automatically created *initial* default database may have a different topology to the default configuration values. See Default database in a cluster for more information.
`dbms.max_databases`	Maximum number of databases that can be used in a Neo4j single instance or cluster. The number includes all the online and offline databases. The value is an integer with a minimum value of 2. Enterprise Edition **Default value:** `100` **Note:** Once the limit has been reached, it is not possible to create any additional databases. Similarly, if the limit is changed to a number lower than the total number of existing databases, no additional databases can be created.
`server.databases.default_to_read_only`	Default mode of all databases. If this setting is set to `true` all existing and new databases will be in read only mode, and so will prevent write queries. **Default value:** `false`

Parameter name	Description
`server.databases.read_only`	List of database names for which to prevent write queries. This set can contain also not yet existing databases, but not the `system` database. **Note:** Regardless of settings of `server.databases.default_to_read_only`, `server.databases.read_only` and `server.databases.writable` the `system` database will never be read-only and will always accept write queries. **Note:** Another way of preventing writes is to set the database access to read-only using the ALTER DATABASE command. Example configuration: `server.databases.read_only=["foo", "bar"]`
`server.databases.writable`	List of database names for which to accept write queries. This set can contain also not yet existing databases. The value of this setting is ignored if `server.databases.default_to_read_only` is set to `false`. If a database name is present in both sets, the database will be read-only and prevent write queries. **Tip:** If most of your databases would read-only with a few exceptions, it can be easier to set `server.databases.default_to_read_only` to `true`, and then put the names of the non read-only databases into `server.databases.writable`. Example configuration: `server.databases.writable=["foo", "bar"]`

Note:

Although it is possible to achieve the same goal, i.e. set a database to read-only, both by using the Cypher command `ALTER DATABASE` and by using configuration parameters in `neo4j.conf`, it is important to understand the difference between the two. `ALTER DATABASE foo SET ACCESS READ ONLY` effectively sets the database `foo` to read-only across the entire DBMS.

Using configuration parameters is more subtle and allows you to configure access on each instance separately, in case of a cluster for example. If you use `server.databases.default_to_read_only` *all databases on that instance* are set to read-only.

If both the Cypher command and the configuration parameters are used and they contain conflicting information, the database in question is set to read-only.

Error handling

Enterprise Edition

When running the database management queries, such as `CREATE DATABASE`, it is possible to encounter errors.

Observing errors

Because database management operations are performed asynchronously, these errors may not be returned immediately upon query execution. Instead, you must monitor the output from the `SHOW DATABASE` command; particularly the `statusMessage` and `currentStatus` columns.

Fail to create a database

```
neo4j@system> CREATE DATABASE foo;
```

```
0 rows available after 108 ms, consumed after another 0 ms
```

```
neo4j@system> SHOW DATABASE foo;
```

In standalone mode:

```
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
| name   | type       | aliases | access       | address          | role      | writer | requestedStatus |
currentStatus | statusMessage             | default | home  | constituents |
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
| "foo"  | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"        |
"dirty"       | "File system permissions" | FALSE   | FALSE | []           |
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+

1 rows available after 4 ms, consumed after another 1 ms
```

In a cluster:

```
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
| name   | type       | aliases | access       | address          | role      | writer | requestedStatus |
currentStatus | statusMessage             | default | home  | constituents |
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
```

```
| "foo"  | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"        |
"online"      | ""                          | FALSE   | FALSE | []          |
| "foo"  | "standard" | []      | "read-write" | "localhost:7688" | "primary" | FALSE  | "online"        |
"online"      | ""                          | FALSE   | FALSE | []          |
| "foo"  | "standard" | []      | "read-write" | "localhost:7689" | "primary" | FALSE  | "online"        |
"dirty"       | "File system permissions" | FALSE   | FALSE | []          |
+-------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------+

3 row available after 100 ms, consumed after another 6 ms
```

Database states

A database management operation may fail for a number of reasons. For example, if the file system instance has incorrect permissions, or Neo4j itself is misconfigured. As a result, the contents of the `statusMessage` column in the `SHOW DATABASE` query results may vary significantly.

However, databases may only be in one of a select number of states:

- `online`
- `offline`
- `starting`
- `stopping`
- `store copying`
- `initial`
- `deallocating`
- `dirty`
- `quarantined`
- `unknown`

For more details about the various states, see Database states. Most often, when a database management operation fails, Neo4j attempts to transition the database in question to the `offline` state. If the system is certain that no store files have yet been created, it transitions the database to `initial` instead. Similarly, if the system suspects that the store files underlying the database are invalid (incomplete, partially deleted, or corrupt), then it transitions the database to `dirty`.

Retrying failed operations

Database management operations may be safely retried in the event of failure. However, these retries are not guaranteed to succeed, and errors may persist through several attempts.

Note:

If a database is in the `quarantined` state, retrying the last operation will not work.

Retry to start a database

```
neo4j@system> START DATABASE foo;
```

```
0 rows available after 108 ms, consumed after another 0 ms
```

```
neo4j@system> SHOW DATABASE foo;
```

```
+-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+
| name   | type       | aliases | access       | address          | role      | writer | requestedStatus |
currentStatus | statusMessage             | default | home  | constituents |
+-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+
| "foo"  | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"        |
"offline"     | "File system permissions" | FALSE   | FALSE | []           |
+-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+

1 rows available after 4 ms, consumed after another 1 ms
```

After investigating and addressing the underlying issue, you can start the database again and verify that it is running properly:

```
neo4j@system> START DATABASE foo;
```

```
0 rows available after 108 ms, consumed after another 0 ms
```

```
neo4j@system> SHOW DATABASE foo;
```

```
+-------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------+
| name     | type       | aliases | access       | address          | role      | writer | requestedStatus
| currentStatus | statusMessage | default | home  | constituents |
+-------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------+
| "foo"    | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | FALSE   | FALSE | []           |
+-------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------+

1 rows available after 4 ms, consumed after another 1 ms
```

If repeated retries of a command have no effect, or if a database is in a `dirty` state, you may drop and recreate the database, as detailed in Create database.

> **Note:**
>
> When running `DROP DATABASE` as part of an error handling operation, you can also append `DUMP DATA` to the command. It produces a database dump that can be further examined and potentially repaired.

Quarantined databases

There are two ways to get a database into a `quarantined` state:

- By using the `dbms.quarantineDatabase()` procedure locally to isolate a specific database. The procedure must be executed on the instance whose copy of the database you want to quarantine. A reason for that can be, for example, when a database is unable to start on a given instance due to a file system permissions issue with the volume where the database is located or when a recently started database begins to log errors. The quarantine state renders the database inaccessible on that instance and prevents its state from being changed, for example, with the `START DATABASE` command.

  > **Note:**
  >
  > If running in a cluster, database management commands such as `START DATABASE foo` will still take effect on the instances which have **not** quarantined `foo`.

- When a database encounters a severe error during its normal run, which prevents it from a further operation, Neo4j stops that database and brings it into a `quarantined` state. Meaning, it is not possible to restart it with a simple `START DATABASE` command. You have to execute `CALL dbms.quarantineDatabase(databaseName, false)` on the instance with the failing database in order to lift the quarantine.

After lifting the quarantine, the instance will automatically try to bring the database to the desired state.

> **Note:**
>
> It is recommended to run the quarantine procedure over the `bolt://` protocol rather than `neo4j://`, which may route requests to unexpected instances.

Syntax:

```
CALL dbms.quarantineDatabase(databaseName,setStatus,reason)
```

Arguments:

Name	Type	Description
`databaseName`	String	The name of the database that will be put into or removed from quarantine.
`setStatus`	Boolean	`true` for placing the database into quarantine; `false` for lifting the quarantine.
`reason`	String	(Optional) The reason for placing the database in quarantine.

Returns:

Name	Type	Description
`databaseName`	String	The name of the database.
`quarantined`	String	Actual state.
`result`	String	Result of the last operation. The result contains the user, the time, and the reason for the quarantine.

> **Note:**
>
> The `dbms.quarantineDatabase()` procedure replaces `dbms.cluster.quarantineDatabase()`, which has been deprecated in Neo4j 4.3 and will be removed with the next major version.

Quarantine a database

```
neo4j@system> CALL dbms.quarantineDatabase("foo",true);
```

```
+-------------------------------------------------------------------------------------+
| databaseName | quarantined | result                                                 |
+-------------------------------------------------------------------------------------+
| "foo"        | TRUE        | "By neo4j at 2020-10-15T15:10:41.348Z: No reason given" |
+-------------------------------------------------------------------------------------+

3 row available after 100 ms, consumed after another 6 ms
```

Check if a database is quarantined

```
neo4j@system> SHOW DATABASE foo;
```

```
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------+
| name  | type       | aliases | access       | address          | role      | writer | requestedStatus |
currentStatus | statusMessage                                                | default | home  | constituents |
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------+
| "foo" | "standard" | []      | "read-write" | "localhost:7688" | "unknown" | FALSE  | "online"        |
"quarantined" | "By neo4j at 2020-10-15T15:10:41.348Z: No reason given" | FALSE   | FALSE | []           |
| "foo" | "standard" | []      | "read-write" | "localhost:7689" | "primary" | FALSE  | "online"        |
"online"      | ""                                                           | FALSE   | FALSE | []           |
| "foo" | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"        |
"online"      | ""                                                           | FALSE   | FALSE | []           |
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------+

3 row available after 100 ms, consumed after another 6 ms
```

> **Note:**
>
> A `quarantined` state is persisted for user databases. This means that if a database is quarantined, it will remain so even if the Neo4j process is restarted. You can remove it only by running the `dbms.quarantineDatabase()` procedure on the instance where the quarantined database is located, passing `false` for the `setStatus` parameter.

The one exception to this rule is for the built-in `system` database. Any quarantine for that database is removed automatically after instance restart.

[1] This change applies to versions 5.26.5 and later.

Database aliases

Naming rules for database aliases

Enterprise Edition | AuraDB Business Critical | AuraDB Virtual Dedicated Cloud

Database alias names are subject to the standard Cypher restrictions on valid identifiers. See Cypher Manual → Naming rules and recommendations.

The following naming rules apply:

- A name is a valid identifier.
- Name length can be up to 65534 characters.
- Names cannot end with dots.
- Unquoted dots signify that the database alias belongs to a composite database, separating the composite database name and the alias name.
- Names that begin with an underscore or with the prefix `system` are reserved for internal use.
- Non-alphabetic characters, including numbers, symbols, dots, and whitespace characters, can be used in names, but must be quoted using backticks.

The name restrictions and escaping rules apply to all the different database alias commands.

> **Note:**
>
> Having dots (`.`) in the database alias names is not recommended. This is due to the difficulty of determining if a dot is part of the database alias name or a delimiter for a database alias in a composite database. For more details, see names and escaping for database aliases targeting composite databases.

Managing database aliases for standard databases

Enterprise Edition | AuraDB Business Critical | AuraDB Virtual Dedicated Cloud

There are two kinds of database aliases: local and remote. A local database alias can only target a database within the same DBMS. A remote database alias may target a database from another Neo4j DBMS. When a query is run against a database alias, it will be redirected to the target database. The home database for users can be set to an alias, which will be resolved to the target database on use.

This page describes managing database aliases for standard databases. Local and remote database aliases can also be created as part of a composite database. For more information, see Managing database aliases in composite databases.

A local database alias can be used in all other Cypher commands in place of the target database. Please note that the local database alias will be resolved while executing the command. Privileges are defined on

the database, and not the local database alias.

A remote database alias can be used for connecting to a database of a remote Neo4j DBMS, `USE` clauses, setting a user's home database, and defining the access privileges to the remote database. Remote database aliases require configuration to safely connect to the remote target, which is described in Connecting remote databases. It is not possible to impersonate a user on the remote database or to execute an administration command on the remote database via a remote database alias.

Database aliases can be created and managed using a set of Cypher administration commands executed against the `system` database. The required privileges are described in the The DBMS ALIAS MANAGEMENT privileges. When connected to the DBMS over Bolt, administration commands are automatically routed to the `system` database.

> **Note:**
>
> If a transaction modifies a database alias, other transactions concurrently executing against that alias may be aborted and rolled back for safety. This prevents issues such as a transaction executing against multiple target databases for the same alias.

List database aliases

You can list all available database aliases using the `SHOW ALIASES FOR DATABASE` command. The command returns a table of all database aliases, whether they belong to a composite database or not.
If you need more details, you can append the command with `YIELD *`. The `YIELD *` clause returns the full set of columns. The required privileges are described in the The DBMS ALIAS MANAGEMENT privileges.

`SHOW ALIASES FOR DATABASE` will produce a table of database aliases with the following columns:

Column	Description	Type
`name`	The fully qualified name of the database alias. Default Output	`STRING`
`composite`	The name of the composite database this alias belongs to, or `null` if the alias does not belong to a composite database. Default Output	`STRING`

Column	Description	Type
`database`	The name of the target database. Default Output This column is filtered according to the `ACCESS` privileges of the user. However, some privileges enable users to see additional databases regardless of their `ACCESS` privileges: • Users with the `CREATE DATABASE`, `DROP DATABASE`, `ALTER DATABASE`, or `SET DATABASE ACCESS` privileges can see all standard databases. • Users with the `CREATE COMPOSITE DATABASE`, `DROP COMPOSITE DATABASE`, or `COMPOSITE DATABASE MANAGEMENT` privileges can see all composite databases. • Users with the `DATABASE MANAGEMENT` privilege can see all databases. • Users can always see the `system` database. If a user has not been granted the `ACCESS` privilege to the target database and none of the above special cases apply, then it is not visible to the user and this column will be `null`.	`STRING`
`location`	The location of the database, either `local` or `remote`. Default Output	`STRING`
`url`	Target location or `null` if the target is local. Default Output	`STRING`
`user`	User connecting to the remote database or `null` if the target database is local. Default Output	`STRING`
`driver`	The driver options for connection to the remote database or `null` if the target database is local. List of driver settings allowed for remote database aliases.	`MAP`
`properties`	Any properties set on the database alias.	`MAP`

The detailed information for a particular database alias can be displayed using the command `SHOW ALIASES FOR DATABASE YIELD *`. When a `YIELD *` clause is provided, the full set of columns is returned.

Show all aliases for a database

A summary of all available database aliases can be displayed using the command `SHOW ALIASES FOR DATABASE`. This command will show database aliases for both standard and composite databases.

Query

```
SHOW ALIASES FOR DATABASE
```

Result

```
+-----------------------------------------------------------------------------------------+
| name              | composite | database  | location | url                       | user    |
+-----------------------------------------------------------------------------------------+
| "films"           | NULL      | "movies"  | "local"  | NULL                      | NULL    |
| "motion pictures" | NULL      | "movies"  | "local"  | NULL                      | NULL    |
| "movie scripts"   | NULL      | "scripts" | "remote" | "neo4j+s://location:7687" | "alice" |
```

```
+-----------------------------------------------------------------------------------------+
```

Show specific aliases for databases

To list just one database alias, the `SHOW ALIASES` command takes an alias name;

Query

```
SHOW ALIAS films FOR DATABASES
```

Result

```
+-----------------------------------------------------------+
| name    | composite | database | location | url  | user |
+-----------------------------------------------------------+
| "films" | NULL      | "movies" | "local"  | NULL | NULL |
+-----------------------------------------------------------+
```

Show detailed aliases information for a database

Query

```
SHOW ALIASES FOR DATABASE YIELD *
```

Result

```
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
------------+
| name              | composite | database  | location | url                       | user    | driver
| properties                |
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
------------+
| "films"           | NULL      | "movies"  | "local"  | NULL                      | NULL    | NULL
| {}                        |
| "motion pictures" | NULL      | "movies"  | "local"  | NULL                      | NULL    | NULL
| {namecontainsspace: TRUE} |
| "movie scripts"   | NULL      | "scripts" | "remote" | "neo4j+s://location:7687" | "alice" |
{connection_pool_idle_test: PT2M, connection_pool_max_size: 10, logging_level: "INFO", ssl_enforced: TRUE,
connection_pool_acquisition_timeout: PT1M, connection_timeout: PT5S, connection_max_lifetime: PT1H} | {}
|
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
------------+
```

Show `count` of aliases for a database

The number of database aliases can be seen using a `count()` aggregation with `YIELD` and `RETURN`.

Query

```
SHOW ALIASES FOR DATABASE YIELD *
RETURN count(*) as count
```

Result

```
+-------+
| count |
+-------+
| 3     |
+-------+
```

Show filtered aliases information for a database

It is possible to filter and sort the results by using `YIELD`, `ORDER BY`, and `WHERE`.

Query

```
SHOW ALIASES FOR DATABASE YIELD name, url, database
ORDER BY database
WHERE name CONTAINS 'e'
```

In this example:

- The number of columns returned has been reduced with the `YIELD` clause.
- The order of the returned columns has been changed.
- The results are ordered by the `database` column using `ORDER BY`.
- The results have been filtered to only show database alias names containing `'e'`.

It is also possible to use `SKIP` and `LIMIT` to paginate the results.

Result

```
+-------------------------------------------------------------+
| name              | url                       | database  |
+-------------------------------------------------------------+
| "motion pictures" | NULL                      | "movies"  |
| "movie scripts"   | "neo4j+s://location:7687" | "scripts" |
+-------------------------------------------------------------+
```

Create database aliases

Database aliases can be created using `CREATE ALIAS`. The command supports the creation of both local and remote database aliases. For more information on local and remote database aliases as part of a composite database, see Create database aliases in composite databases.

The required privileges are described in the The DBMS ALIAS MANAGEMENT privileges.

> **Note:**
>
> Database alias names are subject to the rules specified in the Alias names section.

Create database aliases for local databases

A local database alias targets a database within the same DBMS.

Query

```
CREATE ALIAS `northwind` FOR DATABASE `northwind-graph-2021`
```

When you create a local database alias, it shows up in the `aliases` column provided by the command `SHOW DATABASES` and in the `SHOW ALIASES FOR DATABASE` command.

Query

```
SHOW DATABASE `northwind`
```

Result

```
+-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------+
| name                   | type       | aliases       | access       | address          | role      |
writer | requestedStatus | currentStatus | statusMessage | default | home  | constituents |
+-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------+
| "northwind-graph-2021" | "standard" | ["northwind"] | "read-write" | "localhost:7687" | "primary" | TRUE
| "online"        | "online"      | ""            | FALSE   | FALSE | []           |
+-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------+
```

Query

```
SHOW ALIAS `northwind` FOR DATABASE
```

Result

```
+-----------------------------------------------------------------------+
| name        | composite | database               | location | url  | user |
+-----------------------------------------------------------------------+
| "northwind" | NULL      | "northwind-graph-2021" | "local"  | NULL | NULL |
+-----------------------------------------------------------------------+
```

Use `IF EXISTS` or `OR REPLACE` when creating database aliases

The `CREATE ALIAS` command is optionally idempotent, with the default behavior to fail with an error if the database alias already exists. There are two ways to circumvent this behavior.

First, appending `IF NOT EXISTS` to the command ensures that no error is returned and nothing happens should the database alias already exist.

Query

```
CREATE ALIAS `northwind` IF NOT EXISTS  FOR DATABASE `northwind-graph-2021`
```

Second, adding `OR REPLACE` to the command results in any existing database being deleted and a new one being created.

Query

```
CREATE OR REPLACE ALIAS `northwind` FOR DATABASE `northwind-graph-2021`
```

This is equivalent to running `DROP ALIAS `northwind` IF EXISTS FOR DATABASE` followed by `CREATE ALIAS `northwind` FOR DATABASE `northwind-graph-2021`++`.

> **Note:**
>
> The `IF NOT EXISTS` and `OR REPLACE` parts of these commands cannot be used together.

Set properties for local database aliases

Local database aliases can also be given properties. These properties can then be used in queries with the `graph.propertiesByName()` function.

Query

```
CREATE ALIAS `northwind-2022`
FOR DATABASE `northwind-graph-2022`
PROPERTIES { newestNorthwind: true, index: 3 }
```

The properties are then shown in the `SHOW ALIASES FOR DATABASE YIELD …` command.

Query

```
SHOW ALIAS `northwind-2022` FOR DATABASE YIELD name, properties
```

Result

```
+-------------------------------------------------------+
| name             | properties                         |
+-------------------------------------------------------+
| "northwind-2022" | {index: 3, newestnorthwind: TRUE} |
+-------------------------------------------------------+
```

Create database aliases for remote databases

A database alias can target a remote database by providing an URL and the credentials of a user on the remote Neo4j DBMS. See Configuring remote database aliases for the necessary configurations.

As with local database aliases, creating remote database aliases allows `IF NOT EXISTS` and `OR REPLACE` clauses. Both check for any remote or local database aliases.

Query

```
CREATE ALIAS `remote-northwind` FOR DATABASE `northwind-graph-2020`
AT "neo4j+s://location:7687"
USER alice
PASSWORD 'example_secret'
```

When you create a database alias targeting a remote database, its details can be shown with the `SHOW ALIASES FOR DATABASE` command.

Query

```
SHOW ALIAS `remote-northwind`
```

```
FOR DATABASE
```

Result

```
+-----------------------------------------------------------------------------------------------------
-+
| name               | composite | database               | location | url                       | user
|
+-----------------------------------------------------------------------------------------------------
-+
| "remote-northwind" | NULL      | "northwind-graph-2020" | "remote" | "neo4j+s://location:7687" | "alice"
|
+-----------------------------------------------------------------------------------------------------
-+
```

Create remote database aliases with driver settings

It is possible to override the default driver settings per database alias, which are used for connecting to the remote database.

This is the list of the allowed driver settings for remote database aliases:

- ssl_enforced (Default: `true`) — SSL for remote database alias drivers is configured through the target URL scheme. If `ssl_enforced` is set to true, a secure URL scheme is enforced. This will be validated when the command is executed.
- connection_timeout — for details, see dbms.routing.driver.connection.connect_timeout
- connection_max_lifetime — for details, see dbms.routing.driver.connection.max_lifetime.
- connection_pool_acquisition_timeout — for details, see dbms.routing.driver.connection.pool.acquisition_timeout.
- connection_pool_idle_test — for details, see dbms.routing.driver.connection.pool.idle_test.
- connection_pool_max_size — for details, see dbms.routing.driver.connection.pool.max_size.
- logging_level (For details, see dbms.routing.driver.logging.level)

The driver settings are set in the `DRIVER` clause of the `CREATE ALIAS` or `ALTER ALIAS` commands. For example, the following query creates a remote database alias using driver settings `connection_timeout` and `connection_pool_max_size` for connecting to the remote database `northwind-graph-2020`:

Query

```
CREATE ALIAS `remote-with-driver-settings` FOR DATABASE `northwind-graph-2020`
AT "neo4j+s://location:7687"
USER alice
PASSWORD 'example_secret'
DRIVER {
  connection_timeout: duration({minutes: 1}),
  connection_pool_max_size: 10
}
```

When a database alias targeting a remote database has been created, its details can be shown with the `SHOW ALIASES FOR DATABASE` command.

Query

```
SHOW ALIAS `remote-with-driver-settings` FOR DATABASE YIELD *
```

Result

```
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------+
| name                          | composite | database              | location | url
| user    | driver                                                  | properties |
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------+
| "remote-with-driver-settings" | NULL      | "northwind-graph-2020" | "remote" |
"neo4j+s://location:7687" | "alice" | {connection_pool_max_size: 10, connection_timeout: PT1M} | {}
|
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------+
```

Set properties for remote database aliases

Just as the local database aliases, the remote database aliases can be given properties. These properties can then be used in queries with the `graph.propertiesByName()` function.

Query

```
CREATE ALIAS `remote-northwind-2021` FOR DATABASE `northwind-graph-2021` AT 'neo4j+s://location:7687'
USER alice PASSWORD 'password'
PROPERTIES { newestNorthwind: false, index: 6 }
```

The properties are then shown in the `SHOW ALIASES FOR DATABASE YIELD …` command.

Query

```
SHOW ALIAS `remote-northwind-2021` FOR DATABASE YIELD name, properties
```

Result

```
+-----------------------------------------------------------+
| name                    | properties                      |
+-----------------------------------------------------------+
| "remote-northwind-2021" | {index: 6, newestnorthwind: FALSE} |
+-----------------------------------------------------------+
```

Alter database aliases

You can alter both local and remote database aliases using the `ALTER ALIAS` command. For all aliases, the command allows you to change the target database and properties of the database alias. For remote aliases, the command also allows you to change the URL, user credentials, or driver settings of the database alias. The required privileges are described in the The DBMS ALIAS MANAGEMENT privileges. Only the clauses used will be altered.

Note:

Local database aliases cannot be altered to remote aliases, or vice versa.

Alter local database aliases

Example of altering a local database alias target.

Query

```
ALTER ALIAS `northwind`
SET DATABASE TARGET `northwind-graph-2021`
```

When a local database alias has been altered, it will show up in the `aliases` column for the target database provided by the command `SHOW DATABASES`.

Query

```
SHOW DATABASE `northwind-graph-2021`
```

Result

```
+-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------+
| name                   | type       | aliases       | access       | address          | role      |
writer | requestedStatus | currentStatus | statusMessage | default | home  | constituents |
+-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------+
| "northwind-graph-2021" | "standard" | ["northwind"] | "read-write" | "localhost:7687" | "primary" | TRUE
| "online"        | "online"      | ""            | FALSE   | FALSE | []           |
+-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------+
```

Alter remote database aliases

Example of altering a remote database alias target.

Query

```
ALTER ALIAS `remote-northwind` SET DATABASE
TARGET `northwind-graph-2020` AT "neo4j+s://other-location:7687"
```

Alter remote credentials and driver settings for remote database aliases

Example of altering a remote database alias credentials and driver settings.

Query

```
ALTER ALIAS `remote-with-driver-settings` SET DATABASE
USER bob
PASSWORD 'new_example_secret'
DRIVER {
  connection_timeout: duration({ minutes: 1}),
  logging_level: 'debug'
}
```

Important:

All driver settings are replaced by the new ones. In this case, by not repeating the driver setting `connection_pool_max_size` the value will be deleted and fall back to the default

value.

Remove custom driver settings from remote database aliases

Example of altering a remote database alias to remove all custom driver settings.

Query

```
ALTER ALIAS `movie scripts` SET DATABASE
DRIVER {}
```

Alter properties for local and remote database aliases

Examples of altering local and remote database alias properties.

Query

```
ALTER ALIAS `motion pictures` SET DATABASE PROPERTIES { nameContainsSpace: true, moreInfo: 'no, not
really' }
```

Query

```
ALTER ALIAS `movie scripts` SET DATABASE PROPERTIES { nameContainsSpace: true }
```

The updated properties can then be used in queries with the `graph.propertiesByName()` function.

Use `IF EXISTS` when altering database aliases

The `ALTER ALIAS` command is optionally idempotent, with the default behavior to fail with an error if the database alias does not exist. Appending `IF EXISTS` to the command ensures that no error is returned and nothing happens should the alias not exist.

Query

```
ALTER ALIAS `no-alias` IF EXISTS SET DATABASE TARGET `northwind-graph-2021`
```

```
(no changes, no records)
```

Delete database aliases

You can delete both local and remote database aliases using the `DROP ALIAS` command. The required privileges are described in the The DBMS ALIAS MANAGEMENT privileges.

Delete local database aliases

Delete a local database alias.

Query

```
DROP ALIAS `northwind` FOR DATABASE
```

When a database alias has been deleted, it will no longer show up in the `aliases` column provided by the command `SHOW DATABASES`.

Query

```
SHOW DATABASE `northwind-graph-2021`
```

Result

```
+-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+
| name                   | type       | aliases | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage | default | home  | constituents |
+-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+
| "northwind-graph-2021" | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   |
"online"        | "online"      | ""            | FALSE   | FALSE | []           |
+-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+
```

Delete remote database aliases

Delete a remote database alias.

Query

```
DROP ALIAS `remote-northwind` FOR DATABASE
```

When a database alias has been deleted, it will no longer show up in the `SHOW ALIASES FOR DATABASE` command.

Query

```
SHOW ALIASES `remote-northwind` FOR DATABASE
```

Result

```
+-----------------------------------------------------+
| name | composite | database | location | url | user |
+-----------------------------------------------------+
+-----------------------------------------------------+
```

Use `IF EXISTS` when deleting database aliases

The `DROP ALIAS` command is optionally idempotent, with the default behavior to fail with an error if the database alias does not exist. Inserting `IF EXISTS` after the alias name ensures that no error is returned and nothing happens should the alias not exist.

Query

```
DROP ALIAS `northwind` IF EXISTS FOR DATABASE
```

```
(no changes, no records)
```

Managing database aliases in composite databases

Enterprise Edition | Not available on Aura

Both local and remote database aliases can be created as part of a composite database.

List database aliases in composite databases

Available database aliases in composite databases can be seen using `SHOW ALIASES FOR DATABASE`. The name of the composite database a particular database alias appears in the returned `composite` column.

The required privileges are described in the The DBMS ALIAS MANAGEMENT privileges.

Query

```
SHOW ALIASES FOR DATABASE
```

Result

```
+-----------------------------------------------------------------------------------------------------+
| name              | composite | database        | location | url                       | user    |
+-----------------------------------------------------------------------------------------------------+
| "library.romance" | "library" | "romance-books" | "remote" | "neo4j+s://location:7687" | "alice" |
| "library.sci-fi"  | "library" | "sci-fi-books"  | "local"  | NULL                      | NULL    |
+-----------------------------------------------------------------------------------------------------+
```

For a description of all the returned columns of this command, and for ways in which the `SHOW ALIASES FOR DATABASE` command can be filtered for aliases, see list aliases for standard databases.

Create database aliases in composite databases

Both local and remote database aliases can be part of a composite database.

The database alias consists of two parts, separated by a dot: the namespace and the alias name.

The namespace must be the name of the composite database.

Query

```
CREATE ALIAS garden.flowers
FOR DATABASE `perennial-flowers`
```

Query

```
CREATE ALIAS garden.trees
FOR DATABASE trees AT 'neo4j+s://location:7687'
USER alice PASSWORD 'password'
```

When a database alias has been created in a composite database, it will show up in the `constituents` column provided by the command `SHOW DATABASES` and in the `SHOW ALIASES FOR DATABASE` command.

Query

```
SHOW DATABASE garden YIELD name, type, constituents
```

Result

```
+-----------------------------------------------------------+
| name     | type        | constituents                     |
+-----------------------------------------------------------+
| "garden" | "composite" | ["garden.flowers", "garden.trees"] |
+-----------------------------------------------------------+
```

Query

```
SHOW ALIASES FOR DATABASE
WHERE composite = 'garden'
```

Result

```
+-----------------------------------------------------------------------------------------------------+
| name             | composite | database            | location | url                       | user    |
+-----------------------------------------------------------------------------------------------------+
| "garden.flowers" | "garden"  | "perennial-flowers" | "local"  | NULL                      | NULL    |
| "garden.trees"   | "garden"  | "trees"             | "remote" | "neo4j+s://location:7687" | "alice" |
+-----------------------------------------------------------------------------------------------------+
```

Database aliases cannot target a composite database.

Query

```
CREATE ALIAS yard FOR DATABASE garden
```

Error message

```
Failed to create the specified database alias 'yard': Database 'garden' is composite.
```

From 5.26 onwards, the error message also contains the GQLSTATUS code `42NA6` and the status description `error: syntax error or access rule violation - invalid alias target. Aliases are not allowed to target composite databases.`

Alter local and remote database aliases in composite databases

Local and remote database aliases belonging to a composite database can be altered using the `ALTER ALIAS` command. This is the same command that is used for altering aliases that are not part of a composite database.

Query

```
ALTER ALIAS garden.flowers SET DATABASE PROPERTIES { perennial: true }
```

Query

```
ALTER ALIAS garden.trees SET DATABASE TARGET updatedTrees AT 'neo4j+s://location:7687' PROPERTIES {
treeVersion: 2 }
```

The updated properties can then be used in queries with the `graph.propertiesByName()` function.

The changes for all database aliases will show up in the `SHOW ALIASES FOR DATABASE` command.

Query

```
SHOW ALIASES FOR DATABASE YIELD *
```

Result

```
+-------------------------------------------------------------------------------------------------------------
-------------------------+
| name              | composite | database            | location | url                       | user    |
driver | properties        |
+-------------------------------------------------------------------------------------------------------------
-------------------------+
| "garden.flowers"  | "garden"  | "perennial-flowers" | "local"  | NULL                      | NULL    |
NULL   | {perennial: TRUE} |
| "garden.trees"    | "garden"  | "updatedtrees"      | "remote" | "neo4j+s://location:7687" | "alice" |
{}     | {treeversion: 2}  |
| "library.romance" | "library" | "romance-books"     | "remote" | "neo4j+s://location:7687" | "alice" |
{}     | {}                |
| "library.sci-fi"  | "library" | "sci-fi-books"      | "local"  | NULL                      | NULL    |
NULL   | {}                |
+-------------------------------------------------------------------------------------------------------------
-------------------------+
```

Delete database aliases in composite databases

To delete an alias in a composite database, use the `DROP ALIAS FOR DATABASE` command. This is the same command that is used for deleting aliases that are not part of a composite database.

Query

```
DROP ALIAS garden.flowers FOR DATABASE
```

When a database alias has been deleted, it will no longer show up in the `SHOW ALIASES FOR DATABASE` command.

Query

```
SHOW ALIASES FOR DATABASE
```

Result

```
+-------------------------------------------------------------------------------------------------+
| name              | composite | database        | location | url                       | user    |
+-------------------------------------------------------------------------------------------------+
| "garden.trees"    | "garden"  | "updatedtrees"  | "remote" | "neo4j+s://location:7687" | "alice" |
| "library.romance" | "library" | "romance-books" | "remote" | "neo4j+s://location:7687" | "alice" |
| "library.sci-fi"  | "library" | "sci-fi-books"  | "local"  | NULL                      | NULL    |
+-------------------------------------------------------------------------------------------------+
```

Additionally, deleted aliases will no longer appear in the `constituents` column for the `SHOW DATABASE` command.

Query

```
SHOW DATABASE garden YIELD name, type, constituents
```

Result

```
+-----------------------------------------+
| name     | type        | constituents     |
+-----------------------------------------+
| "garden" | "composite" | ["garden.trees"] |
+-----------------------------------------+
```

Database alias names and escaping

Naming database aliases in composite databases follows the same rule as naming aliases for standard databases. However, when it comes to escaping names using backticks, there are some additional things to consider:

Quoting database alias and composite database names

The composite database name and the database alias name need to be quoted individually. Backticks may be added regardless of whether the name contains special characters or not, so it is good practice to always backtick both names, e.g. `` `composite`.`alias` ``.

The following example creates a database alias named `my alias with spaces` as a constituent in the composite database named `my-composite-database-with-dashes`:

Query

```
CREATE ALIAS `my-composite-database-with-dashes`.`my alias with spaces` FOR DATABASE `northwind-graph`
```

When not quoted individually, a database alias with the full name `my alias with.dots and spaces` gets created instead:

Query

```
CREATE ALIAS `my alias with.dots and spaces` FOR DATABASE `northwind-graph`
```

Handling multiple dots

Database alias names may also include dots. Though these always need to be quoted in order to avoid ambiguity with the composite database and database alias split character.

Query

```
CREATE ALIAS `my.alias.with.dots` FOR DATABASE `northwind-graph`
```

Query

```
CREATE ALIAS `my.composite.database.with.dots`.`my.other.alias.with.dots` FOR DATABASE `northwind-graph`
```

Single dots and local database aliases

> **Note:**
>
> As of Neo4j 5.26.9, this feature is no longer deprecated.

There is a special case for local database aliases with a single dot without any existing composite database. If a composite database `some` exists, the query below will create a database alias named `alias` within the composite database `some`. If no such database exists, however, the same query will instead create a database alias named `some.alias`:

Query

```
CREATE ALIAS some.alias FOR DATABASE `northwind-graph`
```

Handling parameters

When using parameters, names cannot be quoted. When the given parameter includes dots, the first dot will be considered the divider for the composite database.

Consider the query with parameter:

Parameters

```
{
  "aliasname": "mySimpleCompositeDatabase.myAlias"
}
```

Query

```
CREATE ALIAS $aliasname FOR DATABASE `northwind-graph`
```

If the composite database `mysimplecompositedatabase` exists, then a database alias `myalias` will be created in that composite database. If no such composite database exists, then a database alias `mysimplecompositedatabase.myalias` will be created.

On the contrary, a database alias `myalias` cannot be created in composite `mycompositedatabase.withdot` using parameters. Consider the same query but with the following parameter:

Parameters

```
{
  "aliasname": "myCompositeDatabase.withDot.myAlias"
}
```

Since the first dot will be used as a divider, the command will attempt to create the database alias `withdot.myalias` in the composite database `mycompositedatabase`. If `mycompositedatabase` does not exist, the command will create a database alias with the name `mycompositedatabase.withdot.myalias`, which is not part of any composite database.

In these cases, it is recommended to avoid parameters and explicitly quote the composite database name

and alias name separately to avoid ambiguity.

Handling parameters

Further special handling with parameters is needed for database aliases and similarly named composite databases.

Consider the setup:

Query

```
CREATE COMPOSITE DATABASE foo
CREATE ALIAS `foo.bar` FOR DATABASE `northwind-graph`
```

The alias `foo.bar` does not belong to the composite database `foo`.

Dropping this alias using parameters fails with an error about a missing alias:

Parameters

```
{
  "aliasname": "foo.bar"
}
```

Query

```
DROP ALIAS $aliasname FOR DATABASE
```

Error message

```
Failed to delete the specified database alias 'foo.bar': Database alias does not exist.
```

Had the composite database `foo` not existed, the database alias `foo.bar` would have been dropped.

In these cases, it is recommended to avoid parameters and explicitly quote the composite database name and alias name separately to avoid ambiguity.

Configuring remote database aliases

A remote database alias can be used to provide connection to one or more graphs, on one or more remote standalone servers or clusters.

Although remote database aliases do not store any data, they enable users or applications to perform queries on remote databases as if they were on the local DBMS server. All configurations can be done using administrative commands on a running system. Any changes are automatically synchronized across all instances of a cluster.

The following steps describe the setup required to define a remote database alias both for local and remote DBMSs. They are also useful should there be any changes in the name of the databases or the location of standalone servers and cluster instances. Additionally, you will also find information here on best practices and additional guidance on any changes in the configuration.

Setup example

In this example, *Alice* is an administrator and *Carol* is a user who needs access to a database managed by *Bob*:

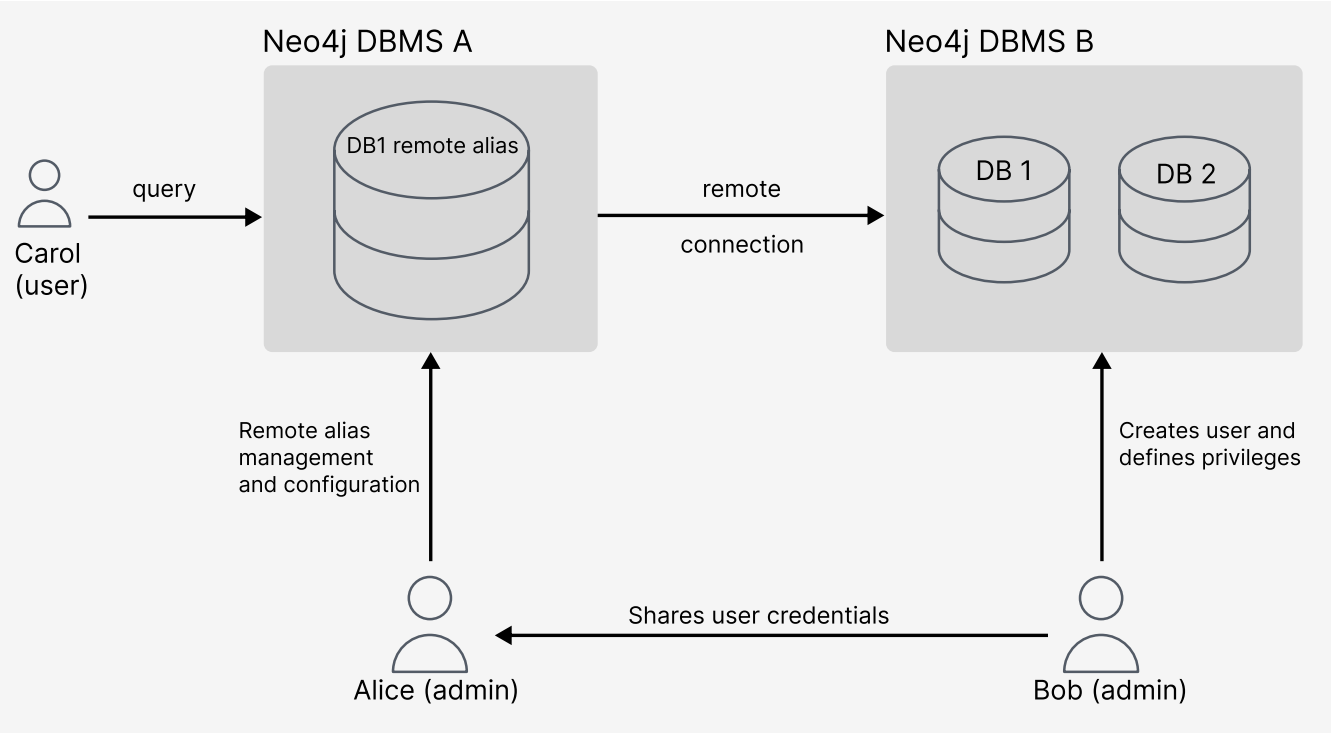


Figure 5. Overview of the required remote database alias setup

A remote alias defines:

- Which user of the remote **DBMS B** is used.
- Where the remote database is located.
- How to connect to the remote database using driver settings.

> **Note:**
>
> A remote database alias is only accessible to users with appropriate privileges. In the example above, *Bob* is the administrator responsible for deciding which databases (`Db1` or `Db2`) the remote aliases can write and/or read. Meanwhile, *Alice* is the administrator that assigns who has access to the privileges set by *Bob*. In the example, *Alice* will assign that access to *Carol*.
>
> See DBMS privileges for more information.

Carol can use her own regular credentials to access the remote database `Db1` in DBMS after *Alice* assigns this privilege to her user profile. This configuration will also allow *Carol* to access `Db2` in **DBMS B**. If the administrators decide this should not be the case, then *Bob* must define the appropriate privileges. See Authentication and authorization for further information.

Configure a remote DBMS (*Bob*)

In the suggested example, there are two administrators: *Alice* and *Bob*. *Bob* is the administrator responsible for the setup of the remote DBMS, which includes the following steps:

1. Create the user profile to share with *Alice*. Currently, only user and password-based authentication (e.g. native authentication) are supported.
2. Define the permissions for the user. If you don't want this user to access `Db2`, here is where you set it.
3. Securely transmit the credentials to *Alice*, setting up the link to database `Db1`. It is recommended to create a custom role to track all users shared on a remote connection, so that they remain trackable.

```
CREATE USER alice SET PASSWORD 'secretpassword'
CREATE ROLE remote
GRANT ACCESS ON DATABASE neo4j TO remote
GRANT MATCH {*} ON GRAPH neo4j TO remote
GRANT ROLE remote TO alice
```

In this case, *Bob* must do the required setup for the SSL framework and check whether the database accepts a non-local connection if required.

```
# accept non-local connections
server.default_listen_address=0.0.0.0

# configure ssl for bolt
dbms.ssl.policy.bolt.enabled=true
dbms.ssl.policy.bolt.base_directory=certificates/bolt
dbms.ssl.policy.bolt.private_key=private.key
dbms.ssl.policy.bolt.public_certificate=public.crt
dbms.ssl.policy.bolt.client_auth=NONE

# enforcing ssl connection
server.bolt.tls_level=REQUIRED
```

Configure a DBMS with a remote database alias (*Alice*)

As *Alice*, you need to generate an encryption key. In this case, the credentials of a user of DBMS B are reversibly encrypted and stored in the system database of DBMS A.

Since the algorithm used is AES/GCM, an AES encryption key needs to be provided. It should have length 256, and be stored in a password-protected keystore, of format pkcs12.

The key can be generated by using the following keytool command in your terminal, which is included in Java Platform, Standard Edition:

```
keytool -genseckey -keyalg aes -keysize 256 -storetype pkcs12 -keystore [keystore-name] -alias [key-name]
-storepass [keystore-password]
```

> **Note:**
>
> It is recommended to generate the keystore on the same Java version on which Neo4j is run, as the supported encryption algorithms may vary. For details on the version of Java required by Neo4j, see System requirements → Java.

The following configuration is necessary for creating a remote database alias:

Configuration	Description
`dbms.security.keystore.path`	The absolute path to the keystore file, including the file name.
`dbms.security.keystore.password`	The password to the keystore file. Use Command expansion to set the password.
`dbms.security.key.name`	The name of the secret key

Caution:

To prevent unauthorized access, the keystore file must be stored in a trusted location. This is the main way to protect the encrypted passwords which will be stored on the system database. It shouldn't be accessible to any user except for the administrator and `neo4j`, for whom the keystore file must be readable.

In a cluster, *Alice* needs to share the same keystore file over all instances.

For example, these would be valid additions to the config when using the suggested keytool command:

```
dbms.security.keystore.path=/home/secure-folder/keystore-name.pkcs12
dbms.security.keystore.password=$(conf/password.sh)
dbms.security.key.name=key-name
```

Where `password.sh` might look like this:

```
#!/bin/bash
echo "$KEYSTORE_PASSWORD_ENVIRONMENT_VARIABLE"
```

Additionally, don't forget to change the permissions of the configuration file, and start Neo4j with the command expansion flag:

```
chmod 640 conf/neo4j.conf
bin/neo4j start --expand-commands
```

Manage remote database aliases

You can use the alias commands to manage remote database aliases. In this case, it is strongly recommended to connect to a remote database alias with a secured connection.

Please note that only client-side SSL is supported. By default, remote aliases require a secured URI scheme such as `neo4j+s`. This can be disabled by setting the driver setting `ssl_enforced` to `false`.

For example, the following command can be used to create a remote database alias:

```
CREATE ALIAS `remote-neo4j` FOR DATABASE `neo4j` AT "neo4j+s://location:7687" USER alice PASSWORD
'secretpassword'
```

In order to do so, either database management or alias management privileges are required. The

permission to create an alias can be granted using the following command:

```
GRANT CREATE ALIAS ON DBMS TO administrator
```

Here is how to grant the `ACCESS` privileges to use the remote database alias:

```
GRANT ACCESS ON DATABASE `remote-neo4j` TO role
```

> **Note:**
>
> If a transaction modifies an alias (e.g. changing the database targeted on DBMS B), other transactions concurrently executing against that alias may be aborted and rolled back for safety. This prevents issues such as a transaction executing against multiple target databases for the same alias.

Change the encryption key

If the encryption key in the keystore is changed, the encrypted credentials for existing remote database aliases will need to be updated as they will no longer be readable.

> **Note:**
>
> If there is a failure when reading the keystore file, investigate the `debug.log` to find out which parameter is the source of the problem. In case it is not possible to connect to the remote alias after its creation, verify its settings by connecting to the remote database at https://browser.neo4j.io/ or at your local browser.

Connect to remote database aliases

A user can connect to a remote database alias the same way they would do to a database. This includes:

- Connecting directly to the remote database alias.
- The Cypher `USE` clause enables a user to query a remote database alias that they are not directly connected to:

```
USE `remote-neo4j` MATCH (n) RETURN *
```

- Connecting to a remote database alias as a home database. This needs to be set by Administrator A. See more about User Management.

```
ALTER USER alice SET HOME DATABASE `remote-neo4j`
```

> **Note:**
>
> Remote alias transactions will not be visible in `SHOW TRANSACTIONS` on DBMS A. However,

they can be accessed and terminated on the remote database when connecting with the same user.

Composite databases

Concepts

Enterprise Edition | Not available on Aura

A Composite database is a special type of database introduced in Neo4j 5. It supersedes the previous Fabric implementation in Neo4j 4.x.

In Neo4j 5, fabric has been expanded as a concept and now refers to the architectural design of a unified system that provides a single access point to local or distributed graph data.

Composite databases are the means to access this partitioned data or graphs with a single Cypher query.

Composite databases **do not store data** independently. They contain *database aliases* that target the local or remote databases (the so-called *constituents*) that constitute the fabric setup. Local database aliases target databases within the same DBMS, while remote database aliases target databases from another Neo4j DBMS. For more information, see Managing database aliases in composite databases.

Composite databases are managed using Cypher administrative commands. Therefore, you can create them as any other database in Neo4j standalone and cluster deployments without needing to deploy a dedicated proxy server (as with Fabric in Neo4j 4). For a detailed example of how to create a Composite database and add database aliases to it, see Set up and query composite databases.

Composite databases cannot guarantee compatibility between constituents from different major versions of Neo4j. Therefore, all constituents should belong to the same major version. If a new feature is introduced, its availability will be limited to the intersection of features available for all constituents.

The following table summarizes the similarities and differences between Composite databases in Neo4j 5 and Fabric in Neo4j 4:

Table 379. Composite database Neo4j 5 vs. Fabric Neo4j 4

	Composite database (Neo4j 5)	Fabric (Neo4j 4)
Data access	Gives access to graphs found on other databases (local or remote).	
Data storage	Does not store data independently.	
Deployment	Both standalone and cluster deployments.	Both standalone and cluster deployments. However, in a cluster deployment, the *fabric* proxy must be deployed on a dedicated machine.
Configuration	Managed using Cypher commands. Composite databases are created with the `CREATE COMPOSITE DATABASE` command and database aliases are added with `CREATE ALIAS ...`.	Managed through configuration settings. The Neo4j DBMSs that host the same *fabric* database must have the same configuration settings. The configuration must be always kept in sync.

	Composite database (Neo4j 5)	Fabric (Neo4j 4)
Sharding an existing database	With the help of the `neo4j-admin copy` command.	
Security credentials	Composite databases use the same user credentials as the database aliases.	The Neo4j DBMSs that host the same Fabric virtual database must have the same user credentials. Any change of password on a machine that is part of Fabric must be kept in sync and applied to all the Neo4j DBMSs that are part of Fabric.
Database management	Does not support database management. Any database management commands, index and constraint management commands, or user and security management commands must be issued directly to the DBMSs and databases, not the Composite databases.	
Transactions	Only transactions with queries that read from multiple graphs, or read from multiple graphs and write to a single graph, are allowed.	
Neo4j embedded	Not available when Neo4j is used as an embedded database in Java applications. It can be used only in a typical client/server mode when users connect to a Neo4j DBMS from their client application or tool via Bolt or HTTP protocol.	

The main concepts that are relevant to understand when working with Composite databases are:

Data Federation

Data federation is when your data is in two **disjoint graphs** with **different labels and relationship types**. For example, you have data about users with their location and data about the users' posts on different forums, and you want to query them together.

Data Sharding

Data sharding is when you have two graphs that share the **same model** (same labels and relationship types) but contain **different data**. For example, you can deploy shards on separate servers, splitting the load on resources and storage. Or, you can deploy shards in different locations, to be able to manage them independently or split the load on network traffic. An existing database can be sharded with the help of the `neo4j-admin database copy` command. For an example, see Sharding data with the copy command.

Connecting data across graphs

Because relationships cannot span across graphs, to query your data, you have to federate the graphs by using a *proxy node* modeling pattern, where nodes with a specific label must be present in both federated domains.

In one of the graphs, nodes with that specific label contain all the data related to that label, while in the other graph, the same label is associated with a proxy node that only contains the `<node>ID` property. The `<node>ID` property allows you to link data across the graphs in this federation.

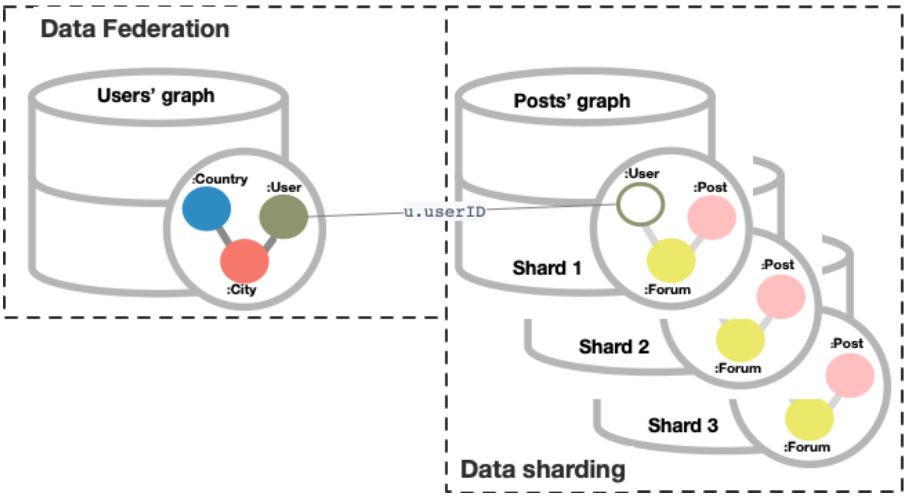


Figure 6. Data federation and sharding

> **Tip:**
>
> For a step-by-step tutorial on setting up and using a Composite database with federated and sharded data, see Tutorials → Setting up and using a composite database.

Create, start, and stop composite databases

Enterprise Edition | Not available on Aura

Composite databases are managed using Cypher® administrative commands. Note that it is not possible to modify access options or database topologies for composite databases as these are inherited from the constituent databases. For information about modifying access options, see Alter database access mode. For information about about topologies for databases, see Create databases in a cluster.

Drivers and client applications connect to composite databases just like standard databases. For more information, see the manuals for the different Neo4j drivers and applications.

Create a composite database

Composite databases can be created using `CREATE COMPOSITE DATABASE`.

Composite database names are subject to the same rules as standard databases. One difference is however that the deprecated syntax using dots without enclosing the name in backticks is not available. Both dots and dashes need to be enclosed within backticks when using composite databases.

> **Note:**
>
> Having dots (`.`) in the composite database names is not recommended. This is due to the difficulty of determining if a dot is part of the composite database name or a delimiter for a database alias in a composite database.

Query

```
CREATE COMPOSITE DATABASE inventory
```

When a composite database has been created, it shows up in the listing provided by the command `SHOW DATABASES`.

Query

```
SHOW DATABASES
```

Result

```
+-----------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------+
| name        | type        | aliases            | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage | default | home  | constituents       |
+-----------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------+
| "inventory" | "composite" | []                 | "read-only"  | "localhost:7687" | NULL      | FALSE  |
"online"        | "online"      | ""            | FALSE   | FALSE | []                 |
| "library"   | "composite" | []                 | "read-only"  | "localhost:7687" | NULL      | FALSE  |
"online"        | "online"      | ""            | FALSE   | FALSE | ["library.sci-fi"] |
| "neo4j"     | "standard"  | []                 | "read-write" | "localhost:7687" | "primary" | TRUE   |
"online"        | "online"      | ""            | TRUE    | TRUE  | []                 |
| "sci-fi"    | "standard"  | ["library.sci-fi"] | "read-write" | "localhost:7687" | "primary" | TRUE   |
"online"        | "online"      | ""            | FALSE   | FALSE | []                 |
| "system"    | "system"    | []                 | "read-write" | "localhost:7687" | "primary" | TRUE   |
"online"        | "online"      | ""            | FALSE   | FALSE | []                 |
+-----------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------+
```

For a full description of the columns returned by this command, and how to sort the results by specific columns, see List databases.

To create database aliases in the composite database, give the composite database as a namespace for the alias. For information about creating aliases in composite databases, see Managing aliases in composite databases.

Use `IF NOT EXISTS` or `OR REPLACE` when creating composite databases

The `CREATE COMPOSITE DATABASE` command is optionally idempotent, with the default behavior to fail with an error if the database already exists. There are two ways to circumvent this behavior.

First, appending `IF NOT EXISTS` to the command ensures that no error is returned and nothing happens should the database already exist.

Query

```
CREATE COMPOSITE DATABASE inventory IF NOT EXISTS
```

This will not create a new composite database, because a composite database with the name `inventory` already exists.

Second, adding `OR REPLACE` to the command will result in any existing database being deleted and a new one being created.

Query

```
CREATE OR REPLACE COMPOSITE DATABASE inventory
```

This is equivalent to running `DROP DATABASE inventory IF EXISTS` followed by `CREATE COMPOSITE DATABASE inventory`.

The behavior of `IF NOT EXISTS` and `OR REPLACE` apply to both standard and composite databases (e.g. a composite database may replace a standard database or another composite database).

> **Note:**
>
> The `IF NOT EXISTS` and `OR REPLACE` parts of these commands cannot be used together.

Stop composite databases

Databases can be stopped using the command `STOP DATABASE`.

Query

```
STOP DATABASE inventory
```

> **Note:**
>
> Both standard databases and composite databases can be stopped using this command.

The status of the stopped database can be seen using the command `SHOW DATABASE name`.

Query

```
SHOW DATABASE inventory YIELD name, requestedStatus, currentStatus
```

Result

```
+-----------------------------------------------+
| name        | requestedStatus | currentStatus |
+-----------------------------------------------+
| "inventory" | "offline"       | "offline"     |
+-----------------------------------------------+
```

Start composite databases

Databases can be started using the command `START DATABASE`.

Query

```
START DATABASE inventory
```

> **Note:**

Both standard databases and composite databases can be started using this command.

The status of the started database can be seen using the command `SHOW DATABASE name`.

Query

```
SHOW DATABASE inventory YIELD name, requestedStatus, currentStatus
```

Result

```
+-----------------------------------------------+
| name        | requestedStatus | currentStatus |
+-----------------------------------------------+
| "inventory" | "online"        | "online"      |
+-----------------------------------------------+
```

List composite databases

Enterprise Edition | Not available on Aura

The `type` column returned by the `SHOW DATABASES` command displays which databases are composite.

Query

```
SHOW DATABASES
```

Result

```
+-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------+
| name      | type        | aliases            | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage | default | home  | constituents       |
+-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------+
| "library" | "composite" | []                 | "read-only"  | "localhost:7687" | NULL      | FALSE  |
"online"        | "online"      | ""            | FALSE   | FALSE | ["library.sci-fi"] |
| "neo4j"   | "standard"  | []                 | "read-write" | "localhost:7687" | "primary" | TRUE   |
"online"        | "online"      | ""            | TRUE    | TRUE  | []                 |
| "sci-fi"  | "standard"  | ["library.sci-fi"] | "read-write" | "localhost:7687" | "primary" | TRUE   |
"online"        | "online"      | ""            | FALSE   | FALSE | []                 |
| "system"  | "system"    | []                 | "read-write" | "localhost:7687" | "primary" | TRUE   |
"online"        | "online"      | ""            | FALSE   | FALSE | []                 |
+-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------+
```

For a description of all the returned columns of this command, ways in which the `SHOW DATABASE` command can be filtered, and details about the privileges required for the command, see List standard databases.

For composite databases, the `constituents` column is particularly interesting as it lists the aliases that make up the composite database.

Query

```
SHOW DATABASE library YIELD name, constituents
```

Result

```
+-------------------------------+
| name      | constituents      |
+-------------------------------+
| "library" | ["library.sci-fi"] |
+-------------------------------+
```

Delete composite databases

Enterprise Edition | Not available on Aura

There are two ways of deleting a composite database with constituent database aliases (local or remote) by either dropping the constituent database aliases first and then deleting the composite database, or deleting the composite database while also dropping the constituent database aliases.

Delete a composite database

Before deleting a composite database, you must ensure that it is not in use by any database aliases. If the composite database is in use, you must first drop the aliases that reference it. For more information, see Delete database aliases in composite databases.

You can delete composite databases using either the command `DROP COMPOSITE DATABASE name` or the more general one `DROP DATABASE name`. However, keep in mind that the first command targets only composite databases, while the second one targets any database.

Query

```
DROP COMPOSITE DATABASE inventory
```

Delete a composite database while dropping its constituents

Introduced in 5.24

You can use the `CASCADE ALIASES` option of the `DROP COMPOSITE DATABASE` Cypher command to drop the constituent database aliases while deleting the composite database.

> **Note:**
>
> This operation does not deletes the actual target databases of the constituent database aliases.

The `CASCADE ALIASES` option is useful when you want to delete a composite database and its constituent database aliases in one step. Using `CASCADE ALIASES` requires the `DROP ALIAS` privilege. For more information about this privilege, see ALIAS MANAGEMENT privileges.

Example 33. Drop a composite database and its consitutent alias

This example shows how to create a composite database `movies` and a database alias `movies.sweden`

for the database `swedish-movies` and then delete the alias `sweden` and the composite database `movies`.

Create a composite database `movies` and a database alias `movies.sweden` for the database `swedish-movies`

```
CREATE COMPOSITE DATABASE movies
CREATE ALIAS movies.sweden FOR DATABASE `swedish-movies`
```

Delete the composite database `movies` while also dropping the alias `movies.sweden`

```
DROP COMPOSITE DATABASE movies CASCADE ALIASES
```

This behavior is the same for the more general command `DROP DATABASE name` when using it to drop a composite database.

Note:

For composite databases, the aliases that are dropped when using the `CASCADE ALIASES` option can be found in the `constituents` column of `SHOW DATABASE`.

Set up and query composite databases

Enterprise Edition | Not available on Aura

The examples featured in this section make use of the two Cypher clauses: `CALL {}`.

Graph set-up

The following set-up is required to recreate the examples on this page:

Create a standard database `movies2022`

```
CREATE DATABASE movies2022
```

Create a composite database `cineasts`

```
CREATE COMPOSITE DATABASE cineasts
```

Create database alias `cineasts.latest` for a local database in a composite database

```
CREATE ALIAS `cineasts`.`latest`
  FOR DATABASE movies2022
```

Create database alias `cineasts.upcoming` for a remote database in a composite database

```
CREATE ALIAS `cineasts`.`upcoming`
  FOR DATABASE upcoming
  AT 'neo4j+s://location:7687'
  USER neo4j
```

```
  PASSWORD 'password'
```

For more information about composite databases and database aliases in composite databases, see Concepts, and Managing database aliases in composite databases.

Graph selection

Queries submitted to a composite database may contain several `USE` clauses that direct different parts of the query to different constituent graphs.

Each constituent graph is named after the alias that introduces it into the Composite database.

Query a single graph

Example 34. Reading and returning data from a single graph

```
USE cineasts.latest
MATCH (movie:Movie)
RETURN movie.title AS title
```

The `USE` clause at the beginning of the query selects the `cineasts.latest` graph for all the subsequent clauses. `MATCH` is performed on that graph.

Query multiple graphs

Example 35. Reading and returning data from two graphs

```
USE cineasts.latest
MATCH (movie:Movie)
RETURN movie.title AS title
  UNION
USE cineasts.upcoming
MATCH (movie:Movie)
RETURN movie.title AS title
```

The first part of the `UNION` query selects the `cineasts.latest` graph and the second part selects the `cineasts.upcoming` graph.

Dynamic graph access

Queries can also select constituent graphs dynamically, using the form `USE graph.byName(graphName)`.

Example 36. Reading and returning data from dynamically selected graphs

```
UNWIND ['cineasts.latest', 'cineasts.upcoming'] AS graphName
CALL {
  USE graph.byName(graphName)
  MATCH (movie:Movie)
  RETURN movie
}
RETURN movie.title AS title
```

In the example above, the part of the query accessing graph data, `MATCH (movie:Movie)`, is wrapped in a sub-query with a dynamic `USE` clause. `UNWIND` is used to get the names of your graphs, each on one row. The `CALL {}` sub-query executes once per input row. In this case, once selecting `cineasts.latest`, and once selecting `cineasts.upcoming`.

Listing graphs

The built-in function `graph.names()` returns a list containing the names of all constituent graphs on the current Composite database.

Example 37. The `graph.names()` function

```
UNWIND graph.names() AS graphName
RETURN graphName
```

```
+---------------------+
| graphName           |
+---------------------+
| "cineasts.latest"   |
| "cineasts.upcoming" |
+---------------------+
```

The names returned by this function can be used for dynamic graph access.

Example 38. Reading and returning data from all graphs

```
UNWIND graph.names() AS graphName
CALL {
  USE graph.byName(graphName)
  MATCH (movie:Movie)
  RETURN movie
}
RETURN movie.title
```

Query result aggregation

Example 39. Getting the earliest release year of all movies from all graphs

```
UNWIND graph.names() AS graphName
CALL {
  USE graph.byName(graphName)
  MATCH (movie:Movie)
  RETURN movie.released AS released
}
RETURN min(released) AS earliest
```

The sub-query returns the `released` property of each movie, from each constituent graph. The `RETURN` at the end of the main query aggregates across the full result to calculate the global minimum.

Correlated subqueries

This query finds all movies in `cineasts.upcoming` that are to be released in the same month as the longest movie in `cineasts.latest`.

Example 40. Correlated subquery

```
CALL {
  USE cineasts.latest
  MATCH (movie:Movie)
  RETURN movie.releasedMonth AS monthOfLongest
    ORDER BY movie.runningTime DESC
    LIMIT 1
}
CALL {
  USE cineasts.upcoming
  WITH monthOfLongest
  MATCH (movie:Movie)
  WHERE movie.releasedMonth = monthOfLongest
  RETURN movie
}
RETURN movie
```

The first part of the query finds the movie with the longest running time from `cineasts.latest`, and returns its release month. The second part of the query finds all movies in `cineasts.upcoming` that fulfill your condition and returns them. The sub-query imports the `monthOfLongest` variable using `WITH monthOfLongest`, to make it accessible.

Updates

Composite database queries can perform updates to constituent graphs.

Example 41. Constituent graph update

```
USE cineasts.upcoming
CREATE (:Movie {title: 'Dune: Part Two'})
```

Note:

Updates can only be performed on a single constituent graph per transaction.

Example 42. Multi-graph update will fail

```
UNWIND graph.names() AS graphName
CALL {
  USE graph.byName(graphName)
  CREATE (:Movie {title: 'The Flash'})
}
```

```
Writing to more than one database per transaction is not allowed.
```

Limitations

Queries on Composite databases have a few limitations.

Graph accessing operations

Consider a Composite database query:

```
UNWIND graph.names() AS graphName
CALL {
  USE graph.byName(graphName)
  MATCH (movie:Movie)
  RETURN movie
}
RETURN movie
```

Here the outer clauses, i.e. the `UNWIND`, the `CALL` itself, and the final `RETURN`, appear in the root scope of the query, without a specifically chosen graph. Clauses or expressions in scopes where no graph has been specified must not be graph-accessing.

The following Composite database query is invalid because `[p=(movie)->() | p] AS paths` is a graph-accessing operation in the root scope of the query:

```
UNWIND graph.names() AS graphName
CALL {
  USE graph.byName(graphName)
  MATCH (movie:Movie)
  RETURN movie
}
RETURN [p=(movie)-->() | p] AS paths
```

See examples of graph-accessing operations:

- `RETURN 1 + 2 AS number`
- `WITH node.property AS val`

Nested `USE` clauses

An inner scope must use the same graph as its outer scope:

```
USE cineasts.latest
MATCH (n)
CALL {
  USE cineasts.upcoming
  MATCH (m)
  RETURN m
}
RETURN n, m
```

```
Nested subqueries must use the same graph as their parent query.
Attempted to access graph cineasts.upcoming
"    USE cineasts.upcoming"
     ^
```

Sub-queries without a `USE` clause can be nested. They inherit the specified graph from the outer scope.

```
CALL {
  USE cineasts.upcoming
```

```
  CALL {
    MATCH (m:Movie)
    RETURN m
  }
  RETURN m
}
RETURN m
```

Cypher runtime

When a query is submitted to a Composite database, different parts of the query may run using different runtimes. Clauses or expressions in scopes where no graph has been specified run using the *slotted* runtime. Parts of the query directed to different constituent graphs are run using the default runtime for that graph, or respect the submitted Cypher query options if specified.

Built-in graph functions

Graph functions are located in the namespace `graph`. The following table describes these functions:

Table 380. Built-in graph functions

Function	Explanation
`graph.names()`	Provides a list of names of all constituent graphs on the current Composite database.
`graph.byName(graphName)`	Used with the `USE` clause to select a constituent graph by name dynamically. This function is supported only with `USE` clauses.
`graph.propertiesByName(graphName)`	Returns a map containing the properties associated with the given graph.

For more information, see *Graph functions* in the Cypher Manual.

Sharding data with the `copy` command

Enterprise Edition | Not available on Aura

The `copy` command can be used to filter out data when creating database copies. In the following example, a sample database is separated into 3 shards.

Example 43. Use the `copy` command to filter out data

The sample database contains the following data:

```
(p1 :Person :S2 {id:123, name: "Ava"})
(p2 :Person :S2 {id:124, name: "Bob"})
(p3 :Person :S3 {id:125, name: "Cat", age: 54})
(p4 :Person :S3 {id:126, name: "Dan"})
(t1 :Team :S1 :SAll {id:1, name: "Foo", mascot: "Pink Panther"})
(t2 :Team :S1 :SAll {id:2, name: "Bar", mascot: "Cookie Monster"})
(d1 :Division :SAll {name: "Marketing"})
(p1)-[:MEMBER]->(t1)
```

```
(p2)-[:MEMBER]->(t2)
(p3)-[:MEMBER]->(t1)
(p4)-[:MEMBER]->(t2)
```

The data has been prepared using queries to add the labels `:S1`, `:S2`, `:S3`, and `:SAll`, which denotes the target shard. *Shard 1* contains the team data. *Shard 2* and *Shard 3* contain person data.

1. Create *Shard 1* with:

   ```
   bin/neo4j-admin database copy neo4j shard1 \
      --copy-only-nodes-with-labels=S1,SAll \
      --skip-labels=S1,S2,S3,SAll
   ```

   The `--copy-only-node-with-labels` property is used to filter out everything that does not have the label `:S1` or `:SAll`.

   The `--skip-labels` property is used to exclude the temporary labels you created for the sharding process.

   The resulting shard contains the following:

   ```
   (t1 :Team {id:1, name: "Foo", mascot: "Pink Panther"})
   (t2 :Team {id:2, name: "Bar", mascot: "Cookie Monster"})
   (d1 :Division {name: "Marketing"})
   ```

2. Create *Shard 2*:

   ```
   bin/neo4j-admin database copy neo4j shard2 \
      --copy-only-nodes-with-labels=S2,SAll \
      --skip-labels=S1,S2,S3,SAll \
      --copy-only-node-properties=Team.id
   ```

   In *Shard 2*, you want to keep the `:Team` nodes as proxy nodes, to be able to link together information from the separate shards. The nodes will be included since they have the label `:SAll`, but you specify `--copy-only-node-properties` so as to not duplicate the team information from *Shard 1*.

   ```
   (p1 :Person {id:123, name: "Ava"})
   (p2 :Person {id:124, name: "Bob"})
   (t1 :Team {id:1})
   (t2 :Team {id:2})
   (d1 :Division {name: "Marketing"})
   (p1)-[:MEMBER]->(t1)
   (p2)-[:MEMBER]->(t2)
   ```

   Observe that `--copy-only-node-properties` did not filter out `Person.name` since the `:Person` label was not mentioned in the filter.

3. Create *Shard 3*, but with the filter `--skip-node-properties`, instead of `--copy-only-node-properties`.

   ```
   bin/neo4j-admin database copy neo4j shard3 \
      --copy-only-nodes-with-labels=S3,SAll \
      --skip-labels=S1,S2,S3,SAll \
   ```

```
--skip-node-properties=Team.name,Team.mascot
```

The result is:

```
(p3 :Person {id:125, name: "Cat", age: 54})
(p4 :Person {id:126, name: "Dan"})
(t1 :Team {id:1})
(t2 :Team {id:2})
(d1 :Division {name: "Marketing"})
(p3)-[:MEMBER]->(t1)
(p4)-[:MEMBER]->(t2)
```

As demonstrated, you can achieve the same result with both `--skip-node-properties` and `--copy-only-node-properties`. In this example, it is easier to use `--copy-only-node-properties` because only one property should be kept. The relationship property filters work in the same way.

Query routing decisions

Enterprise Edition

Query routing is the process of deciding which Cypher executor (database) should be used and at which physical location the query should be executed. Every query that arrives at a Neo4j server, over the bolt protocol from a driver, undergoes the process described here.

Routing decision tree

Before the query is executed, these are the decisions taken during query routing stage:

Step 1: Determine the name of the target database

Pick the first of these that has a value:

1. Cypher `USE` clause
   - Note that administration commands implicitly have `USE system`.
2. Driver session database
3. Home or default database

Step 2: Reuse open transaction

- If there is an already open transaction to the target database, local or remote, then proceed to step 6.
- If not, then proceed to step 3.

Step 3: Determine the type of the target database (execution context type)

- If the target database is a database in this DBMS, then the context type is *Internal*.
- If the target database is a Composite database, then the context type is *Composite*.

Note:

This also allows the query to target multiple databases.

- If the target database is a Remote Alias, then the context type is *External*.

Step 4: Determine the location of execution

- If context type is *Internal*
  - and the URI scheme is `bolt://` (routing disabled), then location is *Local*.
  - and the transaction mode is `READ`
    - and the database is hosted on this server, then location is *Local*.
    - and the database is hosted on another sever
      - and Server-side routing is enabled, then location is *Remote* (using the routing advertised address of that server).
      - if not, then fail.
  - if the transaction mode is `WRITE`
    - and the local server is the leader for the database, then location is *Local*.
    - and another server is the leader for the database
      - and Server-side routing is enabled, then location is *Remote* (using the routing advertised address of that server).
      - and Server-side routing is *not* enabled, then fail.
- If context type is *Composite*, then location is *Local* (for this part of the query).
- If context type is *External*, then location is *Remote* (using the URI and database given in the configuration).

Step 5: Open a transaction

- If location is *Local*, then open a transaction to the database on this server.
- If location is *Remote*, then open a driver transaction to the database using the URI determined in step 4.

Step 6: Execute query

- Execute the query in the open transaction.

Illustrated routing decision tree

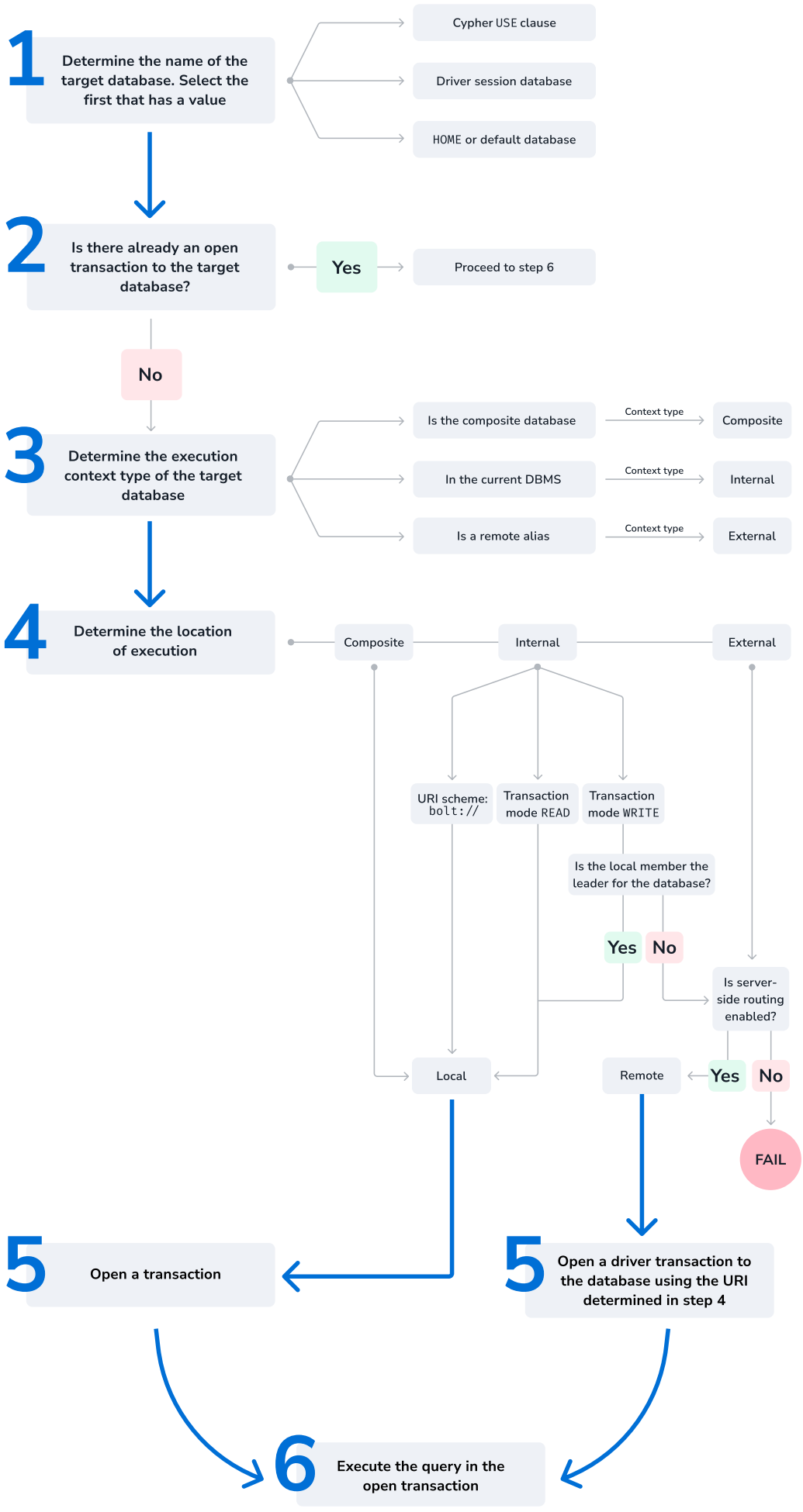


Figure 7. Illustrated routing decision tree

Database internals and transactional behavior

To maintain data integrity and ensure reliable transactional behavior, Neo4j DBMS supports transactions with full ACID properties, and it uses a write-ahead transaction log to ensure durability.

- **Atomicity** — If a part of a transaction fails, the database state is left unchanged.
- **Consistency** — Every transaction leaves the database in a consistent state.
- **Isolation** — During a transaction, modified data cannot be accessed by other operations.
- **Durability** — The DBMS can always recover the results of a committed transaction.

Neo4j DBMS supports the following transactional behavior:

- All database operations that access the graph, indexes, or schema must be performed in a transaction.
- The default isolation level is *read-committed* isolation level.
- Write locks are acquired automatically at the node and relationship levels. However, you can also manually acquire write locks if you want to achieve a higher level of isolation — *serializable* isolation level.
- Data retrieved by traversals is not protected from modification by other transactions.
- Non-repeatable reads may occur (i.e., only write locks are acquired and held until the end of the transaction).
- Deadlock detection is built into the core transaction management.

The following sections describe the transactional behavior in detail and how to control it:

- Transaction management
- Concurrent data access
- Transaction logging
- Checkpointing and log pruning
- Store formats

Note:

For information on Neo4j 4.4, see Java Reference 4.4 → Transaction management.

Transaction management

Transactions

Database operations that access the graph, indexes, or schema are performed in a transaction to ensure the ACID properties. Transactions are single-threaded, confined, and independent. Multiple transactions can be started in a single thread and they are independent of each other.

The interaction cycle of working with transactions follows the steps:

1. Begin a transaction.
2. Perform database operations.
3. Commit or roll back the transaction.

It is crucial to finish each transaction because the locks or memory acquired by a transaction are only released upon completion. All non-committed transactions are rolled back as part of resource cleanup at the end of the statement. No resource cleanup is required for a transaction that is explicitly committed or rolled back, and the transaction closure is an empty operation.

Note:

All modifications performed in a transaction are kept in memory. This means that very large updates must be split into several transactions to avoid running out of memory.

Configure transactional behavior

The transaction settings help you manage the transactions in your database, for example, the transaction timeout, the maximum number of concurrently running transactions, how much time to allow Neo4j to wait for running transactions to complete before allowing initiated database shutdown to continue, and so on. For all available settings, see Transaction settings.

Configure the maximum number of concurrently running transactions

By default, Neo4j can run a maximum of 1000 concurrent transactions. To change this value, use the `db.transaction.concurrent.maximum` setting. If set to `0`, the limit is disabled.

Configure transaction timeout

It is recommended to configure Neo4j to terminate transactions whose execution time has exceeded the configured timeout.

- Set `db.transaction.timeout` to some positive time interval value (e.g., `10s`) denoting the default transaction timeout. Setting `db.transaction.timeout` to `0` — which is the default value — disables the feature.
- You can also set this dynamically on each primary server using the procedure `dbms.setConfigValue('db.transaction.timeout','10s')`.

Example 44. Configure transaction timeout

Set the timeout to ten seconds.

```
db.transaction.timeout=10s
```

Configuring transaction timeout does not affect transactions executed with custom timeouts (e.g., via the

Java API or Neo4j Drivers), as the custom timeout overrides the value set for `db.transaction.timeout`. Note that the timeout value can only be overridden to a value smaller than that configured by `db.transaction.timeout`.

Introduced in 5.3

Starting from Neo4j 5.3, you can set the transaction timeout to any value, even larger than configured by `db.transaction.timeout`.

Manage transactions

Transactions can be managed using the Cypher commands `SHOW TRANSACTIONS` and `TERMINATE TRANSACTIONS`. The `TERMINATE TRANSACTIONS` command can be combined with multiple `SHOW TRANSACTIONS` and `TERMINATE TRANSACTIONS` commands in the same query.

For more information, see Cypher manual → Transaction commands.

Concurrent data access

Isolation levels

Neo4j supports the following isolation levels:

read-committed isolation level

Default A transaction that reads a node/relationship does not block another transaction from writing to that node/relationship before the first transaction finishes. This type of isolation is weaker than *serializable isolation level* but offers significant performance advantages while being sufficient for the overwhelming majority of cases.

serializable isolation level

Explicit locking of nodes and relationships. Using locks allows for simulating the effects of higher levels of isolation by obtaining and releasing locks explicitly. For example, if a write lock is taken on a common node or relationship, then all transactions are serialized on that lock — giving the effect of a *serializable isolation level*. For more information on how to manually acquire write locks, see Lost updates.

Anomalies

Depending on the isolation level, different anomalies may occur when multiple transactions concurrently read or write the same data.

All the anomalies listed here can only occur with the read-committed isolation level.

Lost updates

In Cypher, it is possible to acquire write locks to simulate improved isolation in some cases. Consider the case where multiple concurrent Cypher queries increment the value of a property. Due to the limitations of the *read-committed isolation level*, the increments might not result in a deterministic final value.

Cypher automatically acquires write locks in some cases, but not in others. When a Cypher query uses the `SET` clause to update a property, it may or may not acquire a write lock on the node or relationship being updated, depending on whether there is a direct dependency on the property being read.

Acquiring a write lock automatically

When a Cypher query has a direct dependency on the property being read, Cypher automatically acquires a write lock before reading the property. This is the case when the query uses the `SET` clause to update a property on a node or relationship, and the right-hand side of the `SET` clause has a dependency on the property being read. For example, in the following queries, the right-hand side of `SET` has a dependent property read in an expression or a value of a key-value pair in a literal map.

Example 45. Incrementing a property using an expression

```
MATCH (n:Example {id: 42})
SET n.prop = n.prop + 1
```

This query increments the property `n.prop` by 1. In this case, Cypher automatically acquires a write lock on the node `n` before reading the value of `n.prop`. This ensures that no other concurrent queries can modify the node `n` while this query is running, thus preventing lost updates.

Example 46. Incrementing a property using a map literal

```
MATCH (n)
SET n += {prop: n.prop + 1}
```

This query also increments the property `n.prop` by 1, but it does so using a map literal. In this case, Cypher also acquires a write lock on the node `n` before reading the value of `n.prop`.

No direct dependency to acquire a write lock

When a query does not have a direct dependency on the property being read, Cypher does not automatically acquire a write lock. This means if you run multiple concurrent queries that read and write the same property, it is possible to end up with lost updates by allowing other concurrent queries to modify the property value at the same time.

For example, if you run the following queries by one hundred concurrent clients, it is very likely not to increment the property `n.prop` to 100, unless a write lock is acquired before reading the property value. This is because all queries read the value of `n.prop` within their own transaction, and cannot see the incremented value from any other transaction that has not yet been committed. In the worst-case scenario, the final value would be as low as 1 if all threads perform the read before any has committed their transaction.

Example 47. Variable depending on results from reading the property in an earlier statement

```
MATCH (n)
WITH n.prop AS p
// ... operations depending on p, producing k
```

```
SET n.prop = k + 1
```

Example 48. Circular dependency between properties read and written in the same query

```
MATCH (n)
SET n += {propA: n.propB + 1, propB: n.propA + 1}
```

Workaround

To ensure deterministic behavior also in the more complex cases, it is necessary to explicitly acquire a write lock on the node in question. In Cypher there is no explicit support for this, but it is possible to work around this limitation by writing to a temporary property. For example, the following query acquires a write lock for the node by writing to a **dummy** property (`n._dummy_`) before reading the requested value (`n.prop`). When acquired, the write lock ensures that no other concurrent queries can modify the node until the transaction is committed or rolled back. The dummy property is used only to acquire the write lock, therefore, it can be removed immediately after the lock is acquired.

Example 49. Dummy property to acquire a write lock

```
MATCH (n:Example {id: 42})
SET n._dummy_ = true
REMOVE n._dummy_
WITH n.prop AS p
// ... operations depending on p, producing k
SET n.prop = k + 1
```

Non-repeatable reads

A non-repeatable read is when the same transaction reads the same data but gets inconsistent results. This can easily happen if reading the same data twice in a query and the data gets modified in-between by another concurrent query.

For example, the following query shows that reading the same property twice can give inconsistent results. If there are other queries running concurrently, it is not guaranteed that `p1` and `p2` have the same value.

Example 50. Non-repeatable read

```
MATCH (n:Example {id: 42})
WITH n.prop AS p1
// another concurrent query changes the value of n.prop here.
WITH *, n.prop AS p2
RETURN p1, p2
```

The easiest way to work around this is to only read each property once, and keep it as long as needed in the query.

Missing and double reads

When scanning an index, entities may be observed multiple times or skipped entirely, even if they are present in the index. This is true even for indexes that back property uniqueness constraints.

During the scan, if another concurrent query changes an entity's property to a position ahead of the scan, the entity might appear again in the index. Similarly, the entity may not appear at all if the property is changed to a previously scanned position.

This anomaly can only occur with operators that scan an index, or parts of an index, for example `DirectedRelationshipIndexSeekByRange`.

In the following query, each node `n` that has the property `prop` is expected to appear exactly once. However, concurrent updates that modify the `prop` property during index scanning may cause a node to appear multiple times or not at all in the result set.

Example 51. Missing and double read

```
MATCH (n:Example) WHERE n.prop IS NOT NULL
RETURN n
```

Locks

When a write transaction occurs, Neo4j takes locks to preserve data consistency while updating.

Locks are used in Neo4j to ensure data consistency and isolation levels. They not only protect logical entities (such as nodes and relationships) but also the integrity of internal data structures.

Locks are taken automatically by the queries that users run. They ensure that a node/relationship is locked to one particular transaction until that transaction is completed. In other words, a lock on a node or a relationship by one transaction pauses other transactions to concurrently modify the same node or relationship. As such, locks prevent concurrent modifications of shared resources between transactions.

Default locking behavior

The locks are added to the transaction and released when the transaction finishes. If the transaction is rolled back, the locks are released immediately.

The following is the default locking behavior for different operations:

- When adding, changing, or removing a property on a node or relationship, a write lock is taken on the specific node or relationship.
- When creating or deleting a node a write lock is taken for the specific node.
- When creating or deleting a relationship a write lock is taken on the specific relationship and both its nodes.

To view all active locks held by the transaction executing a query with the `queryId`, use the `CALL dbms.listActiveLocks(queryId)` procedure. You need to be an administrator to be able to run this procedure.

Table 381. Procedure output

Name	Type	Description
`mode`	`String`	Lock mode corresponding to the transaction.
`resourceType`	`String`	Resource type of the locked resource.
`resourceId`	`Integer`	Resource ID of the locked resource.

Example 52. Viewing active locks for a query

The following example shows the active locks held by the transaction executing a given query.

1. To get the IDs of the currently executing queries, yield the `currentQueryId` from the `SHOW TRANSACTIONS` command:

```
SHOW TRANSACTIONS YIELD currentQueryId, currentQuery
```

2. Run `CALL dbms.listActiveLocks` passing the `currentQueryId` of interest (`query-614` in this example):

```
CALL dbms.listActiveLocks( "query-614" )
```

"mode"	"resourceType"	"resourceId"
"SHARED"	"SCHEMA"	0

1 row

Lock contention

Lock contention may arise if an application needs to perform concurrent updates on the same nodes/relationships. In such a scenario, to be completed, transactions must wait for locks held by other transactions to be released. If two or more transactions attempt to modify the same data concurrently, it will increase the likelihood of a deadlock. In larger graphs, it is less likely that two transactions modify the same data concurrently, and so the likelihood of a deadlock is reduced. That said, even in large graphs, a deadlock can occur if two or more transactions are attempting to modify the same data concurrently.

Types of acquired locks

The following table shows the type of lock acquired depending on the graph modification:

Table 382. Obtained locks for graph modifications

Modification	Acquired lock
Creating a node	No lock

Modification	Acquired lock
Updating a node label	`NODE` lock
Updating a node property	`NODE` lock
Deleting a node	`NODE` lock
Creating a relationship*	If the node is sparse: `NODE` lock. If a node is dense: `NODE DELETE` prevention lock.
Updating a relationship property	`RELATIONSHIP` lock
Deleting a relationship*	If the node is sparse: `NODE` lock. If a node is dense: `NODE DELETE` prevention lock. `RELATIONSHIP` lock for both sparse and dense nodes.

**Applies for both source nodes and target nodes.*

Additional locks are often taken to maintain indexes and other internal structures depending on how other data in the graph is affected by a transaction. For these additional locks, no assumptions or guarantees can be made concerning which lock will or will not be taken.

Locks for dense nodes

When creating or deleting relationships, Neo4j does not exclusively lock dense nodes during a transaction. Rather, internally shared locks prevent the deletion of nodes and shared degree locks are acquired for synchronizing with concurrent label changes for those nodes to ensure correct count updates.

`standard`, `aligned`, *and* `high_limit` *store formats*

A node is considered dense if it, at any point, has had 50 or more relationships. Even if it later has fewer than 50 relationships, it is still considered dense.
A node is considered sparse if it has never had more than 50 relationships.
You can configure the relationship count threshold for when a node is considered dense by setting `db.relationship_grouping_threshold` configuration parameter.

`block` *format*

A node is considered dense for **a particular relationship type** when it surpasses a certain internal size threshold, which typically happens at approximately 50 relationships of that type. However, it also depends on the number and size of properties connected to these relationships. Therefore, a node may be dense for one relationship type (e.g., A) and sparse for another (e.g., relationship type B).

At commit time, relationships are inserted into the backing data structures in a manner that allows concurrent modification. For example, multiple transactions can create, update, or delete relationships connected to the same dense nodes concurrently. This process may in rare cases acquire additional exclusive locks in a sorted manner if necessary, to ensure data consistency.

In other words, relationship modifications acquire coarse-grained shared node locks when doing the

operation in the transaction, and then acquire precise exclusive locks during the commit.

The locking is very similar for sparse and dense nodes. The biggest contention for sparse nodes is the update of the degree (i.e. number of relationships) for the node. Dense nodes store this data in a concurrent data structure, and so can avoid exclusive node locks in almost all cases for relationship modifications.

Configure lock acquisition timeout

An executing transaction may get stuck while waiting for some lock to be released by another transaction. To kill that transaction and remove the lock, set `db.lock.acquisition.timeout` to some positive time interval value (e.g., `10s`) denoting the maximum time interval within which any particular lock should be acquired, before failing the transaction. Setting `db.lock.acquisition.timeout` to `0` — which is the default value — disables the lock acquisition timeout.

This feature cannot be set dynamically.

Example 53. Set the timeout to ten seconds

```
db.lock.acquisition.timeout=10s
```

Deadlocks

Since locks are used, deadlocks can happen. A deadlock occurs when two transactions are blocked by each other because they are attempting to concurrently modify a node or a relationship that is locked by the other transaction. In such a scenario, neither of the transactions will be able to proceed. When Neo4j detects a deadlock, the transaction is terminated with the transient error message code `Neo.TransientError.Transaction.DeadlockDetected`. From 5.25 onwards, the error message also contains the GQLSTATUS code `50N05` and the status description `error: general processing exception - deadlock detected. Deadlock detected while trying to acquire locks. See log for more details.`

All locks acquired by the transaction are still held but will be released when the transaction finishes. Once the locks are released, other transactions that were waiting for locks held by the transaction causing the deadlock can proceed. You can then retry the work performed by the transaction causing the deadlock if needed.

Experiencing frequent deadlocks is an indication of concurrent write requests happening in such a way that it is not possible to execute them while at the same time living up to the intended isolation and consistency. The solution is to make sure concurrent updates happen reasonably. For example, given two specific nodes (A and B), adding or deleting relationships to both these nodes in random order for each transaction results in deadlocks when two or more transactions do that concurrently. One option is to make sure that updates always happen in the same order (first A then B). Another option is to make sure that each thread/transaction does not have any conflicting writes to a node or relationship as some other concurrent transaction. This can, for example, be achieved by letting a single thread do all updates of a specific type.

Important:

Deadlocks caused by the use of other synchronization than the locks managed by Neo4j can still happen. Other code that requires synchronization should be synchronized in such a way that it never performs any Neo4j operation in the synchronized block.

Deadlock detection

For example, running the following two queries in Cypher-shell at the same time will result in a deadlock because they are attempting to modify the same node properties concurrently:

Example 54. Transaction A

```
:begin
MATCH (n:Test) SET n.prop = 1
WITH collect(n) as nodes
CALL apoc.util.sleep(5000)
MATCH (m:Test2) SET m.prop = 1;
```

Example 55. Transaction B

```
:begin
MATCH (n:Test2) SET n.prop = 1
WITH collect(n) as nodes
CALL apoc.util.sleep(5000)
MATCH (m:Test) SET m.prop = 1;
```

The following error message is thrown:

```
The transaction will be rolled back and terminated. Error: ForsetiClient[transactionId=6698,
clientId=1] can't acquire ExclusiveLock{owner=ForsetiClient[transactionId=6697, clientId=3]} on
NODE(27), because holders of that lock are waiting for ForsetiClient[transactionId=6698, clientId=1].
 Wait list:ExclusiveLock[
Client[6697] waits for [ForsetiClient[transactionId=6698, clientId=1]]]
```

Note:

The Cypher clause `MERGE` takes locks out of order to ensure the uniqueness of the data, and this may prevent Neo4j's internal sorting operations from ordering transactions in a way that avoids deadlocks. When possible, you are, therefore, encouraged to use the Cypher clause `CREATE` instead, which does not take locks out of order.

Deadlock handling in code

When dealing with deadlocks in code, there are several issues you may want to address:

- Only do a limited amount of retries, and fail if a threshold is reached.
- Pause between each attempt to allow the other transaction to finish before trying again.
- A retry loop can be useful not only for deadlocks but for other types of transient errors as well.

For an example of how deadlocks can be handled in procedures, server extensions, or when using Neo4j embedded, see Transaction management in the Neo4j Java Reference.

Avoiding deadlocks

Most likely, a deadlock will be resolved by retrying the transaction. This will, however, negatively impact the total transactional throughput of the database, so it is useful to know about strategies to avoid deadlocks.

Neo4j assists transactions by internally sorting operations. See below for more information about internal locks). However, this internal sorting only applies to the locks taken when creating or deleting relationships. Users are, therefore, encouraged to sort their operations in cases where Neo4j does not internally assist, such as when locks are taken for property updates. This is done by ensuring that updates occur in the same order. For example, if the three locks `A`, `B`, and `C` are always taken in the same order (e.g. `A→B→C`), then a transaction will never hold lock `B` while waiting for lock `A` to be released, and so a deadlock will not occur.

Another option is to avoid lock contention by not modifying the same entities concurrently.

To avoid deadlocks, internal locks should be taken in the following order:

Warning:

The internal lock types may change without any notification between different Neo4j versions. The lock types are only listed here to give an idea of the internal locking mechanism.

Lock type	Locked entity	Description
`LABEL` or `RELATIONSHIP_TYPE`	Token id	Schema locks, which lock indexes and constraints on the particular label or relationship type.
`SCHEMA_NAME`	Schema name	Lock a schema name to avoid duplicates. Collisions are possible because the hash is stringed. This only affects concurrency and not correctness.
`NODE_RELATIONSHIP_GROUP_DELETE`	Node id	Lock taken on a node during the transaction creation phase to prevent deletion of that node and/or relationship group. This is different from the `NODE` lock in order to allow concurrent label and property changes together with relationship modifications.
`NODE`	Node id	Lock on a node, used to prevent concurrent updates to the node records (i.e. add/remove label, set property, add/remove relationship). Note that updating relationships will only require a lock on the node if the head of the relationship chain/relationship group chain must be updated since that is the only data part of the node record.

Lock type	Locked entity	Description
`DEGREES`	Node id	Used to lock nodes to avoid concurrent label changes when a relationship is added or deleted. Such an update would otherwise lead to an inconsistent count store.
`RELATIONSHIP_DELETE`	Relationship id	Lock a relationship for exclusive access during deletion.
`RELATIONSHIP_GROUP`	Node id	Lock the full relationship group chain for a given dense node. This will not lock the node, in contrast to the lock `NODE_RELATIONSHIP_GROUP_DELETE`.
`RELATIONSHIP`	Relationship	Lock on a relationship, or more specifically a relationship record, to prevent concurrent updates.

Delete semantics

When deleting a node or a relationship, all properties for that entity will be automatically removed but the relationships of a node will not be removed. Neo4j enforces a constraint (upon commit) that all relationships must have a valid start node and end node. In effect, this means that trying to delete a node that still has relationships attached to it will throw an exception upon commit. It is, however, possible to choose in which order to delete the node and the attached relationships as long as no relationships exist when the transaction is committed.

The delete semantics can be summarized as follows:

- All properties of a node or relationship will be removed when it is deleted.
- A deleted node cannot have any attached relationships when the transaction commits.
- It is possible to acquire a reference to a deleted relationship or node that has not yet been committed.
- Any write operation on a node or relationship after it has been deleted (but not yet committed) will throw an exception.
- Trying to acquire a new or work with an old reference to a deleted node or relationship after commit, will throw an exception.

Transaction logging

Neo4j keeps track of all write operations to each database to ensure data consistency and enable recovery.

Transaction log files

A transaction log file contains a sequence of records with all changes made to a particular database as part of each transaction, including data, indexes, and constraints.

The transaction log serves multiple purposes, including providing differential backups and supporting cluster operations. At a minimum, the most recent non-empty transaction log is retained for any given configuration. It is important to note that transaction logs are unrelated to log monitoring.

The transaction logging configuration is set per database and can be configured using the following configuration settings:

Configure transaction log location

By default, transaction logs for a database are located at *<NEO4J_HOME>/data/transactions/<database-name>*.

The root directory where those folders are located is configured by `server.directories.transaction.logs.root`. The value is a path. If relative, it is resolved from `server.directories.data`. For maximum performance, it is recommended to configure transaction logs to be stored on a dedicated device.

Configure transaction log preallocation

You can specify if Neo4j should try to preallocate logical log files in advance using the parameter `db.tx_log.preallocate`. By default, it is `true`. Log preallocation optimizes the filesystem by ensuring there is room to accommodate newly generated files and avoid file-level fragmentation. This configuration setting is dynamic and can be changed at runtime.

Configure transaction log rotation size

You can specify how much space a single transaction log file can roughly occupy using `db.tx_log.rotation.size`. By default, it is set to `256 MiB`, which means that after a transaction log file reaches this size, it is rotated and a new one is created. The minimum accepted value is `128K` (128 KiB). This configuration setting is dynamic and can be changed at runtime.

This setting influences how much space can be reclaimed by all checkpoint strategies under the following:

To reclaim a given file, the newest checkpoint for the transaction log must exist in another file. So if you have a huge transaction log, then it is likely that your latest checkpoint is in the same file, making it impossible to reclaim said file. For information about checkpointing, see Control transaction log pruning.

Configure transaction log retention policy

Warning:

Manually deleting transaction log files is not supported.

You can control the number of transaction logs that Neo4j keeps to back up the database using the parameter `db.tx_log.rotation.retention_policy`. This configuration setting is dynamic and can be changed at runtime. For more information about how to do it, see Update dynamic settings.

Up to Neo4j 5.12, the default value is set to `2 days`, which means Neo4j keeps logical logs that contain any transaction committed within 2 days and prunes the ones that only contain transactions older than 2 days.

Introduced in 5.9

From Neo4j 5.9 onwards, you can optionally add a period-based restriction to the size of logs to keep. For example, `2 days 1G` prunes logical logs that only contain transactions older than 2 days or are larger than 1G.

Introduced in 5.13

Starting from Neo4j 5.13, the default value is changed to `2 days 2G`, which means Neo4j keeps logical logs that contain any transaction committed within 2 days from the current time and within the allocated log space (2G).

Other possible ways to configure the log retention policy are:

- `db.tx_log.rotation.retention_policy=true|keep_all` — keep transaction logs indefinitely.

  > **Note:**
  >
  > This option is not recommended due to the effectively unbounded storage usage. Old transaction logs cannot be safely archived or removed by external jobs since safe log pruning requires knowledge about the most recent successful checkpoint.

- `db.tx_log.rotation.retention_policy=false|keep_none` — keep only the most recent non-empty log.

  Log pruning is called only after checkpoint completion to ensure at least one checkpoint and points to a valid place in the transaction log data. In reality, this means that all transaction logs created between checkpoints are kept for some time, and only after a checkpoint, the pruning strategy removes them. For more details on how to speed up checkpointing, see Control transaction log pruning. To force a checkpoint, run the procedure `CALL db.checkpoint()`.

  > **Note:**
  >
  > This option is not recommended in production Enterprise Edition environments, as differential backups rely on the presence of the transaction logs since the last backup.

- `<number><optional unit> <type> <optional space restriction>` where valid units are `K`, `M`, and `G`, and valid types are `files`, `size`, `txs`, `entries`, `hours`, and `days`. Valid optional space restriction is a logical log space restriction like `1G`. For example, `2 days 1G` limits the logical log space on the disk to 1G at most 2 days per database.

  Table 383. Types that can be used to control log retention

Type	Description	Example
`files`	The number of the most recent transaction log files to keep after pruning.	`db.tx_log.rotation.retention_policy=10 files`
`size`	The max disk size of the transaction log files to keep after pruning. For example, `500M size` leaves at least 500M worth of files behind.	`db.tx_log.rotation.retention_policy=300M size`
`txs or entries`	The number of transactions (in the files) to keep after pruning, regardless of file count or size. `txs` and `entries` are synonymous. If set, the policy keeps the 500k latest transactions from each database and prunes any older transactions.	`db.tx_log.rotation.retention_policy=500k txs`

Type	Description	Example
`hours`	Keep logs that contain any transaction committed within the specified number of hours from the current time. The value of `10 hours` ensures that at least 10 hours' worth of transactions is present in the logs.	`db.tx_log.rotation.retention_policy=10 hours`
`days`	Keep logs that contain any transaction committed within the specified number of days from the current time.	`db.tx_log.rotation.retention_policy=30 days`
`days and size`	Keep logs that contain any transaction committed within the specified number of days from the current time and within the allocated log space.	`db.tx_log.rotation.retention_policy=2 days 1G`

Checkpointing and log pruning

Checkpointing is the process of flushing all pending updates from volatile memory to non-volatile data storage. This action is crucial to limit the number of transactions that need to be replayed during the recovery process, particularly to minimize the time required for recovery after an improper shutdown of the database or a crash.

Independent of the presence of checkpoints, database operations remain secure, as any transactions that have not been confirmed to have their modifications persisted to storage will be replayed upon the next database startup. However, this assurance is contingent upon the availability of the collection of changes comprising these transactions, which is maintained in the transaction logs.

Maintaining a long list of unapplied transactions (due to infrequent checkpoints) leads to the accumulation of transaction logs, as they are essential for recovery purposes. Checkpointing involves the inclusion of a special *Checkpointing* entry in the transaction log, marking the last transaction at which checkpointing occurred. This entry serves the purpose of identifying transaction logs that are no longer necessary, as all the transactions they contain have been securely stored in the storage files.

The process of eliminating transaction logs that are no longer required for recovery is known as *pruning*. Pruning is reliant on checkpointing. Checkpointing determines which logs can be pruned and determines the occurrence of pruning, as the absence of a checkpoint implies that the set of transaction log files available for pruning cannot have changed. Consequently, pruning is triggered whenever checkpointing occurs.

Note:

For information on checkpointing and log pruning in Neo4j 4.4, refer to Configuration settings → Checkpoint settings, Performance → Checkpoint IOPS limit, and Transaction log → Log pruning respectively.

Configure the checkpointing policy

The checkpointing policy, which is the driving event for log pruning is configured by `db.checkpoint`. Depending on your needs, the checkpoint can run on a periodic basis, which is the default, when a certain amount of data has been written to the transaction log, or continuously.

Table 384. Available checkpointing policies

Policy	Description
`PERIODIC`	Default This policy checks every 10 minutes whether there are changes pending flushing and if so, it performs a checkpoint and subsequently triggers a log prune. The periodic policy is specified by the `db.checkpoint.interval.tx` and `db.checkpoint.interval.time` settings and the checkpointing is triggered when either of them is reached. See Configure the checkpoint interval for more details.
`VOLUME`	This policy runs a checkpoint when the size of the transaction logs reaches the value specified by the `db.checkpoint.interval.volume` setting. By default, it is set to `250.00MiB`.
`CONTINUOUS`	Enterprise Edition This policy ignores `db.checkpoint.interval.tx` and `db.checkpoint.interval.time` settings and runs the checkpoint process all the time. The log pruning is triggered immediately after the checkpointing completes, just like in the periodic policy.
`VOLUMETRIC`	Enterprise Edition This policy checks every 10 seconds if there is enough volume of logs available for pruning and, if so, it triggers a checkpoint and subsequently, it prunes the logs. By default, the volume is set to 256MiB, but it can be configured using the setting `db.tx_log.rotation.retention_policy` and `db.tx_log.rotation.size`. For more information, see Configure transaction log rotation size.

Configure the checkpoint interval

Observing that you have more transaction log files than you expected is likely due to checkpoints either not happening frequently enough, or taking too long. This is a temporary condition and the gap between the expected and the observed number of log files will be closed on the next successful checkpoint. The interval between checkpoints can be configured using:

Table 385. Checkpoint interval configuration

Checkpoint configuration	Default value	Description
`db.checkpoint.interval.time`	`15m`	Configures the time interval between checkpoints.
`db.checkpoint.interval.tx`	`100000`	Configures the transaction interval between checkpoints.

Control transaction log pruning

Transaction log pruning refers to the safe and automatic removal of old, unnecessary transaction log files. Two things are necessary for a file to be removed:

- The file must have been rotated.
- At least one checkpoint must have happened in a more recent log file.

Transaction log pruning configuration primarily deals with specifying the number of transaction logs that should remain available. The primary reason for leaving more than the absolute minimum amount required for recovery comes from the requirements of clustered deployments and online backup. Since database updates are communicated between cluster members and backup clients through the transaction logs, keeping more than the minimum amount necessary allows for transferring just the incremental changes (in the form of transactions) instead of the whole store files, which can lead to substantial savings in time and

network bandwidth.

The number of transaction logs left after a pruning operation is controlled by the setting `db.tx_log.rotation.retention_policy`.

Introduced in 5.13

The default value of `db.tx_log.rotation.retention_policy` is changed from `2 days` to `2 days 2G`, which means that Neo4j keeps logical logs that contain any transaction committed within two days and within the designated log space of 2G. For more information, see Configure transaction log retention policy.

Having the least amount of transaction log data speeds up the checkpoint process. To configure the number of IOs per second the checkpoint process is allowed to use, use the configuration parameter `db.checkpoint.iops.limit`.

> **Note:**
>
> Disabling the IOPS limit can cause transaction processing to slow down a bit. For more information, see Checkpoint IOPS limit and Transaction log settings.

Checkpoint logging and metrics

The following details the expected messages to appear in the *logs\debug.log* upon a checkpoint event:

- Checkpoint based upon `db.checkpoint.interval.time`:

```
2023-05-28 12:55:05.174+0000 INFO [o.n.k.i.t.l.c.CheckPointerImpl] Checkpoint triggered by "Scheduled
checkpoint for time threshold" @ txId: 49 checkpoint started...
2023-05-28 12:55:05.253+0000 INFO [o.n.k.i.t.l.c.CheckPointerImpl] Checkpoint triggered by "Scheduled
checkpoint for time threshold" @ txId: 49 checkpoint completed in 79ms
```

- Checkpoint based upon `db.checkpoint.interval.tx`:

```
2023-05-28 13:08:51.603+0000 INFO [o.n.k.i.t.l.c.CheckPointerImpl] Checkpoint triggered by "Scheduled
checkpoint for tx count threshold" @ txId: 118 checkpoint started...
2023-05-28 13:08:51.669+0000 INFO [o.n.k.i.t.l.c.CheckPointerImpl] Checkpoint triggered by "Scheduled
checkpoint for tx count threshold" @ txId: 118 checkpoint completed in 66ms
```

- Checkpoint when `db.checkpoint=continuous`:

```
2023-05-28 13:17:21.927+0000 INFO [o.n.k.i.t.l.c.CheckPointerImpl] Checkpoint triggered by "Scheduled
checkpoint for continuous threshold" @ txId: 171 checkpoint started...
2023-05-28 13:17:21.941+0000 INFO [o.n.k.i.t.l.c.CheckPointerImpl] Checkpoint triggered by "Scheduled
checkpoint for continuous threshold" @ txId: 171 checkpoint completed in 13ms
```

- Checkpoint as a result of database shutdown:

```
2023-05-28 12:35:56.272+0000 INFO [o.n.k.i.t.l.c.CheckPointerImpl] Checkpoint triggered by "Database
shutdown" @ txId: 47 checkpoint started...
2023-05-28 12:35:56.306+0000 INFO [o.n.k.i.t.l.c.CheckPointerImpl] Checkpoint triggered by "Database
shutdown" @ txId: 47 checkpoint completed in 34ms
```

- Checkpoint as a result of `CALL db.checkpoint()`:

```
2023-05-28 12:31:56.463+0000 INFO [o.n.k.i.t.l.c.CheckPointerImpl] Checkpoint triggered by "Call to
db.checkpoint() procedure" @ txId: 47 checkpoint started...
2023-05-28 12:31:56.490+0000 INFO [o.n.k.i.t.l.c.CheckPointerImpl] Checkpoint triggered by "Call to
db.checkpoint() procedure" @ txId: 47 checkpoint completed in 27ms
```

- Checkpoint as a result of a backup run:

```
2023-05-28 12:33:30.489+0000 INFO [o.n.k.i.t.l.c.CheckPointerImpl] Checkpoint triggered by "Full
backup" @ txId: 47 checkpoint started...
2023-05-28 12:33:30.509+0000 INFO [o.n.k.i.t.l.c.CheckPointerImpl] Checkpoint triggered by "Full
backup" @ txId: 47 checkpoint completed in 20ms
```

Checkpoint Metrics are also available and are detailed in the following files, in the *metrics/* directory:

```
neo4j.check_point.duration.csv
neo4j.check_point.total_time.csv
neo4j.check_point.events.csv
```

Store formats

Neo4j's storage engine supports several store formats that describe how data is written to disk. Starting from 5.23, `block` format is the recommended format for Enterprise Edition due to its superior performance and scalability. `block` format uses advanced data structures and inlining techniques to enhance data locality, which leads to better resource utilization.

`aligned` is the default format for Community Edition.

`standard` and `high_limit` formats are deprecated starting from Neo4j 5.23. It is not recommended to use these formats for new databases. For more information on the deprecation and timeline for the eventual removal of these formats, see Format deprecations.

Note:

For information on store formats in Neo4j 4.4, see Tools → Neo4j Admin → Display store information.

Available store formats

Here's an overview of the available formats and their features:

Block

Enterprise Edition | GA in Neo4j 5.16

- Default format from Neo4j 5.22 for new databases (unless `db.format` is specified).
- **Performance:** Fast queries; uses advanced data structures and inlining techniques for better data locality.
- **Memory efficiency:** Optimized collocation of data, which allows more related data to be fetched by

fewer read operations; enhancing resource utilization. Block format means a few pages need to be loaded to serve a query, i.e. fewer page faults and less IO.

- **Property access:** Properties are stored in blocks with their nodes and relationships drastically reducing the amount of pointer chasing required to access properties.
- **Entity limits:** Able to run graphs at large scales. Supports the highest limits at the time of writing. See Block format entity limits for details.
  Introduced in 5.25 Supports token names (including label, property key, and relationship type names) of any length up to the GQL identifier max length of 16,383 characters.
- **Future-proofing:** Designed to be extended and improved without requiring store migrations. New features such as data types, or performance enhancements are available without rewriting the store.
- **Supported migration paths to other store formats:** Although it is not recommended, you can change your store format from `block` to `aligned` or `high_limit`. Note that `high_limit` is deprecated. The `aligned` format has lower limits than `block`, so your graph must fit within those reduced limits.

Aligned

- Default format in Community Edition and for all new databases in Enterprise Edition prior to Neo4j 5.22.
- **Performance:** Performs better than `standard` but requires slightly more disk space.
- **Memory efficiency:** Based on `standard` but with improved memory efficiency.
- **Property access:** Stores graph data in linked list-like structures on disk.
- **Entity limits:** Supports graphs within some limits. See Aligned format entity limits for details.
- **Supported migration paths to other store formats:** You can change your store format from `aligned` to `block` or `high_limit`. Note that the `high_limit` format is deprecated, therefore, it is not recommended to migrate to it.

High_limit

Enterprise Edition Deprecated in 5.23

- **Performance:** Performs slightly worse than `standard` and requires more disk space, but allows more nodes and relationships.
- **Memory efficiency:** Based on `standard` but with improved memory efficiency.
- **Property access:** Stores graph data in linked list-like structures on disk.
- **Entity limits:** From the record formats, supports the highest limits at the time of writing. For more information, see High_limit format entity limits.
- For information on deprecation and eventual removal, see Format deprecations.
- **Supported migration paths to other store formats:** You can migrate to the `block` format only.

Standard

Deprecated in 5.23

- **Performance:** Basic, foundational format.
- **Property access:** Stores graph data in linked list-like structures on disk.

- **Entity limits:** Supports graphs within some limits. See Standard format entity limits for details.
- For information on deprecation and eventual removal, see Format deprecations.
- **Supported migration paths to other store formats:** You can change your store format from `standard` to `aligned`, `block`, or `high_limit`. However, it is not recommended to migrate to the `high_limit` format as it is deprecated.

Format deprecations

`standard` and `high_limit` formats are deprecated starting from Neo4j 5.23.

`block` format provides better performance and scalability than the deprecated formats. They will remain part of the product throughout the v5 and vNext server lifecycles. The last version of Neo4j to include these formats will be vNext.LTS which is planned for release in November 2026. LTS versions of Neo4j are supported for 3 years following their release. This means that support for `standard` and `high_limit` formats is planned to end in November 2029.

It is recommended that Enterprise Edition users migrate all databases to `block` format at their earliest convenience to ensure the best possible performance. For more information, see Changing the store format of existing databases.

FAQ

1. **How can I find out the format of my database?**
   You can use the `SHOW DATABASES YIELD name, store` Cypher command to get information about the store format of a database. See Show detailed information for a particular database for a detailed example.
2. **When will support for `standard` and `high_limit` formats end?**
   The last version of Neo4j to include these formats will be an LTS release planned for November 2026, which will be supported for three years until November 2029.
3. **How can I change the store format of my database?**
   For information on changing the store format of an existing database, see Changing the store format of existing databases.
4. **From which store formats can I migrate, and to which formats?**
   For information about supported migration paths, refer to Neo4j Admin → Migrate a database.

How to set the database store format

You can either set the store format when creating a new database or change the store format of an existing database.

Creating new databases

In versions previous to Neo4j 5.22, the default store format for all new databases is `aligned`. From Neo4j 5.22, `block` is the default format for all newly-created databases as long as they do not have the `db.format` setting specified.
If you want to change it, you can set a new value for the `db.format` configuration in the *neo4j.conf* file.
You can also create a new database on a specific store format by passing the new format as an argument

to the command creating the database, for example, `neo4j-admin database import full` or `neo4j-admin database copy` commands, or by using `storeFormat:` option in the Cypher command `CREATE DATABASE`.

The following examples show how to create a new database on the `block` store format. However, the same applies to other formats.

Specify the store format when importing a database using the `neo4j-admin database import full` command

```
bin/neo4j-admin database import full ... --format=block blockdb
```

Specify the store format when copying a database using the `neo4j-admin database copy` command

```
bin/neo4j-admin database copy --to-format="block" mydb blockdb
```

Specify the store format when creating a new database using the `CREATE DATABASE` Cypher statement

```
CREATE DATABASE blockdb OPTIONS {storeFormat: 'block'}
```

Changing the store format of existing databases

Starting from 5.23, `block` format is the preferred format for Enterprise Edition due to its superior performance and scalability. Therefore, migrating all databases to `block` format is recommended to ensure optimal performance.

> **Note:**
>
> Be aware that changing the store format changes the internal IDs assigned to nodes and relationships. This is because the ID represents the element's physical location in the store file.

Changing the store format is an IO-intensive offline operation, which re-writes all data in the new format. Therefore, it requires that:

- There is enough disk space for both old and new copies of the database. During the migration to `block` format, the database is inherently compacted. Therefore, the disk space required for the migration is approximately the same as the size of the database. You can use the database store size metrics to determine your available disk space and potentially reusable space.
- The graph fits within the new format's entity limits.

> **Note:**
>
> For large databases changing the store format can be a time-consuming operation and will also require any indexes to be re-populated. The time required depends on the size of the database, number of indices, speed of the storage devices, and the amount of available memory. For example, a 100GB database might take 10 minutes in optimal conditions, or over an hour in the worst case. Therefore, it is recommended to perform a dry run on a backup to estimate the required time for the migration.

In a standalone server

Changing the store format of an existing database in a standalone server requires the database to be offline. The following steps assume that you want to migrate the database called `mydb` to `block` format but the same steps apply to other formats.

1. Stop the database using the Cypher command `STOP DATABASE mydb`.
2. Change the store format of the stopped database using **one** of the following options:
   - Migrate an existing database using `neo4j-admin database migrate` command.

     > **Important:**
     >
     > You do not need to run `neo4j-admin database copy` with the `--compact-node-store` option prior to running `neo4j-admin database migrate`. The database is inherently compacted during the migration process.

     For example:

     ```
     bin/neo4j-admin database migrate --to-format="block" mydb
     ```

   - Pass the new store format as an argument when using the `neo4j-admin database copy` command to create a copy of an existing database. You can also set the `--copy-schema` option to automatically copy the schema definitions. For example:

     ```
     bin/neo4j-admin database copy --to-format="block" mydb blockdb --copy-schema
     ```

3. After the successful completion, start the database using the Cypher command `START DATABASE mydb`. Indexes are populated the first time the database is started, which might take some time if there are property uniqueness constraints.

In a cluster

Changing the store format of an existing database in a cluster requires that you restore a backup of the database that you want to migrate on one of the servers, and then, use that server as a designated seeder for the other cluster members to copy that database from.

The following steps assume that you want to migrate the database called `mydb` to `block` format but the same steps apply to other formats. The database is hosted on three servers in primary mode.

On one of the servers, `server01`

1. In Cypher Shell, put the database that you want to migrate in read-only mode using the Cypher command `ALTER DATABASE databasename SET ACCESS READ ONLY`. For example:

   ```
   @system> ALTER DATABASE mydb SET ACCESS READ ONLY;
   ```

2. In your command-line tool, back up that database using the `neo4j-admin database backup` command

with the `--include-metadata=all` option to include all users and roles associated with it. For example:

```
bin/neo4j-admin database backup mydb --to-path=/path/to/your-backup-folder --include-metadata=all
```

The command creates a backup archive that contains both the database and the metadata associated with it.

3. In Cypher Shell, drop the database to delete it and all users and roles associated with it:

```
@system> DROP DATABASE mydb;
```

4. In the command-line tool, restore the backup that you created using the `neo4j-admin database restore` command:

```
bin/neo4j-admin database restore --from-path=/path/to/your-backup-folder/mydb-2024-03-05T11-26-38.backup mydb
```

5. Migrate the restored database to `block` format:

> **Important:**
>
> You do not need to run `neo4j-admin database copy` with the `--compact-node-store` option prior to running `neo4j-admin database migrate`. The database is inherently compacted during the migration process.

```
bin/neo4j-admin database migrate --to-format="block" mydb
```

6. In Cypher Shell, run `SHOW SERVERS` to find the server ID of `server01`. Cross-reference the address to find the server ID. Use any database to connect.

```
SHOW SERVERS YIELD serverId, name, address, state, health, hosting
```

On one of the servers:

1. Use the `system` database and create the migrated database `mydb` using the server ID of `server01`. The topology of `mydb` is stored in the `system` database and when you create it, it is allocated according to the default topology (which can be shown with `CALL dbms.showTopologyGraphConfig`). For more information, see Designated seeder.

```
CREATE DATABASE mydb OPTIONS {existingData: 'use', existingDataSeedInstance: '<server01 id>'}
```

2. Verify that the database is created and available using the Cypher command `SHOW DATABASE mydb`.
3. After the successful completion, restore the roles and permissions. For more information, see Restore users and roles metadata.

Verify the store format

You can verify the store format of a database using the following Cypher:

```
SHOW DATABASES YIELD name, store
```

Result

```
+---------------------------------+
| name      | store               |
+---------------------------------+
| "blockdb" | "block-block-1.1"   |
| "neo4j"   | "record-aligned-1.1" |
| "system"  | "record-aligned-1.1" |
+---------------------------------+
```

Additionally, you can use the `neo4j-admin database info` command to get detailed information about the store format of a database. For details, see Display store information.

Store formats and entity limits

The following tables show the format and Neo4j version compatibility and the limits of the different store formats:

Block format

Table 386. Block format and Neo4j version compatibility

Name	Store format version	Introduced in	GA from	Default in
`BLOCK_V1`	`block-block-1.1`	`5.14.0`	`5.16.0`	`5.22.0`

Table 387. Block format entity limits

Name	Limit
Nodes	`2^48` (281 474 976 710 656)
Relationships	`∞` (no defined upper bound)
Properties	`∞` (no defined upper bound)
Labels	`2^31` (2 147 483 648)
Relationship types	`2^30` (1 073 741 824)
Property keys	`2^31` (2 147 483 648)

Aligned format

Table 388. Aligned format and Neo4j version compatibility

Name	Store format version	Introduced in	Default in	Unsupported from
`ALIGNED_V5_0`	`record-aligned-1.1`	`5.0.0`	CE, EE < Neo4j 5.22	

Name	Store format version	Introduced in	Default in	Unsupported from
`ALIGNED_V4_3`	`AF4.3.0`	`4.3.0`	`5.0.0`	
`ALIGNED_V4_1`	`AF4.1.a`	`4.1.0`	`5.0.0`	

Table 389. Aligned format entity limits

Name	Limit
Property keys	`2^24` (16 777 216)
Nodes	`2^35` (34 359 738 368)
Relationships	`2^35` (34 359 738 368)
Properties	`2^36` (68 719 476 736)
Labels	`2^31` (2 147 483 648)
Relationship types	`2^16` (65 536)
Relationship groups	`2^35` (34 359 738 368)

Standard format

For information on deprecation and eventual removal, see Format deprecations.

Table 390. Standard format and Neo4j version compatibility

Name	Store format version	Introduced in	Unsupported from
`STANDARD_V5_0`	`record-standard-1.1`	`5.0.0`	
`STANDARD_V4_3`	`SF4.3.0`	`4.3.0`	`5.0.0`
`STANDARD_V4_0`	`SF4.0.0`	`4.0.0`	`5.0.0`
`STANDARD_V3_4`	`v0.A.9`	`3.4.0`	`5.0.0`

Table 391. Standard format entity limits

Name	Limit
Property keys	`2^24` (16 777 216)
Nodes	`2^35` (34 359 738 368)
Relationships	`2^35` (34 359 738 368)
Properties	`2^36` (68 719 476 736)
Labels	`2^31` (2 147 483 648)
Relationship types	`2^16` (65 536)
Relationship groups	`2^35` (34 359 738 368)

High_limit format

Deprecated in 5.23

For information on deprecation and eventual removal, see Format deprecations.

Table 392. High_limit format and Neo4j version compatibility

Name	Store format version	Introduced in	Unsupported from
`HIGH_LIMIT_V5_0`	`record-high_limit-1.1`	`5.0.0`	
`HIGH_LIMIT_V4_3_0`	`HL4.3.0`	`4.3.0`	`5.0.0`
`HIGH_LIMIT_V4_0_0`	`HL4.0.0`	`4.0.0`	`5.0.0`
`HIGH_LIMIT_V3_4_0`	`vE.H.4`	`3.4.0`	`5.0.0`
`HIGH_LIMIT_V3_2_0`	`vE.H.3`	`3.2.0`	`5.0.0`
`HIGH_LIMIT_V3_1_0`	`vE.H.2`	`3.1.0`	`5.0.0`
`HIGH_LIMIT_V3_0_6`	`vE.H.0b`	`3.0.6`	`5.0.0`
`HIGH_LIMIT_V3_0_0`	`vE.H.0`	`3.0.0`	`5.0.0`

Table 393. High_limit format entity limits

Name	Limit
Property keys	`2^24` (16 777 216)
Nodes	`2^50` (1 Quadrillion)
Relationships	`2^50` (1 Quadrillion)
Properties	`2^50` (1 Quadrillion)
Labels	`2^31` (2 147 483 648)
Relationship types	`2^24` (16 777 216)
Relationship groups	`2^50` (1 Quadrillion)

Clustering

This chapter describes the following:

- Introduction: Neo4j clustering architecture — An overview of Neo4j clustering basics.
- **Setting up a cluster** — The basics of configuring and deploying a new cluster.
  - Deploy a basic cluster — How to set up a basic cluster.
  - Deploy an analytics cluster — How to deploy a special case Neo4j cluster for analytic queries.
  - Move from a standalone deployment to a cluster — This section describes how to move from a single Neo4j server to Neo4j cluster.
  - Reconciler — An internal component that observes the requested state of a server and makes changes to the server to match that state.
  - Cluster server discovery — How servers in a cluster discover each other and form a cluster.
  - Leadership, routing and load balancing — Election of leaders, routing and load balancing.
  - Intra-cluster encryption — How to secure the cluster communication.
- Managing servers in a cluster — How to manage the servers in a cluster.
- Managing databases in a cluster — How to manage the databases in a cluster.
- Unbind the `system` database Introduced in 5.0 — How to use the `neo4j-admin dbms unbind-system-db` command.
- **Monitoring** — Monitoring of a cluster.
  - Monitor servers — The tools available for monitoring the servers in a cluster.
  - Monitor databases — The tools available for monitoring the databases in a cluster.
  - Monitor cluster endpoints for status information — The endpoints and semantics of endpoints used to monitor the health of the cluster.
  - Monitor replication status Introduced in 5.24 — The procedure to monitor which members of a clustered database are up-to-date and can participate in a successful replication.
- **Resilient multi-region cluster deployment** — Recommendations and guidance on how to set up a resilient cluster which ensures your database stays available, fast, and recoverable even under failures.
  - Designing a resilient multi-data center cluster — Recommended patterns of cluster deployment across multiple cloud regions / data centers.
  - Multi-data center routing — Clusters on multi-data centers.
  - Disaster recovery — How to recover a cluster in the event of a disaster.
- Settings reference — A summary of the most important cluster settings.
- Server management command syntax — Reference of Cypher administrative commands to add and manage servers.
- Clustering glossary — A glossary of terms related to the Neo4j clustering.

Introduction: Neo4j clustering architecture

Enterprise Edition

Overview

Neo4j's clustering provides these main features:

1. **Scalability:** A Neo4j cluster is a set of servers running multiple databases. Servers and databases are decoupled: servers provide computation and storage power for databases to use. Each database has its own independent topology, organized into primaries and secondaries (for read scaling).
2. **Fault tolerance:** Primary database allocations provide a fault tolerant platform for transaction processing. A database remains available for writes as long as a simple majority of its primary allocations are functioning.
3. **Operability:** Database management is separated from server management. For details, see Managing databases in a cluster and Managing servers in a cluster.
4. **Causal consistency:** When invoked, a client application is guaranteed to read at least its own writes.

For information about cluster design patterns and anti-patterns, see Designing a resilient multi-data center cluster.

Operational view

From an operational point of view, it is useful to view the cluster as a homogenous pool of servers which run a number of databases.

Note that `primary` and `secondary` are roles for a copy of a database. Servers do not have roles. Instead, they can be constrained by a `modeConstraint` set to `PRIMARY`, `SECONDARY`, or `NONE`. That means they can host copies of standard databases that have either a primary role or a secondary role (see servers 1 or 3 in the *Figure 1*). If the `modeConstraint` is set to `NONE`, a server can host copies of databases that have either role. In other words, a server can host primaries for some databases and secondaries for other databases (see server 6 in the *Figure 1*).

Similarly, it is possible for a database to be hosted on only one server, even when that server is part of a cluster. In such cases, the database allocation is always primary. See single primary for details.

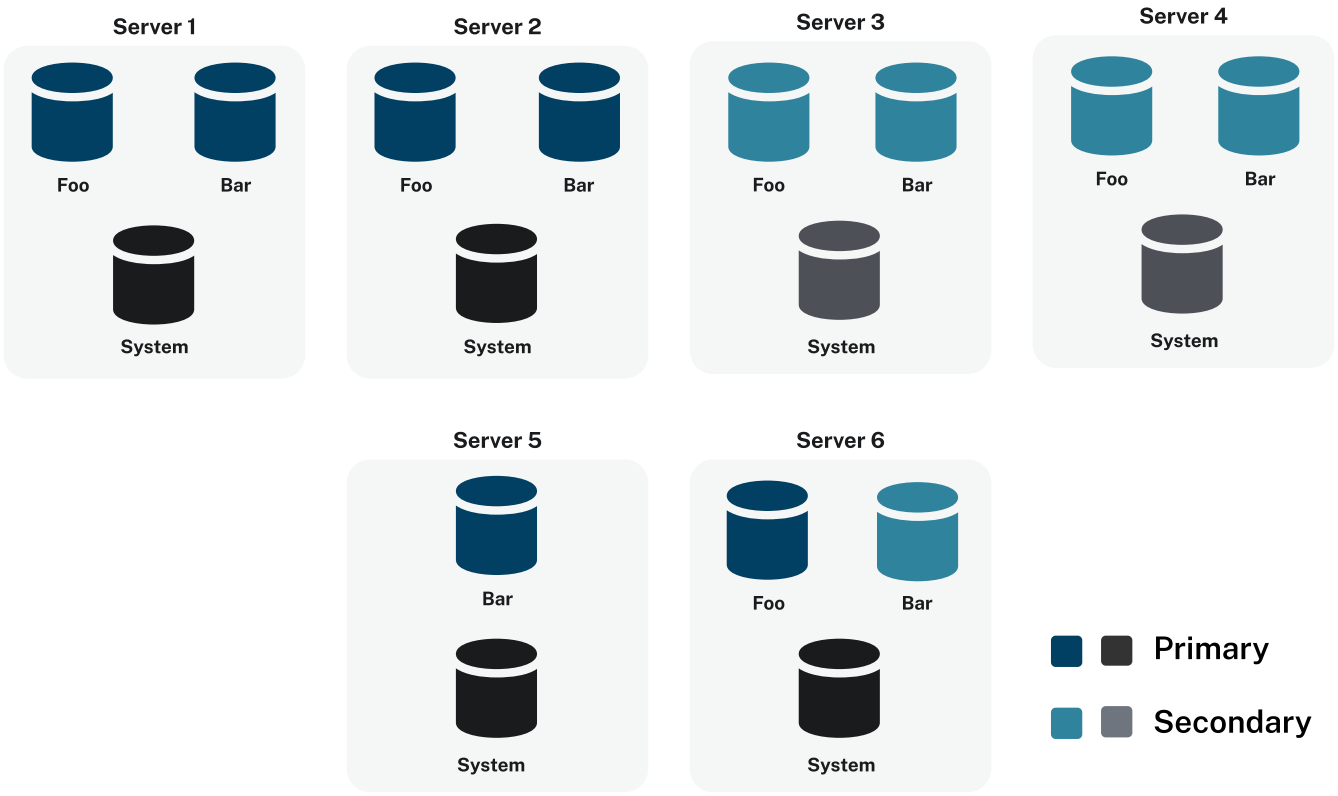


Figure 8. Cluster architecture

Database primaries

A *primary* is a copy of a database that is a participant in the processing of write operations and can be the writer for that database. A database can have one or more primary allocations within a cluster.

Only primaries are eligible to act as the writer for a database. At any given time, only one primary is automatically elected as a writer among the database's primaries. The writer may change over time.

The database writer synchronously pushes writes to other primaries and does not allow a commit to be completed until it receives confirmation that the data has been written to enough members.

For high availability, create a database with multiple primaries. If high availability is not required, a database can be created with a single primary to achieve minimum write latency.

If too many primaries fail, the database can no longer process writes and becomes read-only.

How many primaries you should have

Primaries provide:

- The ability to write to your database (with optional fault tolerance).
- The ability to read from your database.
- Fault tolerance for different failure scenarios.

The fault tolerance is calculated with the formula:
`M = 2F + 1`, where `M` is the number of primaries required to tolerate `F` faults.

Generally speaking, fault tolerance is the number of primary database copies you can lose without affecting a certain operation. For instance, with one primary copy, you have no fault tolerance, because if it goes offline, nothing is available. If you have two servers, each with a primary copy of the database, and one goes offline, the other will still have some copy of the data, so read availability would be preserved.

Types of fault tolerance, listed **from easiest to hardest to lose**, are as follows:

- **Write availability**:
  If write availability is lost, your database cannot accept any more writes.
- **Read availability**:
  If read availability is lost, your database cannot serve any more reads.
- **Durability**:
  If durability is lost, the data written to your database is lost, and you need to restore the database from a backup.

Operations such as shutting Neo4j process down to upgrade the binaries, or taking the server it runs on offline for maintenance, are included as *faults* in this context, since they make the database copy unavailable.

If you want upgrades with no downtime, you need fault tolerance.

Therefore, you need minimum three primaries (and three servers to host each copy) to be able to maintain write availability with the failure of one member. This is enough for most deployments.

If you want to retain write availability with the failure of two primary members, you need five primaries.

The maximum number of primaries you can have is **11**, but it is not recommended having that many primaries. Because the more primaries you have, the more servers you have to contact for each write operation, which can increase the latency of writes.

Database secondaries

A *secondary* is a database copy asynchronously replicated from primaries via transaction log shipping. Secondaries periodically check an upstream database member for new transactions, which are then transferred to them.

The main purpose of database secondaries is to scale out read workloads. Secondaries act like caches for graph data and can execute arbitrary read-only queries and procedures.

Multiple secondaries can be fed data from a relatively small number of primaries, providing a significant distribution of query workloads for better scalability. Databases can have a fairly large number of secondaries.

The loss of a secondary does not affect the database's availability; however, it reduces the query throughput. It also does not affect the database fault tolerance.

While secondaries serve as a copy of your database, providing some level of durability (what is committed cannot be lost), they do not guarantee it completely. Secondaries pull updates from a selected upstream member on their own schedule. Due to their asynchronous nature, secondaries may temporarily lag behind

the primary, meaning recently committed transactions may not be immediately visible on secondaries.

How many secondaries you should have

Secondaries typically provide read scaling, i.e. if you have more read queries happening than your primaries can handle, you can add secondaries to share the load; or even configure the query routing so that reads preferentially target secondaries to leave the primaries free to handle just the write workload.

So, there is no hard rule about the number of secondaries to have. Starting with zero and adding more until your read performance and cluster stability is acceptable is the usual approach.

The maximum number of secondaries you can have is **20**.

Primaries and secondaries for the `system` database

The `system` database, which records what databases are present in the DBMS, also can be in a primary or secondary mode. However, unlike standard databases, it is not configured using Cypher commands to define the topology. Instead, it is controlled through the `server.cluster.system_database_mode` setting.

Use the following guidelines when deciding how many primary and secondary `system` databases to have and which servers should host them:

- Stable, long-lived servers are good candidates to host a `system` primary, since they are expected to remain online and can be intentionally shut down when needed.
- Ephemeral or frequently changing servers are good candidates to host a `system` secondary, as they may be added or removed more often.
- A single `system` primary provides no fault tolerance for writes to the `system` database. Therefore, in a typical cluster deployment, it is best to start with three `system` primaries to ensure write availability.
- Although the write volume for the `system` database is low and it can tolerate higher write latency, allowing to have more than 11 `system` primaries, doing so is generally not recommended.

Examples of database topologies

For information about the cluster deployment across multiple data centers, refer to the Designing a resilient multi-data center cluster.

Single primary

If you have a single copy of the database, it is a primary. All writes and reads goes through this copy. If the copy becomes unavailable, no writes or reads are possible. If the disk for that copy is lost or corrupted, durability is lost and you must restore from the latest full backup.

Three primaries

In a cluster with three primaries, the members elect a leader to process write operations. Each write is replicated to at least one additional primary before being considered committed (durable). This ensures that if any single primary fails, that update remains available on another member. That includes if the

database copy is fully lost, including the disk being unrecoverable.

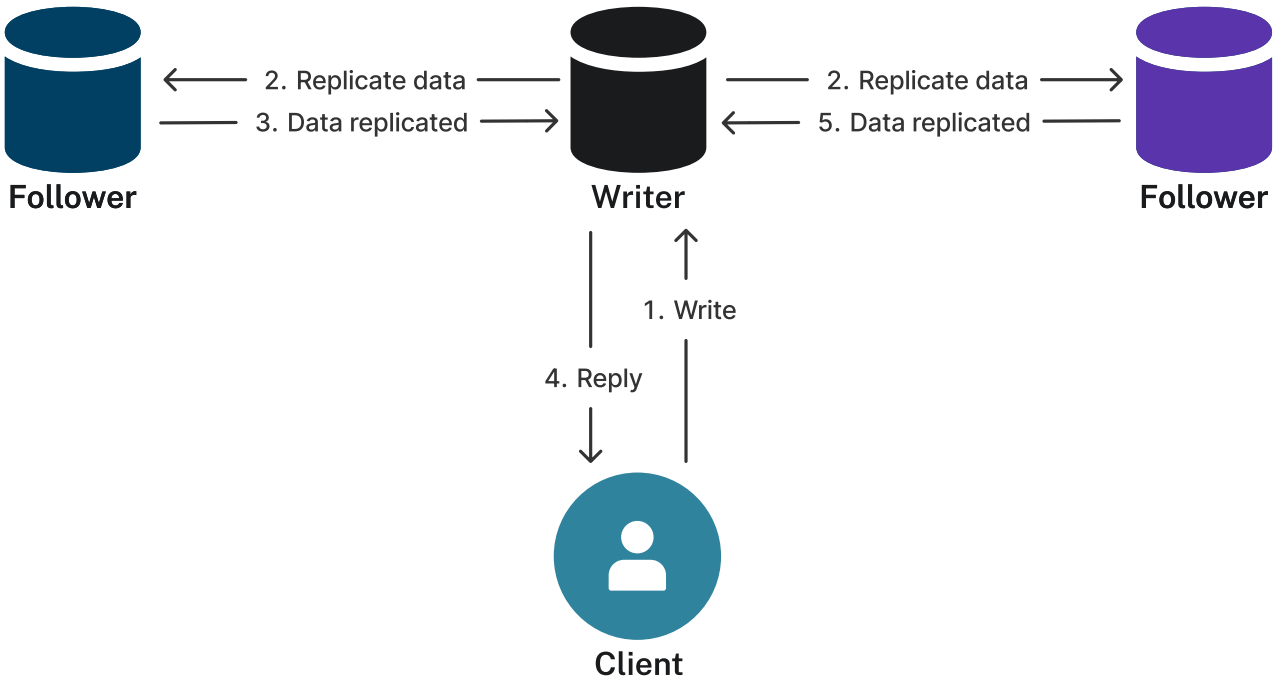


Figure 9. Communication between three primaries and a client

If one primary copy fails, the database is still write-available with the remaining two primaries, but it no longer has fault tolerance for its write availability.

Another failure would prevent any new writes from being processed until either one of the other members is brought back, or the database is recreated with new members. The database would still be read-available on the last member though.

The non-writer primaries also provide read capacity and fault tolerance. By default, read queries are routed away from the writer (see `dbms.routing.reads_on_writers_enabled`).

See Geo-distribution of user database primaries for the pattern of deploying a cluster with three primaries across three data centers.

Five primaries

If you want fault tolerance greater than one arbitrary database member, deploy five primaries. You will have tolerance to the failure of any two primaries. The remaining three primaries can still maintain quorum and ensure the database continues to operate.

For information about deploying five primaries across multiple data centers, see Resilient multi-region cluster deployment → Designing a resilient multi-data center cluster.

Single primary plus secondaries

As described above, a single primary provides no fault tolerance for both write availability or durability. If the single primary fails, no write operations can be processed, and if its disk is lost, the most recent updates may be lost.

However, adding one or more secondaries means that read availability can be maintained despite the loss

of the primary.

Keep in mind that the secondaries may not have the most up to date data, which is only guaranteed to be present on the primary.

The secondaries typically handle all of the read queries (see `dbms.routing.reads_on_writers_enabled`).

Three primaries plus secondaries

As described above, three primaries provide fault tolerance for both write availability and durability. Both secondaries and non-writer primaries can handle read queries. Although, you can configure only secondaries to handle reads (see `dbms.routing.reads_on_primaries_enabled`) if you need primaries to focus on the write workload.

The loss of any single database copy does not affect write availability, read availability, or durability.

If all the secondaries fail, then primaries that are not acting as the writer start handling read queries to maintain read availability.

See Read resilience with user database secondaries.

Causal consistency

While the operational mechanics of the cluster are interesting from an application point of view, it is also helpful to think about how applications use the database to get their work done. In many applications, it is typically desirable to both read from the graph and write to the graph. Depending on the nature of the workload, it is common to want reads from the graph to take into account previous writes to ensure causal consistency.

Note:

Causal consistency is one of numerous consistency models used in distributed computing. It ensures that causally related operations are seen by every instance in the system in the same order. Consequently, client applications are guaranteed to read their own writes, regardless of which instance they communicate with. This simplifies interaction with large clusters, allowing clients to treat them as a single (logical) server.

Causal consistency makes it possible to write to databases hosted on servers in primary mode and read those writes from databases hosted on servers in secondary mode (where graph operations are scaled out). For example, causal consistency guarantees that the write which created a user account is present when that same user subsequently attempts to log in.

On executing a transaction, the client can ask for a bookmark which it then presents as a parameter to subsequent transactions. Using that bookmark, the cluster can ensure that only servers which have processed the client's bookmarked transaction will run its next transaction. This provides a *causal chain* which ensures correct read-after-write semantics from the client's point of view.

Aside from the bookmark everything else is handled by the cluster. The database drivers work with the cluster topology manager to choose the most appropriate servers to route queries to. For instance, routing

reads to database secondaries and writes to database primaries.

Setting up a cluster

Deploy a basic cluster

The first step in setting up a cluster infrastructure is configuring a number of servers to form a cluster that you can host your databases on. The following configuration settings are important to consider when deploying a new cluster. See also Settings reference for more detailed descriptions and examples.

Starting with Neo4j 5.23, you get the option to choose between two discovery services: v1 or v2. They are designed to run independently and in parallel. However, the new version, v2, is recommended for new deployments, as v1 has been considered deprecated since Neo4j 5.23. For more details, see Cluster server discovery.

Table 394. Important settings for clusters

Setting name	Description
`server.default_advertised_address`	The address that other machines are told to connect to. In the typical case, this should be set to the fully qualified domain name or the IP address of this server.
`server.default_listen_address`	The address or network interface this machine uses to listen for incoming messages. Setting this value to `0.0.0.0` makes Neo4j bind to all available network interfaces.
`dbms.cluster.discovery.v2.endpoints` Introduced in 5.22	A comma-separated list of endpoints that a server should contact in order to discover other cluster members. Typically, all cluster members, including the current server, must be specified in this list. The setting configures the endpoints for **discovery service v2**.
`dbms.cluster.discovery.version` Introduced in 5.22	This setting allows you to select which discovery service should be started.
`dbms.cluster.discovery.endpoints` Deprecated in 5.23	The discovery network address for all the members of the cluster, including this server. The setting must be set to the same value on all cluster members. The behavior of this setting can be modified by configuring the setting `dbms.cluster.discovery.resolver_type`. This is described in detail in Cluster server discovery.
`dbms.cluster.minimum_initial_system_primaries_count`	Minimum number of machines initially required to form a clustered DBMS. The cluster is considered formed when at least this many members have discovered each other, bound together, and bootstrapped a highly available `system` database. The default value is `3`. As a result, at least this many of the cluster's initial machines must have `server.cluster.system_database_mode` set to `PRIMARY`.
`server.cluster.system_database_mode`	Users must manually specify the mode for the `system` database on each server. The default value is `PRIMARY`.

Setting name	Description
`initial.dbms.default_primaries_count`	The number of initial database hostings in primary mode. If not specified, it defaults to one hosting in primary mode.
`initial.dbms.default_secondaries_count`	The number of initial database hostings in secondary mode. If not specified, it defaults to zero hostings in secondary mode.
`initial.server.mode_constraint`	Specifies the server mode - whether it can host only primary databases, only secondary databases, or both types. Initialized at the first DBMS startup. When a new server is added to the cluster, the setting is used as the default input for the `ENABLE SERVER` command; can be overriden when the command is executed.
`initial.server.allowed_databases`	List of database names allowed on this server; all others are denied. When a new server is added to the cluster, the setting is used as the default input for the `ENABLE SERVER` command; can be overriden when the command is executed.
`initial.server.denied_databases`	List of database names not allowed on this server. Empty means nothing is denied. When a new server is added to the cluster, the setting is used as the default input for the `ENABLE SERVER` command; can be overriden when the command is executed.
`initial.server.tags`	A list of server tag names used by the database allocation and when configuring load balancing and replication policies. Initialized at the first DBMS startup and/or when a newly added server is enabled. The setting is used as the default input for the `ENABLE SERVER` command; can be overriden when the command is executed.

Points to consider:

- Any setting with the `initial` prefix is effective on the first startup of the DBMS and/or when a new server joins the cluster and has to be explicitly `ENABLED`.
- Changing the default number of primaries and secondaries dynamically can only be done with the `dbms.setDefaultAllocationNumbers` procedure. See `CREATE DATABASE` for more information.
- To view the current default settings, use the `dbms.showTopologyGraphConfig` procedure.

Caution:

Configuring any listen address to be something other than `localhost`, `127.0.0.1`, or another loopback address, will expose the Neo4j process to connections from outside of the server that it is running on.

Make sure you understand the security implications and strongly consider setting up encryption.

Configure a cluster with three servers

The following example shows how to set up a basic cluster with three members hosting the default database, `neo4j` (in addition to the `system` database), in primary mode, using the method of server addresses list.

Depending on the type of `dbms.cluster.discovery.resolver_type` currently in use, the discovery service can use a list of server addresses, DNS records, or Kubernetes services to discover other servers in the cluster.

In this case, you set `dbms.cluster.discovery.resolver_type=LIST`.

Example 56. Configure a cluster with three servers hosting databases in primary mode

In this example, three servers named `server01.example.com`, `server02.example.com` and `server03.example.com` are configured. Neo4j Enterprise Edition is installed on all three servers. They are configured by preparing *neo4j.conf* on each server.

Note that they are all identical, except for the configuration of `server.default_advertised_address`.

To use discovery service v2:

- You have to set the configuration of `dbms.cluster.discovery.version` to `V2_ONLY`.
- Instead of `dbms.cluster.discovery.endpoints`, use `dbms.cluster.discovery.v2.endpoints`.

Besides, in the example below, the `dbms.cluster.minimum_initial_system_primaries_count` setting is not configured, as it defaults to three, and a cluster of three servers is being deployed. However, in case of setting up a cluster with only **two** servers, the setting `dbms.cluster.minimum_initial_system_primaries_count` must be set to `2` on all servers.

1. Prepare a *neo4j.conf* file on each server.

   neo4j.conf on server01.example.com:

   ```
   server.default_listen_address=0.0.0.0
   server.default_advertised_address=server01.example.com
   dbms.cluster.discovery.v2.endpoints=server01.example.com:6000,server02.example.com:6000,serv
   er03.example.com:6000
   dbms.cluster.discovery.version=V2_ONLY
   initial.dbms.default_primaries_count=3
   ```

   neo4j.conf on server02.example.com:

   ```
   server.default_listen_address=0.0.0.0
   server.default_advertised_address=server02.example.com
   dbms.cluster.discovery.v2.endpoints=server01.example.com:6000,server02.example.com:6000,serv
   er03.example.com:6000
   dbms.cluster.discovery.version=V2_ONLY
   initial.dbms.default_primaries_count=3
   ```

   neo4j.conf on server03.example.com:

   ```
   server.default_listen_address=0.0.0.0
   ```

```
server.default_advertised_address=server03.example.com
dbms.cluster.discovery.v2.endpoints=server01.example.com:6000,server02.example.com:6000,serv
er03.example.com:6000
dbms.cluster.discovery.version=V2_ONLY
initial.dbms.default_primaries_count=3
```

The Neo4j servers are ready to be started. The startup order does not matter.

2. After the cluster has started, it is possible to connect to any of the instances and run `SHOW SERVERS` to check the status of the cluster. This shows information about each member of the cluster:

```
SHOW SERVERS;
```

```
+-----------------------------------------------------------------------------------------
----------------+
| name                                   | address          | state     | health      |
hosting             |
+-----------------------------------------------------------------------------------------
----------------+
| "d6fbe54b-0c6a-4959-9bcb-dcbbe80262a4" | "server01:7687" | "Enabled" | "Available" |
["system", "neo4j"] |
| "e56b49ea-243f-11ed-861d-0242ac120002" | "server02:7687" | "Enabled" | "Available" |
["system", "neo4j"] |
| "73e9a990-0a97-4a09-91e9-622bf0b239a4" | "server03:7687" | "Enabled" | "Available" |
["system", "neo4j"] |
+-----------------------------------------------------------------------------------------
----------------+
```

> **Note:**
>
> Note that initialized servers are `ENABLED` automatically at the first DBMS startup.
>
> Their values are taken from the initial settings.
>
> If you need to change the server's options, use the `ALTER SERVER` command.

3. For more extensive information about each server, use the `SHOW SERVERS YIELD *` command:

```
SHOW SERVERS YIELD *;
```

```
+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-------------------------------------------------------------+
| serverId                               | name                                   | address
| state     | health      | hosting             | requestedHosting    | tags |
allowedDatabases | deniedDatabases | modeConstraint | version     |
+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-------------------------------------------------------------+
| "d6fbe54b-0c6a-4959-9bcb-dcbbe80262a4" | "d6fbe54b-0c6a-4959-9bcb-dcbbe80262a4" |
"server01:7687" | "Enabled" | "Available" | ["system", "neo4j"] | ["system", "neo4j"] | []
| []               | []              | "NONE"         | "5.0.0"     |
| "e56b49ea-243f-11ed-861d-0242ac120002" | "e56b49ea-243f-11ed-861d-0242ac120002" |
"server02:7687" | "Enabled" | "Available" | ["system", "neo4j"] | ["system", "neo4j"] | []
| []               | []              | "NONE"         | "5.0.0"     |
| "73e9a990-0a97-4a09-91e9-622bf0b239a4" | "73e9a990-0a97-4a09-91e9-622bf0b239a4" |
"server03:7687" | "Enabled" | "Available" | ["system", "neo4j"] | ["system", "neo4j"] | []
| []               | []              | "NONE"         | "5.0.0"     |
```

```
+-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
-------------------------------------------------------------+
```

> **Tip:**
>
> **Startup time**
>
> The instance may appear unavailable while it is joining the cluster. If you want to follow along with the startup, you can see the messages in *neo4j.log*.

In this example, three servers named `server01.example.com`, `server02.example.com` and `server03.example.com` are configured. Neo4j Enterprise Edition is installed on all three servers. They are configured by preparing *neo4j.conf* on each server. Note that they are all identical, except for the configuration of `server.default_advertised_address`.

Besides, in the example below, the `dbms.cluster.minimum_initial_system_primaries_count` setting is not configured, as it defaults to three, and a cluster of three servers is being deployed. However, in case of setting up a cluster with only **two** servers, the setting `dbms.cluster.minimum_initial_system_primaries_count` must be set to `2` on all servers.

neo4j.conf on server01.example.com:

```
server.default_listen_address=0.0.0.0
server.default_advertised_address=server01.example.com
dbms.cluster.discovery.endpoints=server01.example.com:5000,server02.example.com:5000,server03.ex
ample.com:5000
initial.dbms.default_primaries_count=3
```

neo4j.conf on server02.example.com:

```
server.default_listen_address=0.0.0.0
server.default_advertised_address=server02.example.com
dbms.cluster.discovery.endpoints=server01.example.com:5000,server02.example.com:5000,server03.ex
ample.com:5000
initial.dbms.default_primaries_count=3
```

neo4j.conf on server03.example.com:

```
server.default_listen_address=0.0.0.0
server.default_advertised_address=server03.example.com
dbms.cluster.discovery.endpoints=server01.example.com:5000,server02.example.com:5000,server03.ex
ample.com:5000
initial.dbms.default_primaries_count=3
```

The Neo4j servers are ready to be started. The startup order does not matter.

After the cluster has started, it is possible to connect to any of the instances and run `SHOW SERVERS` to check the status of the cluster. This shows information about each member of the cluster:

```
SHOW SERVERS;
```

```
+-------------------------------------------------------------------------------------------
------------+
| name                                 | address          | state     | health      | hosting
|
+-------------------------------------------------------------------------------------------
------------+
| "d6fbe54b-0c6a-4959-9bcb-dcbbe80262a4" | "server01:7687" | "Enabled" | "Available" |
["system", "neo4j"] |
| "e56b49ea-243f-11ed-861d-0242ac120002" | "server02:7687" | "Enabled" | "Available" |
["system", "neo4j"] |
| "73e9a990-0a97-4a09-91e9-622bf0b239a4" | "server03:7687" | "Enabled" | "Available" |
["system", "neo4j"] |
+-------------------------------------------------------------------------------------------
------------+
```

> **Note:**
>
> Note that initialized servers are `ENABLED` automatically at the first DBMS startup.
>
> Their values are taken from the initial settings.
>
> If you need to change the server's options, use the `ALTER SERVER` command.

For more extensive information about each server, use the `SHOW SERVERS YIELD *` command:

```
SHOW SERVERS YIELD *;
```

```
+-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----------------------------------------------------+
| serverId                             | name                                   | address
| state     | health      | hosting             | requestedHosting    | tags | allowedDatabases
| deniedDatabases | modeConstraint | version     |
+-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----------------------------------------------------+
| "d6fbe54b-0c6a-4959-9bcb-dcbbe80262a4" | "d6fbe54b-0c6a-4959-9bcb-dcbbe80262a4" |
"server01:7687" | "Enabled" | "Available" | ["system", "neo4j"] | ["system", "neo4j"] | []   |
[]               | []              | "NONE"          | "5.0.0"     |
| "e56b49ea-243f-11ed-861d-0242ac120002" | "e56b49ea-243f-11ed-861d-0242ac120002" |
"server02:7687" | "Enabled" | "Available" | ["system", "neo4j"] | ["system", "neo4j"] | []   |
[]               | []              | "NONE"          | "5.0.0"     |
| "73e9a990-0a97-4a09-91e9-622bf0b239a4" | "73e9a990-0a97-4a09-91e9-622bf0b239a4" |
"server03:7687" | "Enabled" | "Available" | ["system", "neo4j"] | ["system", "neo4j"] | []   |
[]               | []              | "NONE"          | "5.0.0"     |
+-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----------------------------------------------------+
```

> **Tip:**
>
> **Startup time**
>
> The instance may appear unavailable while it is joining the cluster. If you want to follow along with the startup, you can see the messages in *neo4j.log*.

Create new databases in a cluster

As mentioned in the Introduction, a server in a cluster can either host a database in primary or secondary mode. For transactional workloads, a database topology with several primaries is preferred for fault tolerance and automatic failover.

The database topology might prioritize secondaries over primaries if the workload is more analytical. Such configuration is optimized for scalability but it is not fault-tolerant and does not provide automatic failover. Both scenarios are covered in the following examples.

Example 57. Create a new database with three primaries

In the `system` database on one of the servers from the previous example, execute the following Cypher command to create a new database:

```
CREATE DATABASE foo
TOPOLOGY 3 PRIMARIES
```

If `TOPOLOGY` is not specified, the database is created according to `initial.dbms.default_primaries_count` specified in `neo4j.conf`. Also, if `initial.dbms.default_secondaries_count` is specified to any other number than 0, the second line of the command would read `TOPOLOGY 3 PRIMARIES 0 SECONDARIES`. Thus the number specified with `TOPOLOGY` overrides both `initial.dbms.default_primaries_count` and `initial.dbms.default_secondaries_count` (if applicable) provided that the specified numbers do not exceed the number of available servers.

Example 58. Create a new database with one primary and two secondaries

In the `system` database on one of the servers from the previous example, execute the following Cypher command to create a new database:

```
CREATE DATABASE bar
TOPOLOGY 1 PRIMARY 2 SECONDARIES
```

Note that this operation is possible even without specifying `initial.dbms.default_secondaries_count` in the initial configuration. Anything specified in the `TOPOLOGY` part of the Cypher command overrides the `initial.dbms.default_secondaries_count` setting.

Note:

Analytic use cases

To learn more about setting up a cluster specifically for analytic use cases, see Deploy an analytics cluster.

Deploy an analytics cluster

In analytic use cases, the analytic queries can be intensive enough to warrant running them on separate

servers, away from the servers handling the transactional workload. This example shows how to set up a cluster to achieve that. Information on using Neo4j's graph data science library in a cluster can be found here: Neo4j Graph Data Science Library Manual → GDS with Neo4j cluster.

The analytics cluster can be set up in two ways, with fault tolerance or without fault tolerance. Bear in mind that the GDS library does **not** support fault tolerance and therefore GDS should only be deployed on servers configured for analytic queries, as described below.

With fault tolerance

Deploy the cluster

Starting with Neo4j 5.23, you can select between the discovery services: v1 or v2 — which allows servers to communicate with each other. Discovery service v2 is recommended for all new deployments, as v1 has been considered deprecated since Neo4j 5.23. For more details on the Neo4j discovery services, see Cluster server discovery.

Example 59. Configure a cluster with five servers, two only for read queries

In this example, three servers named `server01.example.com`, `server02.example.com` and `server03.example.com` are configured as the transactional part of the cluster. Two more servers names `server04.example.com` and `server05.example.com` are configured for the analytical queries. Neo4j Enterprise Edition is installed on all five servers. They are configured by preparing *neo4j.conf* on each server.

Key points:

- All servers include *all* members in their discovery list.
- The servers for analytics have their mode constraints configured to *SECONDARY* to prevent them from participating in write operations.
- In the example below, you set `dbms.cluster.discovery.resolver_type=LIST`.

To use the discovery service v2:

- You have to set the configuration of `dbms.cluster.discovery.version` to `V2_ONLY`.
- Instead of `dbms.cluster.discovery.endpoints`, use `dbms.cluster.discovery.v2.endpoints`.

1. Prepare a *neo4j.conf* file for each server.

   neo4j.conf on server01.example.com:

   ```
   server.default_listen_address=0.0.0.0
   server.default_advertised_address=server01.example.com
   dbms.cluster.discovery.v2.endpoints=server01.example.com:6000,server02.example.com:6000,
   server03.example.com:6000,server04.example.com:6000,server05.example.com:6000
   dbms.cluster.discovery.version=V2_ONLY
   ```

   neo4j.conf on server02.example.com:

   ```
   server.default_listen_address=0.0.0.0
   server.default_advertised_address=server02.example.com
   ```

```
dbms.cluster.discovery.v2.endpoints=server01.example.com:6000,server02.example.com:6000,
server03.example.com:6000,server04.example.com:6000,server05.example.com:6000
dbms.cluster.discovery.version=V2_ONLY
```

neo4j.conf on server03.example.com:

```
server.default_listen_address=0.0.0.0
server.default_advertised_address=server03.example.com
dbms.cluster.discovery.v2.endpoints=server01.example.com:6000,server02.example.com:6000,
server03.example.com:6000,server04.example.com:6000,server05.example.com:6000
dbms.cluster.discovery.version=V2_ONLY
```

neo4j.conf on server04.example.com - an analytics server:

```
server.default_listen_address=0.0.0.0
server.default_advertised_address=server04.example.com
dbms.cluster.discovery.v2.endpoints=server01.example.com:6000,server02.example.com:6000,
server03.example.com:6000,server04.example.com:6000,server05.example.com:6000
dbms.cluster.discovery.version=V2_ONLY
initial.server.mode_constraint=SECONDARY
server.cluster.system_database_mode=SECONDARY
```

neo4j.conf on server05.example.com - an analytics server:

```
server.default_listen_address=0.0.0.0
server.default_advertised_address=server05.example.com
dbms.cluster.discovery.v2.endpoints=server01.example.com:6000,server02.example.com:6000,
server03.example.com:6000,server04.example.com:6000,server05.example.com:6000
dbms.cluster.discovery.version=V2_ONLY
initial.server.mode_constraint=SECONDARY
server.cluster.system_database_mode=SECONDARY
```

2. The Neo4j servers are ready to be started. The startup order does not matter.
3. After the cluster has started, it is possible to connect to any of the instances and run `SHOW SERVERS` to check the status of the cluster. This shows information about each member of the cluster:

```
SHOW SERVERS;
```

```
+-----------------------------------------------------------------------------------------
------------------+
| name                                  | address         | state     | health      |
hosting             |
+-----------------------------------------------------------------------------------------
------------------+
| "f3bd1199-bc6f-4a38-b25c-5f7588df5182" | "server01:7687" | "Enabled" | "Available" |
["system", "neo4j"] |
| "b331e481-c2ba-4b4e-82f3-bb51fe171483" | "server02:7687" | "Enabled" | "Available" |
["system"]          |
| "bd80e8fd-a51b-406a-9ed4-42daf4792aa6" | "server03:7687" | "Enabled" | "Available" |
["system"]          |
| "df3758b1-337f-4b8a-a9de-8e745ca96549" | "server04:7687" | "Free"    | "Available" |
["system"]          |
| "9207bfd9-aa1b-40c2-b965-edcd3955a20e" | "server05:7687" | "Free"    | "Available" |
["system"]          |
+-----------------------------------------------------------------------------------------
------------------+
```

For more extensive information about each server, use the `SHOW SERVERS YIELD *` command:

```
SHOW SERVERS YIELD *;
```

```
+-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
| serverId                                | name                                      |
address          | state     | health      | hosting             | requestedHosting    |
tags | allowedDatabases | deniedDatabases | modeConstraint | version     |
+-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
| "f3bd1199-bc6f-4a38-b25c-5f7588df5182" | "f3bd1199-bc6f-4a38-b25c-5f7588df5182" |
"server01:7687" | "Enabled" | "Available" | ["system", "neo4j"] | ["system", "neo4j"] |
[]   | []              | []              | "NONE"         | "5.8.0"     |
| "b331e481-c2ba-4b4e-82f3-bb51fe171483" | "b331e481-c2ba-4b4e-82f3-bb51fe171483" |
"server02:7687" | "Enabled" | "Available" | ["system"]          | ["system"]          |
[]   | []              | []              | "NONE"         | "5.8.0"     |
| "bd80e8fd-a51b-406a-9ed4-42daf4792aa6" | "bd80e8fd-a51b-406a-9ed4-42daf4792aa6" |
"server03:7687" | "Enabled" | "Available" | ["system"]          | ["system"]          |
[]   | []              | []              | "NONE"         | "5.8.0"     |
| "df3758b1-337f-4b8a-a9de-8e745ca96549" | "df3758b1-337f-4b8a-a9de-8e745ca96549" |
"server04:7687" | "Free"    | "Available" | ["system"]          | []                  |
[]   | []              | []              | "SECONDARY"    | "5.8.0"     |
| "9207bfd9-aa1b-40c2-b965-edcd3955a20e" | "9207bfd9-aa1b-40c2-b965-edcd3955a20e" |
"server05:7687" | "Free"    | "Available" | ["system"]          | []                  |
[]   | []              | []              | "SECONDARY"    | "5.8.0"     |
+-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
```

4. To support analytic queries on the secondary servers, you have to explicitly enable them by running:

```
ENABLE SERVER "df3758b1-337f-4b8a-a9de-8e745ca96549";
ENABLE SERVER "9207bfd9-aa1b-40c2-b965-edcd3955a20e";
```

To check the result, run `SHOW SERVERS`. The output should show:

```
+-------------------------------------------------------------------------------------
-------------------+
| name                                    | address         | state     | health      |
hosting             |
+-------------------------------------------------------------------------------------
-------------------+
| "f3bd1199-bc6f-4a38-b25c-5f7588df5182" | "server01:7687" | "Enabled" | "Available" |
["system", "neo4j"] |
| "b331e481-c2ba-4b4e-82f3-bb51fe171483" | "server02:7687" | "Enabled" | "Available" |
["system"]          |
| "bd80e8fd-a51b-406a-9ed4-42daf4792aa6" | "server03:7687" | "Enabled" | "Available" |
["system"]          |
| "df3758b1-337f-4b8a-a9de-8e745ca96549" | "server04:7687" | "Enabled" | "Available" |
["system"]          |
| "9207bfd9-aa1b-40c2-b965-edcd3955a20e" | "server05:7687" | "Enabled" | "Available" |
["system"]          |
+-------------------------------------------------------------------------------------
-------------------+
```

In this example, three servers named `server01.example.com`, `server02.example.com` and `server03.example.com` are configured as the transactional part of the cluster. Two more servers names `server04.example.com` and `server05.example.com` are configured for the analytical queries. Neo4j Enterprise Edition is installed on all five servers. They are configured by preparing *neo4j.conf* on each server.

Key points:

- All servers include *all* members in their discovery list.
- The servers for analytics have mode constraints configured that restrict their hosting mode to secondary to prevent them from participating in normal write operations.
- In the example below, you set `dbms.cluster.discovery.resolver_type=LIST`.

1. Prepare a *neo4j.conf* file for each server.

neo4j.conf on server01.example.com:

```
server.default_listen_address=0.0.0.0
server.default_advertised_address=server01.example.com
dbms.cluster.discovery.endpoints=server01.example.com:5000,server02.example.com:5000,ser
ver03.example.com:5000,server04.example.com:5000,server05.example.com:5000
```

neo4j.conf on server02.example.com:

```
server.default_listen_address=0.0.0.0
server.default_advertised_address=server02.example.com
dbms.cluster.discovery.endpoints=server01.example.com:5000,server02.example.com:5000,ser
ver03.example.com:5000,server04.example.com:5000,server05.example.com:5000
```

neo4j.conf on server03.example.com:

```
server.default_listen_address=0.0.0.0
server.default_advertised_address=server03.example.com
dbms.cluster.discovery.endpoints=server01.example.com:5000,server02.example.com:5000,ser
ver03.example.com:5000,server04.example.com:5000,server05.example.com:5000
```

neo4j.conf on server04.example.com - an analytics server:

```
server.default_listen_address=0.0.0.0
server.default_advertised_address=server04.example.com
dbms.cluster.discovery.endpoints=server01.example.com:5000,server02.example.com:5000,ser
ver03.example.com:5000,server04.example.com:5000,server05.example.com:5000
initial.server.mode_constraint=SECONDARY
server.cluster.system_database_mode=SECONDARY
```

neo4j.conf on server05.example.com - an analytics server:

```
server.default_listen_address=0.0.0.0
server.default_advertised_address=server05.example.com
dbms.cluster.discovery.endpoints=server01.example.com:5000,server02.example.com:5000,ser
ver03.example.com:5000,server04.example.com:5000,server05.example.com:5000
initial.server.mode_constraint=SECONDARY
server.cluster.system_database_mode=SECONDARY
```

2. The Neo4j servers are ready to be started. The startup order does not matter.
3. After the cluster has started, it is possible to connect to any of the instances and run `SHOW SERVERS` to check the status of the cluster. This shows information about each member of the cluster:

```
SHOW SERVERS;
```

```
+--------------------------------------------------------------------------------------
```

```
-------------------+
| name                                  | address          | state     | health      |
hosting             |
+-----------------------------------------------------------------------------------------
-------------------+
| "f3bd1199-bc6f-4a38-b25c-5f7588df5182" | "server01:7687" | "Enabled" | "Available" |
["system", "neo4j"] |
| "b331e481-c2ba-4b4e-82f3-bb51fe171483" | "server02:7687" | "Enabled" | "Available" |
["system"]          |
| "bd80e8fd-a51b-406a-9ed4-42daf4792aa6" | "server03:7687" | "Enabled" | "Available" |
["system"]          |
| "df3758b1-337f-4b8a-a9de-8e745ca96549" | "server04:7687" | "Free"    | "Available" |
["system"]          |
| "9207bfd9-aa1b-40c2-b965-edcd3955a20e" | "server05:7687" | "Free"    | "Available" |
["system"]          |
+-----------------------------------------------------------------------------------------
-------------------+
```

For more extensive information about each server, use the `SHOW SERVERS YIELD *` command:

```
SHOW SERVERS YIELD *;
```

```
+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
| serverId                              | name                                    |
address          | state     | health      | hosting             | requestedHosting    |
tags | allowedDatabases | deniedDatabases | modeConstraint | version     |
+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
| "f3bd1199-bc6f-4a38-b25c-5f7588df5182" | "f3bd1199-bc6f-4a38-b25c-5f7588df5182" |
"server01:7687" | "Enabled" | "Available" | ["system", "neo4j"] | ["system", "neo4j"] |
[]   | []               | []              | "NONE"         | "5.8.0"     |
| "b331e481-c2ba-4b4e-82f3-bb51fe171483" | "b331e481-c2ba-4b4e-82f3-bb51fe171483" |
"server02:7687" | "Enabled" | "Available" | ["system"]          | ["system"]          |
[]   | []               | []              | "NONE"         | "5.8.0"     |
| "bd80e8fd-a51b-406a-9ed4-42daf4792aa6" | "bd80e8fd-a51b-406a-9ed4-42daf4792aa6" |
"server03:7687" | "Enabled" | "Available" | ["system"]          | ["system"]          |
[]   | []               | []              | "NONE"         | "5.8.0"     |
| "df3758b1-337f-4b8a-a9de-8e745ca96549" | "df3758b1-337f-4b8a-a9de-8e745ca96549" |
"server04:7687" | "Free"    | "Available" | ["system"]          | []                  |
[]   | []               | []              | "SECONDARY"    | "5.8.0"     |
| "9207bfd9-aa1b-40c2-b965-edcd3955a20e" | "9207bfd9-aa1b-40c2-b965-edcd3955a20e" |
"server05:7687" | "Free"    | "Available" | ["system"]          | []                  |
[]   | []               | []              | "SECONDARY"    | "5.8.0"     |
+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
```

4. To support analytic queries on the secondary servers, you have to explicitly enable them by running:

```
ENABLE SERVER "df3758b1-337f-4b8a-a9de-8e745ca96549";
ENABLE SERVER "9207bfd9-aa1b-40c2-b965-edcd3955a20e";
```

To check the result, run `SHOW SERVERS`. The output should show:

```
+-----------------------------------------------------------------------------------------
-------------------+
| name                                  | address          | state     | health      |
hosting             |
+-----------------------------------------------------------------------------------------
-------------------+
| "f3bd1199-bc6f-4a38-b25c-5f7588df5182" | "server01:7687" | "Enabled" | "Available" |
```

```
["system", "neo4j"] |
| "b331e481-c2ba-4b4e-82f3-bb51fe171483" | "server02:7687" | "Enabled" | "Available" |
["system"]          |
| "bd80e8fd-a51b-406a-9ed4-42daf4792aa6" | "server03:7687" | "Enabled" | "Available" |
["system"]          |
| "df3758b1-337f-4b8a-a9de-8e745ca96549" | "server04:7687" | "Enabled" | "Available" |
["system"]          |
| "9207bfd9-aa1b-40c2-b965-edcd3955a20e" | "server05:7687" | "Enabled" | "Available" |
["system"]          |
+-------------------------------------------------------------------------------------------
------------------+
```

Create new databases in the cluster

As mentioned in the Introduction, a server in a cluster can either host a database in primary or secondary mode. The primaries provide write operations and fault tolerance (if there are more than one of them). The secondaries provide somewhere for read queries to run, including the analytic queries for this example.

Example 60. Create a new database with three primaries and two secondaries

In the `system` database from the previous example, execute the following Cypher command to create a new database:

```
CREATE DATABASE bar
TOPOLOGY 3 PRIMARIES 2 SECONDARIES
```

Without fault tolerance

If fault tolerance is not a priority, it is possible to have a single server handling write operations for the DBMS. This means fewer servers are needed, but a failure can affect the entire DBMS.

Warning:

If the single writer server or process fails, all databases will be unavailable for writes. They may be less available for reads as well, since that server manages the DBMS.

The following example shows how to set up a non-fault tolerant analytics cluster with three members.

Deploy the cluster

Example 61. Configure a cluster with three servers

In this example, three servers named `server01.example.com`, `server02.example.com` and `server03.example.com` are configured. Neo4j Enterprise Edition is installed on all three servers. They are configured by preparing *neo4j.conf* on each server. Note that `server01.example.com` is different from the others, and is the only server where write operations take place. The other servers are able to execute read queries, and if using GDS, to write results back to the writing server.

Key points:

- The writer server only has itself in the list of discovery. This means it does not seek out the other members when it starts, they have to discover it. This is required in order to have a cluster with only a single primary for the `system` database.
- The servers for analytics have their mode constraints configured to `SECONDARY` to prevent them from participating in write operations.
- In the example below, you set `dbms.cluster.discovery.resolver_type=LIST`.

To use the discovery service v2:

- You have to set the configuration of `dbms.cluster.discovery.version` to `V2_ONLY`.
- Instead of `dbms.cluster.discovery.endpoints`, use `dbms.cluster.discovery.v2.endpoints`.

1. Prepare a *neo4j.conf* file for each server.

   neo4j.conf on server01.example.com - the writer:

   ```
   server.default_listen_address=0.0.0.0
   server.default_advertised_address=server01.example.com
   # Only has self in this list
   dbms.cluster.discovery.v2.endpoints=server01.example.com:6000
   dbms.cluster.discovery.version=V2_ONLY
   ```

   neo4j.conf on server02.example.com - an analytics server:

   ```
   server.default_listen_address=0.0.0.0
   server.default_advertised_address=server02.example.com
   dbms.cluster.discovery.v2.endpoints=server01.example.com:6000,server02.example.com:6000,
   server03.example.com:6000
   dbms.cluster.discovery.version=V2_ONLY
   initial.server.mode_constraint=SECONDARY
   server.cluster.system_database_mode=SECONDARY
   ```

   neo4j.conf on server03.example.com - an analytics server:

   ```
   server.default_listen_address=0.0.0.0
   server.default_advertised_address=server03.example.com
   dbms.cluster.discovery.v2.endpoints=server01.example.com:6000,server02.example.com:6000,
   server03.example.com:6000
   dbms.cluster.discovery.version=V2_ONLY
   initial.server.mode_constraint=SECONDARY
   server.cluster.system_database_mode=SECONDARY
   ```

2. The Neo4j servers are ready to be started. The startup order does not matter.
3. After the cluster has started, it is possible to connect to any of the instances and run `SHOW SERVERS` to check the status of the cluster. This shows information about each member of the cluster:

   ```
   SHOW SERVERS;
   ```

   ```
   +-----------------------------------------------------------------------------------------
   -------------------+
   | name                                  | address          | state     | health      |
   hosting             |
   +-----------------------------------------------------------------------------------------
   ```

```
-------------------+
| "d6fbe54b-0c6a-4959-9bcb-dcbbe80262a4" | "server01:7687" | "Enabled" | "Available" |
["system", "neo4j"] |
| "e56b49ea-243f-11ed-861d-0242ac120002" | "server02:7687" | "Free"    | "Available" |
["system"]          |
| "73e9a990-0a97-4a09-91e9-622bf0b239a4" | "server03:7687" | "Free"    | "Available" |
["system"]          |
+-------------------------------------------------------------------------------------------
-------------------+
```

For more extensive information about each server, use the `SHOW SERVERS YIELD *` command:

```
SHOW SERVERS YIELD *;
```

```
+-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
| serverId                                | name                                          |
address          | state     | health      | hosting             | requestedHosting    |
tags | allowedDatabases | deniedDatabases | modeConstraint | version     |
+-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
| "d6fbe54b-0c6a-4959-9bcb-dcbbe80262a4" | "d6fbe54b-0c6a-4959-9bcb-dcbbe80262a4" |
"server01:7687" | "Enabled" | "Available" | ["system", "neo4j"] | ["system", "neo4j"] |
[]   | []               | []              | "NONE"         | "5.8.0"     |
| "e56b49ea-243f-11ed-861d-0242ac120002" | "e56b49ea-243f-11ed-861d-0242ac120002" |
"server02:7687" | "Free"    | "Available" | ["system"]          | ["system"]          |
[]   | []               | []              | "SECONDARY"    | "5.8.0"     |
| "73e9a990-0a97-4a09-91e9-622bf0b239a4" | "73e9a990-0a97-4a09-91e9-622bf0b239a4" |
"server03:7687" | "Free"    | "Available" | ["system"]          | ["system"]          |
[]   | []               | []              | "SECONDARY"    | "5.8.0"     |
+-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
```

4. To support analytic queries on the secondary servers, you need to explicitly enable them by running:

```
ENABLE SERVER "e56b49ea-243f-11ed-861d-0242ac120002";
ENABLE SERVER "73e9a990-0a97-4a09-91e9-622bf0b239a4";
```

To check the result, run `SHOW SERVERS`. The output should show:

```
+-------------------------------------------------------------------------------------------
-------------------+
| name                                    | address         | state     | health      |
hosting             |
+-------------------------------------------------------------------------------------------
-------------------+
| "d6fbe54b-0c6a-4959-9bcb-dcbbe80262a4" | "server01:7687" | "Enabled" | "Available" |
["system", "neo4j"] |
| "e56b49ea-243f-11ed-861d-0242ac120002" | "server02:7687" | "Enabled" | "Available" |
["system"]          |
| "73e9a990-0a97-4a09-91e9-622bf0b239a4" | "server03:7687" | "Enabled" | "Available" |
["system"]          |
+-------------------------------------------------------------------------------------------
-------------------+
```

In this example, three servers named `server01.example.com`, `server02.example.com` and

`server03.example.com` are configured. Neo4j Enterprise Edition is installed on all three servers. They are configured by preparing *neo4j.conf* on each server. Note that `server01.example.com` is different from the others, and is the only server where write operations take place. The other servers are able to execute read queries, and if using GDS, to write results back to the writing server.

Key points:

- The writer server only has itself in the list of discovery. This means it does not seek out the other members when it starts, they have to discover it. This is required in order to have a cluster with only a single primary for the `system` database.
- The servers for analytics have mode constraints configured that restrict their hosting mode to *secondary* to prevent them from participating in normal write operations.
- In the example below, you set `dbms.cluster.discovery.resolver_type=LIST`.

1. Prepare a *neo4j.conf* file for each server.

   neo4j.conf on server01.example.com - the writer:

   ```
   server.default_listen_address=0.0.0.0
   server.default_advertised_address=server01.example.com
   # Only has self in this list
   dbms.cluster.discovery.endpoints=server01.example.com:5000
   ```

   neo4j.conf on server02.example.com - an analytics server:

   ```
   server.default_listen_address=0.0.0.0
   server.default_advertised_address=server02.example.com
   dbms.cluster.discovery.endpoints=server01.example.com:5000,server02.example.com:5000,ser
   ver03.example.com:5000
   initial.server.mode_constraint=SECONDARY
   server.cluster.system_database_mode=SECONDARY
   ```

   neo4j.conf on server03.example.com - an analytics server:

   ```
   server.default_listen_address=0.0.0.0
   server.default_advertised_address=server03.example.com
   dbms.cluster.discovery.endpoints=server01.example.com:5000,server02.example.com:5000,ser
   ver03.example.com:5000
   initial.server.mode_constraint=SECONDARY
   server.cluster.system_database_mode=SECONDARY
   ```

2. The Neo4j servers are ready to be started. The startup order does not matter.
3. After the cluster has started, it is possible to connect to any of the instances and run `SHOW SERVERS` to check the status of the cluster. This shows information about each member of the cluster:

   ```
   SHOW SERVERS;
   ```

   ```
   +--------------------------------------------------------------------------------------
   ------------------+
   | name                                 | address         | state     | health      |
   hosting            |
   +--------------------------------------------------------------------------------------
   ------------------+
   ```

```
| "d6fbe54b-0c6a-4959-9bcb-dcbbe80262a4" | "server01:7687" | "Enabled" | "Available" |
["system", "neo4j"] |
| "e56b49ea-243f-11ed-861d-0242ac120002" | "server02:7687" | "Free"    | "Available" |
["system"]          |
| "73e9a990-0a97-4a09-91e9-622bf0b239a4" | "server03:7687" | "Free"    | "Available" |
["system"]          |
+-------------------------------------------------------------------------------------
------------------+
```

For more extensive information about each server, use the `SHOW SERVERS YIELD *` command:

```
SHOW SERVERS YIELD *;
```

```
+-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
| serverId                                | name                                    |
address          | state     | health      | hosting             | requestedHosting    |
tags | allowedDatabases | deniedDatabases | modeConstraint | version     |
+-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
| "d6fbe54b-0c6a-4959-9bcb-dcbbe80262a4" | "d6fbe54b-0c6a-4959-9bcb-dcbbe80262a4" |
"server01:7687" | "Enabled" | "Available" | ["system", "neo4j"] | ["system", "neo4j"] |
[]   | []              | []              | "NONE"         | "5.8.0"     |
| "e56b49ea-243f-11ed-861d-0242ac120002" | "e56b49ea-243f-11ed-861d-0242ac120002" |
"server02:7687" | "Free"    | "Available" | ["system"]          | ["system"]          |
[]   | []              | []              | "SECONDARY"    | "5.8.0"     |
| "73e9a990-0a97-4a09-91e9-622bf0b239a4" | "73e9a990-0a97-4a09-91e9-622bf0b239a4" |
"server03:7687" | "Free"    | "Available" | ["system"]          | ["system"]          |
[]   | []              | []              | "SECONDARY"    | "5.8.0"     |
+-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
```

4. To support analytic queries on the secondary servers, you need to explicitly enable them by running:

```
ENABLE SERVER "e56b49ea-243f-11ed-861d-0242ac120002";
ENABLE SERVER "73e9a990-0a97-4a09-91e9-622bf0b239a4";
```

To check the result, run `SHOW SERVERS`. The output should show:

```
+-------------------------------------------------------------------------------------
------------------+
| name                                    | address         | state     | health      |
hosting             |
+-------------------------------------------------------------------------------------
------------------+
| "d6fbe54b-0c6a-4959-9bcb-dcbbe80262a4" | "server01:7687" | "Enabled" | "Available" |
["system", "neo4j"] |
| "e56b49ea-243f-11ed-861d-0242ac120002" | "server02:7687" | "Enabled" | "Available" |
["system"]          |
| "73e9a990-0a97-4a09-91e9-622bf0b239a4" | "server03:7687" | "Enabled" | "Available" |
["system"]          |
+-------------------------------------------------------------------------------------
------------------+
```

Create new databases in the cluster

As mentioned in the Introduction, a server in a cluster can either host a database in primary or secondary mode. For transactional workloads, a database topology with several primaries is preferred for fault tolerance and automatic failover. The database topology may prioritize secondaries over primaries if the workload is more analytical. Such configuration is optimized for scalability but it is **not** fault-tolerant and does not provide automatic failover.

Example 62. Create a new database with one primary and two secondaries

In the `system` database on the writer from the previous example, execute the following Cypher command to create a new database:

```
CREATE DATABASE bar
TOPOLOGY 1 PRIMARY 2 SECONDARIES
```

Tip:

Startup time

The instance may appear unavailable while it is joining the cluster. If you want to follow along with the startup, you can see the messages in *neo4j.log*.

Running analytic queries

If running large normal Cypher queries, it is possible to use server tags to identify the large servers, and a routing policy to direct the read queries towards those servers. See Multi-data center routing for more details.

If using GDS, follow the guidance in Neo4j Graph Data Science Library Manual → GDS with Neo4j cluster.

Move from a standalone deployment to a cluster

It is possible to move from a standalone deployment with a single `system` database to a cluster with multiple `system` primaries. In essence, this is done by dumping the `system` database from the standalone server and loading it into the other servers that are to form the cluster. The following example shows how to move from a standalone server with a single `system` primary to a cluster with three `system` primaries.

Note:

Another deployment with a single `system` database is an **analytics cluster** with a single `system` primary. If desired to move to a cluster with multiple `system` primaries, the following example is applicable with the addition that the secondaries are discarded (this is done in the first step when the `neo4j.conf` file is modified). See Deploy an analytics cluster for more information on analytics clusters.

Example 63. Move from a single `system` database to a cluster with three `system` primaries

In this example, a standalone server named `server01` is running and two additional servers, `server02` and `server03`, are to be added to form a cluster. The two additional servers are configured according to Configure a cluster with three servers. These two new servers should not be started up yet. Neo4j Enterprise Edition is installed on all three servers.

Start by stopping the standalone server. Once it is stopped, edit the *neo4j.conf* file to include the discovery endpoints of itself and the servers that will form the cluster.

Starting with Neo4j 5.23, it is recommended to use v2 of the discovery service. For more details on discovery services, see Cluster server discovery.

To use the discovery service v2:

- You have to set the configuration of `dbms.cluster.discovery.version` to `V2_ONLY`.
- Instead of `dbms.cluster.discovery.endpoints`, use `dbms.cluster.discovery.v2.endpoints`.

The example below uses `dbms.cluster.discovery.resolver_type=LIST`.

neo4j.conf on server01.example.com:

```
server.default_listen_address=0.0.0.0
server.default_advertised_address=server01.example.com
dbms.cluster.discovery.v2.endpoints=server01.example.com:6000,server02.example.com:6000,server03
.example.com:6000
dbms.cluster.discovery.version=V2_ONLY
initial.dbms.default_primaries_count=3
```

(The *neo4j.conf* file looks identical except for the `server.default_advertised_address` on all three servers. Please refer to Configure a cluster with three servers for more information.)

On `server01` (the standalone server) dump the `system` database using the `neo4j-admin database dump` command.

```
bin/neo4j-admin database dump system --to-path=/full/path/to/dumps/
```

See Back up an offline database for more information on the dump command.

Use the `neo4j-admin database load` command to load the `system` database dump from `server01` to `server02` and `server03`.

```
bin/neo4j-admin database load --from-path=/full-path/data/dumps system
```

See Restore a database dump for more information on the load command.

Once the `system` database has been loaded on `server02` and `server03`, start all servers. The newly added servers should be in the `Free` state (`server02` and `server03`) and this can be verified using `SHOW SERVERS`.

```
SHOW SERVERS;
```

```
+-----------------------------------------------------------------------------------------------
------------+
| name                                   | address         | state     | health      | hosting
|
+-----------------------------------------------------------------------------------------------
------------+
| "d6fbe54b-0c6a-4959-9bcb-dcbbe80262a4" | "server01:7687" | "Enabled" | "Available" |
["system", "neo4j"]  |
| "e56b49ea-243f-11ed-861d-0242ac120002" | "server02:7687" | "Free"    | "Available" |
["system"]           |
| "73e9a990-0a97-4a09-91e9-622bf0b239a4" | "server03:7687" | "Free"    | "Available" |
["system"]           |
+-----------------------------------------------------------------------------------------------
------------+
```

On `server01` (the previously standalone server) verify that all user databases are still running using `SHOW DATABASES`.

The last step is to enable the `Free` servers using `ENABLE SERVER` (see Managing servers in a cluster for more information on server states).

Once all servers are enabled, you can scale up user databases using `ALTER DATABASE`, if desired.

In this example, a standalone server named `server01` is running and two additional servers, `server02` and `server03`, are to be added to form a cluster. The two additional servers are configured according to Configure a cluster with three servers. These two new servers should not be started up yet. Neo4j Enterprise Edition is installed on all three servers.

Start by stopping the standalone server. Once it is stopped, edit the *neo4j.conf* file to include the discovery endpoints of itself and the servers that will form the cluster.

neo4j.conf on server01.example.com:

```
server.default_listen_address=0.0.0.0
server.default_advertised_address=server01.example.com
dbms.cluster.discovery.endpoints=server01.example.com:5000,server02.example.com:5000,server03.ex
ample.com:5000
initial.dbms.default_primaries_count=3
```

(The *neo4j.conf* file looks identical except for the `server.default_advertised_address` on all three servers. Please refer to Configure a cluster with three servers for more information.)

On `server01` (the standalone server) dump the `system` database using the `neo4j-admin database dump` command.

```
bin/neo4j-admin database dump system --to-path=/full/path/to/dumps/
```

See Back up an offline database for more information on the dump command.

Use the `neo4j-admin database load` command to load the `system` database dump from `server01` to `server02` and `server03`.

```
bin/neo4j-admin database load --from-path=/full-path/data/dumps system
```

See Restore a database dump for more information on the load command.

Once the `system` database has been loaded on `server02` and `server03`, start all servers. The newly added servers should be in the `Free` state (`server02` and `server03`) and this can be verified using `SHOW SERVERS`.

```
SHOW SERVERS;
```

```
+-----------------------------------------------------------------------------------------------------
------------+
| name                                   | address          | state     | health      | hosting
|
+-----------------------------------------------------------------------------------------------------
------------+
| "d6fbe54b-0c6a-4959-9bcb-dcbbe80262a4" | "server01:7687" | "Enabled" | "Available" |
["system", "neo4j"]  |
| "e56b49ea-243f-11ed-861d-0242ac120002" | "server02:7687" | "Free"    | "Available" |
["system"]           |
| "73e9a990-0a97-4a09-91e9-622bf0b239a4" | "server03:7687" | "Free"    | "Available" |
["system"]           |
+-----------------------------------------------------------------------------------------------------
------------+
```

On `server01` (the previously standalone server) verify that all user databases are still running using `SHOW DATABASES`.

The last step is to enable the `Free` servers using `ENABLE SERVER` (see Managing servers in a cluster for more information on server states).

Once all servers are enabled, you can scale up user databases using `ALTER DATABASE`, if desired.

Reconciler

In Neo4j, administrative operations such as database creation do not happen synchronously. Instead, the changes are first written to the `system` database, recording the requested state of the DBMS. Each server has a *reconciler*, an internal component which observes the requested state and makes changes to the local server to match that state. For example, the `system` database may record that the database `bar` is expected to be running on `server-1`. When the reconciler on `server-1` becomes aware of that, it starts `bar`. This means the loss of any one server cannot prevent all progress on an operation.

Executing an operation and getting back a successful response means that the request is safely committed to the `system` database, and will be processed by each member of the cluster at some point, assuming the server is healthy enough.

Servers can become aware the same operation at different times, since the `system` database is just another Raft group, where followers and replicas can lag behind the leader in some situations. Eventually, the new state gets propagated to all healthy servers, and they take action on it.

If you want to be sure that each server has processed an administrative operation before running the next

action, you can use the `WAIT` keyword. `WAIT` makes the statement not return until all servers have seen the transaction, and their reconciler has finished processing it.

Example of database creation with `WAIT`

```
CREATE DATABASE foo WAIT
```

> **Note:**
>
> When a statement with `WAIT` returns, it does not mean that the operation was necessarily successful everywhere. `WAIT` just ensures the operation has been processed. For example, a request to start a database is considered *processed* if the reconciler tried to start it, but the database failed and went into the dirty state.

Errors

An operation might only succeed on some servers. For instance, some servers might be offline when a database is created, or a disk error might cause a server to fail to start a database. In this situation, the operation has not *failed* as a whole, since the database may even be running (although with less fault tolerance than you intended).

Some failed operations can just be re-tried: for example, `START DATABASE` can be re-run if not all members started successfully, and you think the cause of that has been resolved. Other failures only require the underlying problem to be resolved. When the issue is fixed, the reconciler will make the intended change, because it is still trying to achieve the requested state. For example, if a new server fails during a `DEALLOCATE DATABASES FROM SERVER` (meaning a database cannot be safely moved), fixing the new server should be enough to resolve the problem, as the new server will start its copy of the database as soon as the target is ready, allowing the deallocating server to shut down its copy.

Cluster server discovery

In order to join a running cluster, any new member must know the addresses of at least some of the other servers in the cluster. This information is necessary to connect to the servers, run the discovery protocol, and obtain all the information about the cluster.

Neo4j provides several mechanisms for cluster members to discover each other and form a cluster based on the configuration and the environment in which the cluster is running, as well as the version of Neo4j being used.

> **Important:**
>
> In Neo4j 5.23, the discovery service v2 has been introduced. The new version, v2, is recommended for new deployments.
>
> The current version, v1, has been deprecated and will be removed in a future release. Therefore, it is highly recommended to move from v1 to v2. For details, see Moving from discovery service v1 to v2.

The new version v2 introduces the following configuration settings to control the discovery service:

- `dbms.cluster.discovery.v2.endpoints` to specify the list of server addresses for discovery when using version v2.
- `dbms.cluster.discovery.version` to specify the version of the discovery service. Possible values are `V1_ONLY`, `V2_ONLY`, `V1_OVER_V2`, and `V2_OVER_V1`.
- `dbms.kubernetes.discovery.v2.service_port_name` to specify the name of the service port name used in the Kubernetes API when using version v2.

`Ports` have also changed in discovery service v2. Port `5000`, the discovery v1 port, is no longer used. Instead, discovery service v2 uses port `6000` - the cluster internal traffic port.

The following sections describe the different methods for server discovery, configuration settings, and examples of how to move from discovery service v1 to v2.

Methods for server discovery

Depending on the type of `dbms.cluster.discovery.resolver_type` currently in use, the discovery service can use a list of server addresses, DNS records, or Kubernetes services to discover other servers in the cluster. The discovery configuration is used for initial discovery and to continuously exchange information about changes to the topology of the cluster.

Note:

Regardless of the method used to resolve the list of server addresses, ensure that the endpoint for each server hosting the `system` database in primary mode is included.

Discovery using a list of server addresses

If the addresses of the other cluster members are known upfront, they can be listed explicitly. However, this method has limitations, such as:

- If servers are replaced and the new members have different addresses, the list becomes outdated. An outdated list can be avoided by ensuring that the new members can be reached via the same address as the old members, but this is not always practical.
- Under some circumstances the addresses are unknown when configuring the cluster. This can be the case, for example, when using container orchestration to deploy a cluster.

To use this method, set `dbms.cluster.discovery.resolver_type=LIST` and hard code the addresses in the configuration of each server. For example:

```
dbms.cluster.discovery.resolver_type=LIST

server.cluster.advertised_address=server01.example.com:6000
dbms.cluster.discovery.v2.endpoints=server01.example.com:6000,server02.example.com:6000,server03
.example.com:6000

dbms.cluster.discovery.version=V2_ONLY
```

```
dbms.cluster.discovery.resolver_type=LIST

server.discovery.advertised_address=server01.example.com:5000
dbms.cluster.discovery.endpoints=server01.example.com:5000,server02.example.com:5000,server03.ex
ample.com:5000

dbms.cluster.discovery.version=V1_ONLY
```

An example of using this method with discovery service v1 is illustrated by Configure a cluster with three servers.

Discovery using DNS with multiple records

Where it is not practical or possible to explicitly list the addresses of cluster members to discover, you can use DNS-based mechanisms. In such cases, a DNS record lookup is performed when a server starts up based on configuration settings. Once a server has joined a cluster, further topology changes are communicated amongst the servers in the cluster as part of the discovery service.

The following DNS-based mechanisms can be used to get the addresses of other servers in the cluster for discovery:

`dbms.cluster.discovery.resolver_type=DNS`

With this configuration, the initial discovery members are resolved from *DNS A* records to find the IP addresses to contact. In version 1, the value of `dbms.cluster.discovery.endpoints` should be set to a single domain name and the port of the discovery service, while in version 2, the value of `dbms.cluster.discovery.v2.endpoints` should be set to a single domain name and the cluster advertised port. For example:

```
dbms.cluster.discovery.resolver_type=DNS

server.cluster.advertised_address=server01.example.com:6000
dbms.cluster.discovery.v2.endpoints=cluster01.example.com:6000

dbms.cluster.discovery.version=V2_ONLY
```

```
dbms.cluster.discovery.resolver_type=DNS

server.discovery.advertised_address=server01.example.com:5000
dbms.cluster.discovery.endpoints=cluster01.example.com:5000

dbms.cluster.discovery.version=V1_ONLY
```

When a DNS lookup is performed, the domain name returns an A record for every server in the cluster, where each A record contains the IP address of the server. The configured server uses all the IP addresses from the A records to join or form a cluster.

> **Note:**
>
> The discovery port must be the same on all servers when using this configuration. If this is not possible, consider using the discovery type `SRV`.

`dbms.cluster.discovery.resolver_type=SRV`

With this configuration, the initial discovery members are resolved from *DNS SRV* records to find the IP addresses/hostnames and discovery service ports (v1) **or** cluster advertised ports (v2) to contact.

The value of `dbms.cluster.discovery.endpoints` must be set to a single domain name and the port set to `0`. The domain name returns a single SRV record when a DNS lookup is performed. For example:

```
dbms.cluster.discovery.resolver_type=SRV

server.cluster.advertised_address=server01.example.com:6000
dbms.cluster.discovery.v2.endpoints=cluster01.example.com:0

dbms.cluster.discovery.version=V2_ONLY
```

The SRV record returned by DNS should contain the IP address or hostname, and the **cluster** port for the servers to be discovered. The configured server uses all the addresses from the SRV record to join or form a cluster.

```
dbms.cluster.discovery.resolver_type=SRV

server.discovery.advertised_address=server01.example.com:5000
dbms.cluster.discovery.endpoints=cluster01.example.com:0

dbms.cluster.discovery.version=V1_ONLY
```

The SRV record returned by DNS should contain the IP address or hostname, and the **discovery** port for the servers to be discovered. The configured server uses all the addresses from the SRV record to join or form a cluster.

Discovery in Kubernetes

A special case is when a cluster is running in Kubernetes and each server is running as a Kubernetes service. Then, the addresses of the other servers can be obtained using the List Service API, as described in the Kubernetes API documentation.

The following settings are used to configure for this scenario:

- Set `dbms.cluster.discovery.resolver_type=K8S`.
- Set `dbms.kubernetes.label_selector` to the label selector for the cluster services. For more information, see the Kubernetes official documentation.
- Depending on your discovery service version, set either `dbms.kubernetes.service_port_name` (v1), or

`dbms.kubernetes.discovery.v2.service_port_name` (v2) to the name of the service port used in the Kubernetes service definition for the Core's discovery port. For more information, see the Kubernetes official documentation.

With this configuration, `dbms.cluster.discovery.endpoints` is not used and any value assigned to it is ignored.

> **Note:**
>
> - The pod running Neo4j must use a service account that has permission to list services. For further information, see the Kubernetes documentation on RBAC authorization or ABAC authorization.
> - The configured `server.cluster.advertised_address` must exactly match the Kubernetes-internal DNS name, which is of the form `<service-name>.<namespace>.svc.cluster.local`.

The discovery configuration is used for initial discovery and to continuously exchange information about changes to the topology of the cluster.

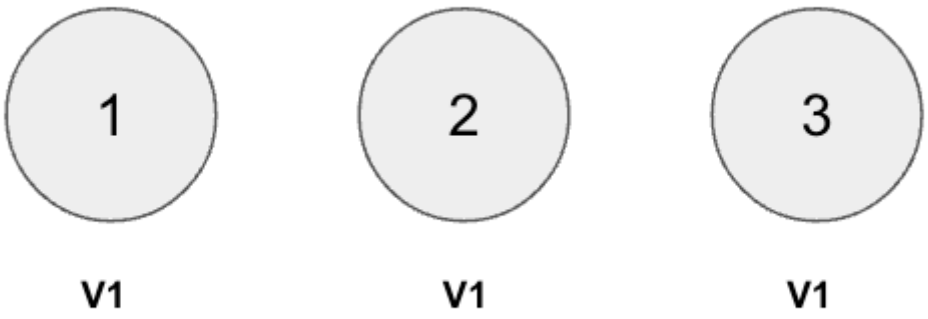
Moving from discovery service v1 to v2

From Neo4j 5.23, you can move from the current discovery service v1 to the new version v2. The v1 and v2 discovery services are designed to be able to run in parallel. They are completely independent of each other, thus allowing you to keep the cluster functioning while switching over from v1 to v2.

There are four ways to move from the current discovery service v1 to the new version v2 depending on the environment and the requirements of the cluster.

Preparation

The following examples assume that you have a running cluster with three servers and you want to move from the current discovery service v1 to the new version v2.



Before moving from the current discovery service v1 to the new version v2, ensure that the new settings are added to the configuration depending on the type of `dbms.cluster.discovery.resolver_type` in use:

- If `dbms.cluster.discovery.resolver_type=LIST`, set `dbms.cluster.discovery.v2.endpoints` to a comma-separated list of `server.cluster.advertised_address`. It is important that both `dbms.cluster.discovery.endpoints` and `dbms.cluster.discovery.v2.endpoints` are set during the operation. For more information, see Discovery using a list of server addresses.
- If `dbms.cluster.discovery.resolver_type=DNS`, set `dbms.cluster.discovery.v2.endpoints` to a single domain name and the cluster port. Alternatively, if `dbms.cluster.discovery.resolver_type=SRV`, set `dbms.cluster.discovery.v2.endpoints` to a single domain name and the port set to `0`. It is important that both `dbms.cluster.discovery.endpoints` and `dbms.cluster.discovery.v2.endpoints` are set during the operation. For more information, see Discovery using DNS with multiple records.
- If `dbms.cluster.discovery.resolver_type=K8S`, set `dbms.kubernetes.discovery.v2.service_port_name` to the name of the service port used in the Kubernetes service definition for the cluster port. It is important that both `dbms.kubernetes.service_port_name` and `dbms.kubernetes.discovery.v2.service_port_name` are set during the operation. For more information, see Discovery in Kubernetes.

Procedure for moving an entire cluster

The first way to migrate from discovery service v1 to v2 is to use a semi-automated procedure, which simplifies the migration process. It automates the approach described in the next section.

1. Make sure that server side routing is enabled as detailed in Server-side routing section
2. For all the servers, ensure that new settings are updated to the configuration as detailed the Preparation section.

   As an example, for those using the list resolver, the settings for all the servers should include:

   ```
   dbms.cluster.discovery.resolver_type=LIST

   dbms.cluster.discovery.endpoints=server01.example.com:5000,server02.example.com:5000,server03.example.
   com:5000
   dbms.cluster.discovery.v2.endpoints=server01.example.com:6000,server02.example.com:6000,server03.examp
   le.com:6000
   ```

3. In Cypher Shell, connect to the `system` database of any server:

   ```
   ./cypher-shell -a bolt://localhost:7687 -d system
   ```

4. Run the procedure:

   ```
   CALL dbms.cluster.showParallelDiscoveryState();
   ```

   The output indicates mode `V1_ONLY`, i.e., only `V1` is running on this server.

   Expected result

   ```
   +-------------------------------------------------------------+
   | mode      | stateComparison | v1ServerCount | v2ServerCount |
   +-------------------------------------------------------------+
   | "V1_ONLY" | "N/A"           | "3"           | "N/A"         |
   +-------------------------------------------------------------+
   ```

5. Run the following procedure to turn on `V2` in the background for all servers, but keep `V1` running in the foreground. If the procedure fails, read the error message and resolve the issue manually. One possible reason for failure is that some servers might have had inconsistent `dbms.cluster.discovery.version` settings initially.

```
CALL dbms.cluster.moveToNextDiscoveryVersion();
```

6. Check the state again:

```
CALL dbms.cluster.showParallelDiscoveryState();
```

Now the returned mode for this server must be `V1_OVER_V2` and the `stateComparison` must show that the states are matching. If they are not, wait and try again till matching.

Expected result

```
+-------------------------------------------------------------------+
| mode         | stateComparison | v1ServerCount | v2ServerCount |
+-------------------------------------------------------------------+
| "V1_OVER_V2" | "Matching"      | "3"           | "3"           |
+-------------------------------------------------------------------+
```

7. Once again, run the following procedure to turn on `V1` in the background for all servers, but keep `V2` running in the foreground.

```
CALL dbms.cluster.moveToNextDiscoveryVersion();
```

8. Check that transition to `V2_OVER_V1` was successful:

```
CALL dbms.cluster.showParallelDiscoveryState();
```

Now the returned mode for this server must be `V2_OVER_V1` and the `stateComparison` must show that the states are matching. If they are not, wait and try again till matching.

Expected result

```
+-------------------------------------------------------------------+
| mode         | stateComparison | v1ServerCount | v2ServerCount |
+-------------------------------------------------------------------+
| "V2_OVER_V1" | "Matching"      | "3"           | "3"           |
+-------------------------------------------------------------------+
```

9. Finally, turn off `V1` by running the following procedure:

```
CALL dbms.cluster.moveToNextDiscoveryVersion();
```

10. Check that transition to `V2_ONLY` was successful:

```
CALL dbms.cluster.showParallelDiscoveryState();
```

Note that `stateComparison` is `N/A` because you do not have `V1` to compare states anymore.

Expected result

```
+-------------------------------------------------------------+
| mode      | stateComparison | v1ServerCount | v2ServerCount |
+-------------------------------------------------------------+
| "V2_ONLY" | "N/A"           | "N/A"         | "3"           |
+-------------------------------------------------------------+
```

Important

Important:

Remember to update the *neo4j.conf* files for all the servers. The switching using procedures does not persist anything to disk. Therefore, when a server restarts, it starts right back with only v1 running. As such, ensure that `dbms.cluster.discovery.version=V2_ONLY`, and that `dbms.cluster.discovery.v2.endpoints` or `dbms.kubernetes.discovery.v2.service_port_name` are set as required, so that the servers start with v2 running on the next restart.

Per server procedure

Important:

If the semi-automated procedure fails, or you need to change individual servers to resolve an issue, this approach will let you do so.

1. For all the servers, ensure that new settings are updated to the configuration as detailed the Preparation section.

   As an example, for those using the list resolver, the settings for all the servers should include:

   ```
   dbms.cluster.discovery.resolver_type=LIST

   dbms.cluster.discovery.endpoints=server01.example.com:5000,server02.example.com:5000,server03.example.
   com:5000
   dbms.cluster.discovery.v2.endpoints=server01.example.com:6000,server02.example.com:6000,server03.examp
   le.com:6000
   ```

2. In Cypher Shell, connect to the `system` database of server01 using `bolt://`. It is important to connect via `bolt://` because otherwise the procedure might be routed and executed not on the intended server.

   ```
   ./cypher-shell -a bolt://localhost:7687 -d system
   ```

3. Run the procedure:

   ```
   CALL dbms.cluster.showParallelDiscoveryState();
   ```

The output indicates mode `V1_ONLY`, i.e., only v1 is running on this server.

Expected result

```
+-------------------------------------------------------------+
| mode      | stateComparison | v1ServerCount | v2ServerCount |
+-------------------------------------------------------------+
| "V1_ONLY" | "N/A"           | "3"           | "N/A"         |
+-------------------------------------------------------------+
```

4. Run the following procedure to turn on v2 in the background, but keep v1 running in the foreground:

```
CALL dbms.cluster.switchDiscoveryServiceVersion("V1_OVER_V2");
```

5. Check the state again:

```
CALL dbms.cluster.showParallelDiscoveryState();
```

Now the returned mode for this server must be `V1_OVER_V2` and the `stateComparison` must show that the states are not matching yet.

Expected result

```
+-------------------------------------------------------------------------------------------------------
--+
| mode         | stateComparison                                      | v1ServerCount |
v2ServerCount |
+-------------------------------------------------------------------------------------------------------
--+
| "V1_OVER_V2" | "States are not matching after PT1M9.518S: (score:18)" | "3"           | "1"
|
+-------------------------------------------------------------------------------------------------------
--+
```

The score is a measure of how different the states are. `serverCounts` displays how many servers can be found by v1 and v2 of the discovery service, respectively. The score is 0 when the states are matching. When some members are running just one of the discovery services (v1 or v2) and other members run both, the score stays permanently high. This is no reason for worry.

6. To fulfill this convergence, in different terminals, connect to server02 and server03 via `bolt://` and repeat steps 3 and 4 on both of them.
7. Check the state on all servers again. It should show that the states are `Matching`. Both `serverCounts` should be 3.

Expected result

```
+-------------------------------------------------------------+
| mode         | stateComparison | v1ServerCount | v2ServerCount |
+-------------------------------------------------------------+
| "V1_OVER_V2" | "Matching"      | "3"           | "3"           |
+-------------------------------------------------------------+
```

8. On all three servers, run:

```
CALL dbms.cluster.switchDiscoveryServiceVersion("V2_OVER_V1");
```

At this point, v2 is the service that is running the cluster, with v1 running in the background.

Expected result

```
+-------------------------------------------------------------------+
| mode         | stateComparison | v1ServerCount | v2ServerCount |
+-------------------------------------------------------------------+
| "V2_OVER_V1" | "Matching"      | "3"           | "3"           |
+-------------------------------------------------------------------+
```

9. Finally, turn off v1 by running the following procedure on all three servers:

```
CALL dbms.cluster.switchDiscoveryServiceVersion("V2_ONLY");
```

10. Verify that `CALL dbms.cluster.showParallelDiscoveryState();` now shows `V2_ONLY` running. Note that `stateComparison` is `N/A` because you do not have v1 to compare states anymore.

Expected result

```
+-------------------------------------------------------------------+
| mode      | stateComparison | v1ServerCount | v2ServerCount |
+-------------------------------------------------------------------+
| "V2_ONLY" | "N/A"           | "N/A"         | "3"           |
+-------------------------------------------------------------------+
```

> *Important*
>
> **Important:**
>
> Remember to update the *neo4j.conf* files for all the servers. The switching using procedures does not persist anything to disk. Therefore, when a server restarts, it starts right back with only v1 running. As such, ensure that `dbms.cluster.discovery.version=V2_ONLY`, and that `dbms.cluster.discovery.v2.endpoints` or `dbms.kubernetes.discovery.v2.service_port_name` are set as required, so that the servers start with v2 running on the next restart.

In-place rolling

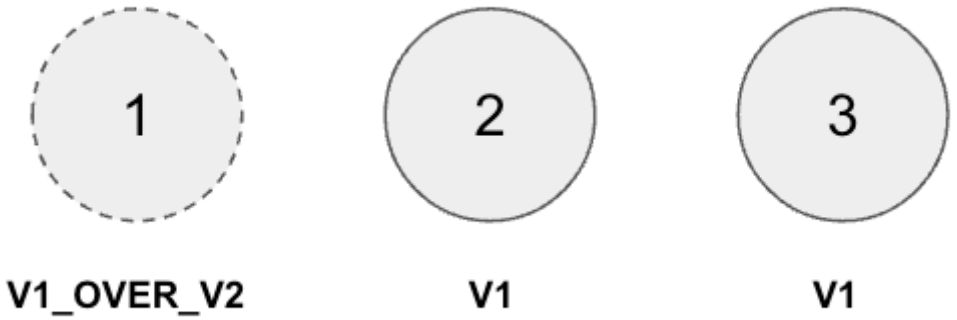
> **Important:**
>
> In-place rolling reduces fault tolerance temporarily because you are restarting a running server. To keep fault-tolerance, you can introduce a fourth server temporarily.

1. For all the servers, ensure that new settings are added to the configuration as detailed the Preparation section.

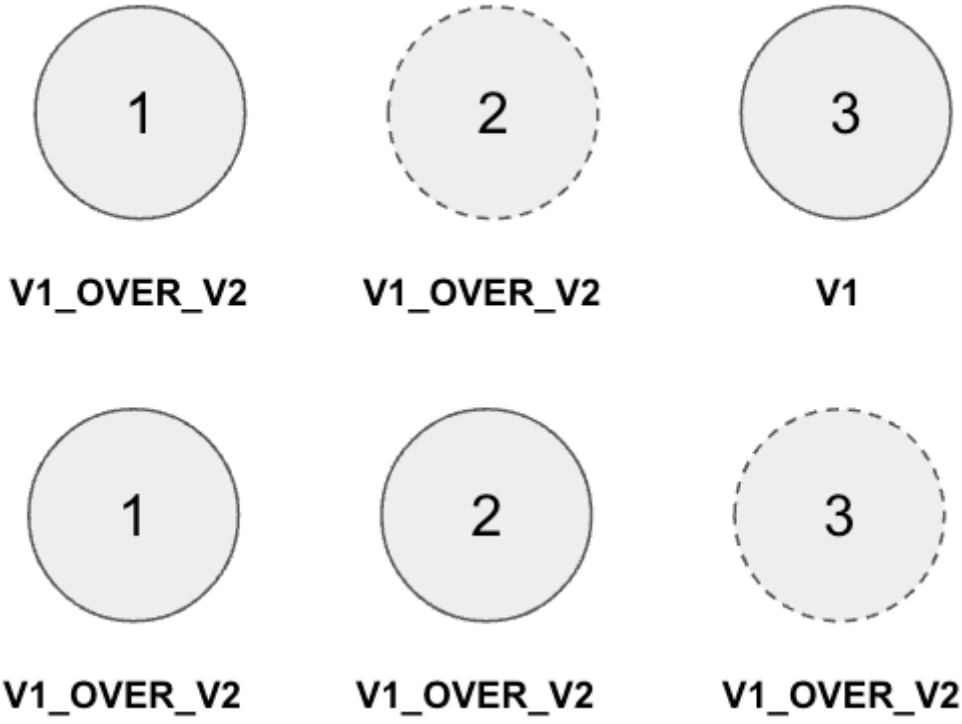
   As an example, for those using the list resolver, the settings for all the servers should include:

```
dbms.cluster.discovery.resolver_type=LIST

dbms.cluster.discovery.endpoints=server01.example.com:5000,server02.example.com:5000,server03.example.
com:5000
dbms.cluster.discovery.v2.endpoints=server01.example.com:6000,server02.example.com:6000,server03.examp
le.com:6000
```

2. Restart server01 with the new setting `dbms.cluster.discovery.version=V1_OVER_V2`.



3. Run `SHOW SERVERS` and ensure that all three members are alive.
4. Then repeat steps 2 and 3 for server02 and server03. Ensure that they are set to `dbms.cluster.discovery.version=V1_OVER_V2`. Restart them sequentially, not in parallel.



5. Using `bolt://`, connect to the system database of servers 1, 2, 3, and run the following procedure. This can be done using via `./cypher-shell -a bolt://localhost:7687 -d system` for example.

```
CALL dbms.cluster.showParallelDiscoveryState();
```

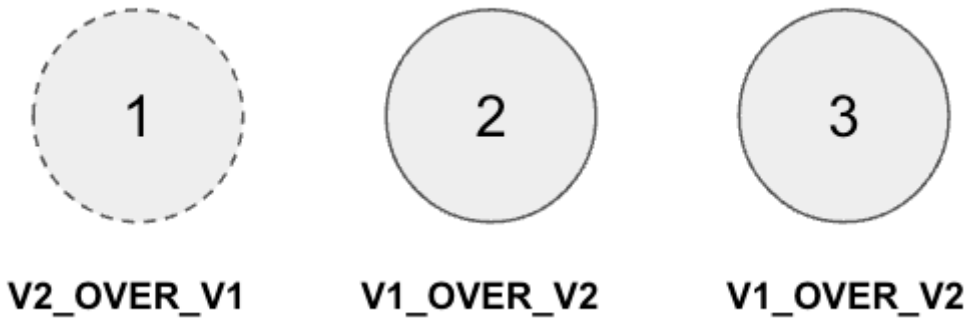
They should display "Matching" in the `stateComparison` column.

Expected result

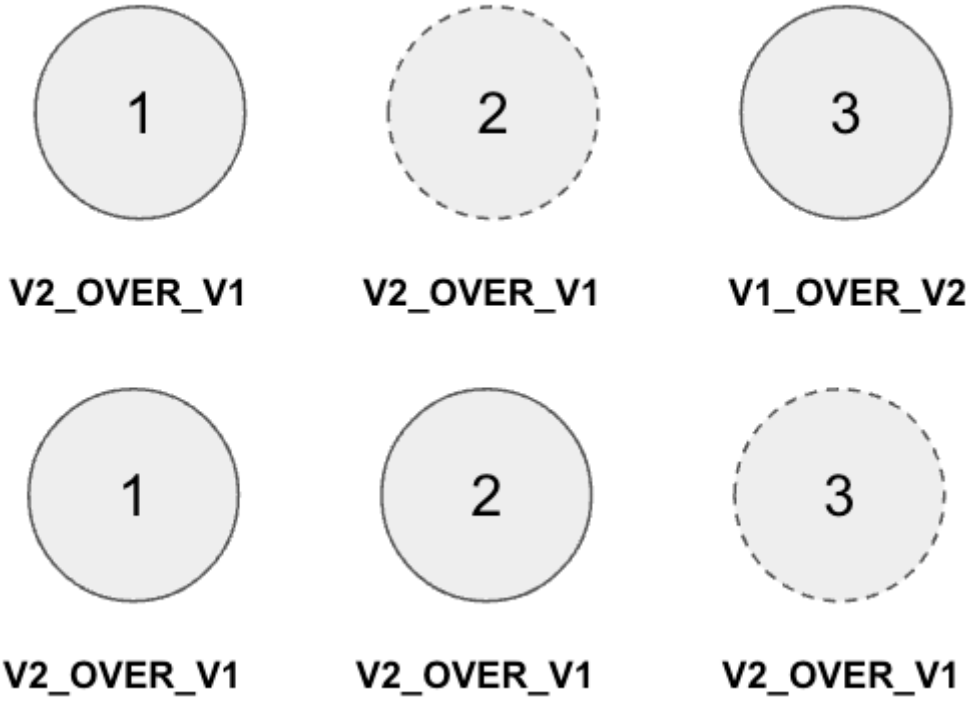
```
+-------------------------------------------------------------------+
| mode         | stateComparison | v1ServerCount | v2ServerCount |
+-------------------------------------------------------------------+
| "V1_OVER_V2" | "Matching"      | "3"           | "3"           |
+-------------------------------------------------------------------+
```

If they are not, wait and try again.

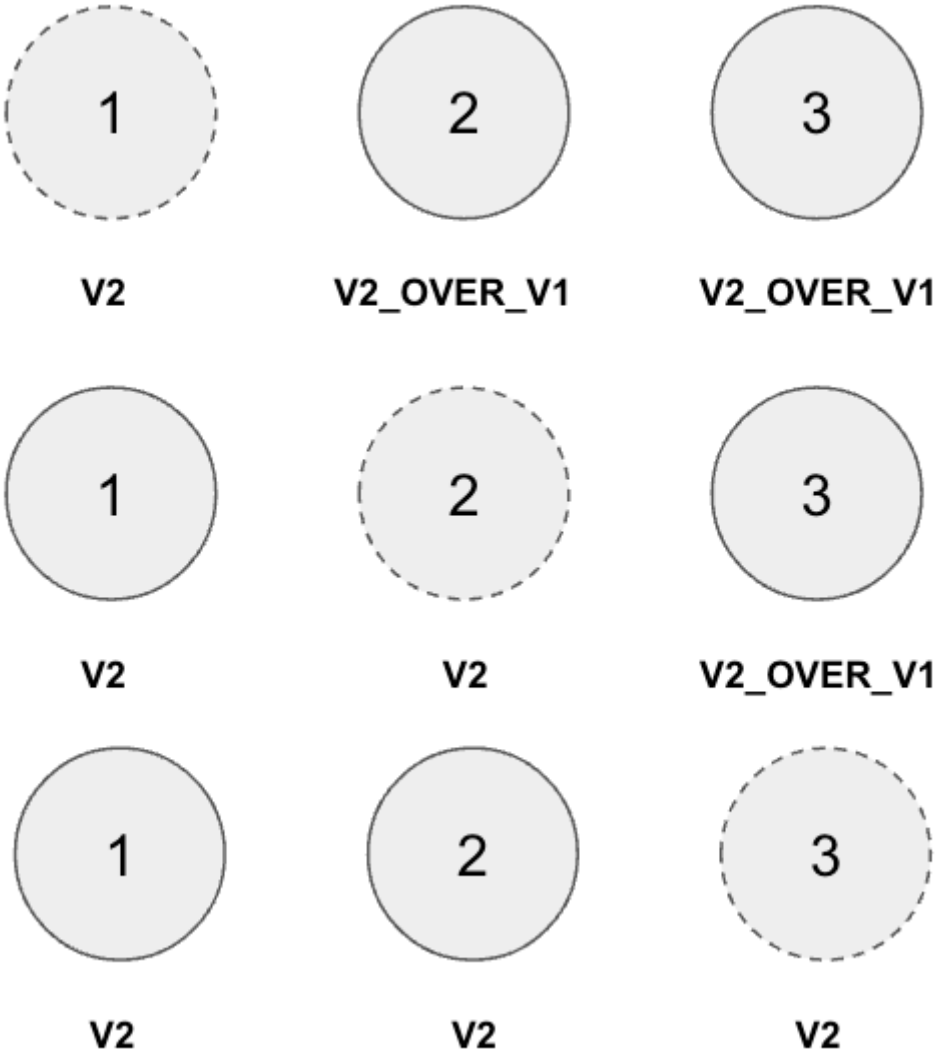
6. Restart server01 again with the new setting `dbms.cluster.discovery.version=V2_OVER_V1`.



7. Run `SHOW SERVERS` and ensure that all three members are alive.
8. Then repeat steps 6 and 7 for servers 2 and 3. Ensure that they are set to `dbms.cluster.discovery.version=V2_OVER_V1`. Restart them sequentially, not in parallel.



9. Similar to step 5, verify that `stateComparison` shows `Matching`.
10. Repeat steps 6, 7, 8, 9, restarting servers 1, 2, and 3 sequentially, with the new setting `dbms.cluster.discovery.version=V2_ONLY`

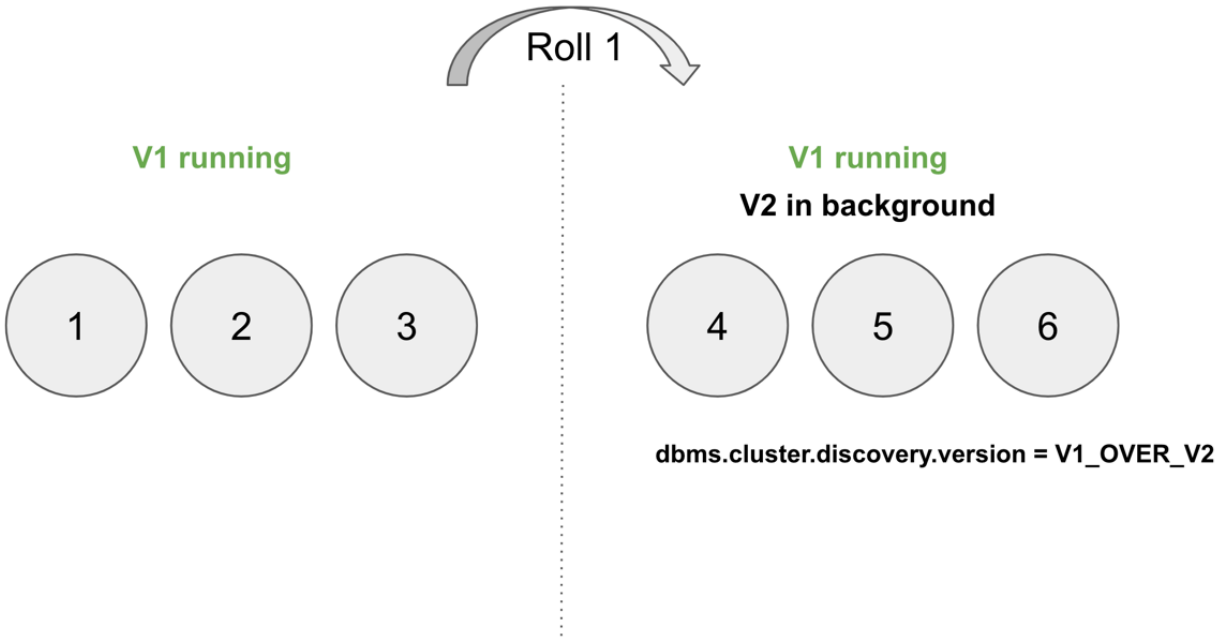


11. Verify that `CALL dbms.cluster.showParallelDiscoveryState()` now shows `V2_ONLY` running. Note that `stateComparison` is `N/A` because you do not have v1 to compare states anymore.

New server rolling

The new server rolling requires three running servers and three new servers.

1. Start up the three new servers with the setting `dbms.cluster.discovery.version=V1_OVER_V2`.



The new servers should have settings which are updated as detailed in the Preparation section. The discovery addresses should include addresses of the new members, and the previous members.

As an example, for those using the list resolver, the settings for the new servers should include:

```
dbms.cluster.discovery.resolver_type=LIST
dbms.cluster.discovery.version=V1_OVER_V2

dbms.cluster.discovery.endpoints=server01.example.com:5000,server02.example.com:5000,server03.example.
com:5000,server04.example.com:5000,server05.example.com:5000,server06.example.com:5000
dbms.cluster.discovery.v2.endpoints=server01.example.com:6000,server02.example.com:6000,server03.examp
le.com:6000,server04.example.com:6000,server05.example.com:6000,server06.example.com:6000
```

2. Using `bolt://`, connect to the system database of servers 4, 5, 6, and run the following procedure. This can be done using via `./cypher-shell -a bolt://localhost:7685 -d system` for example.

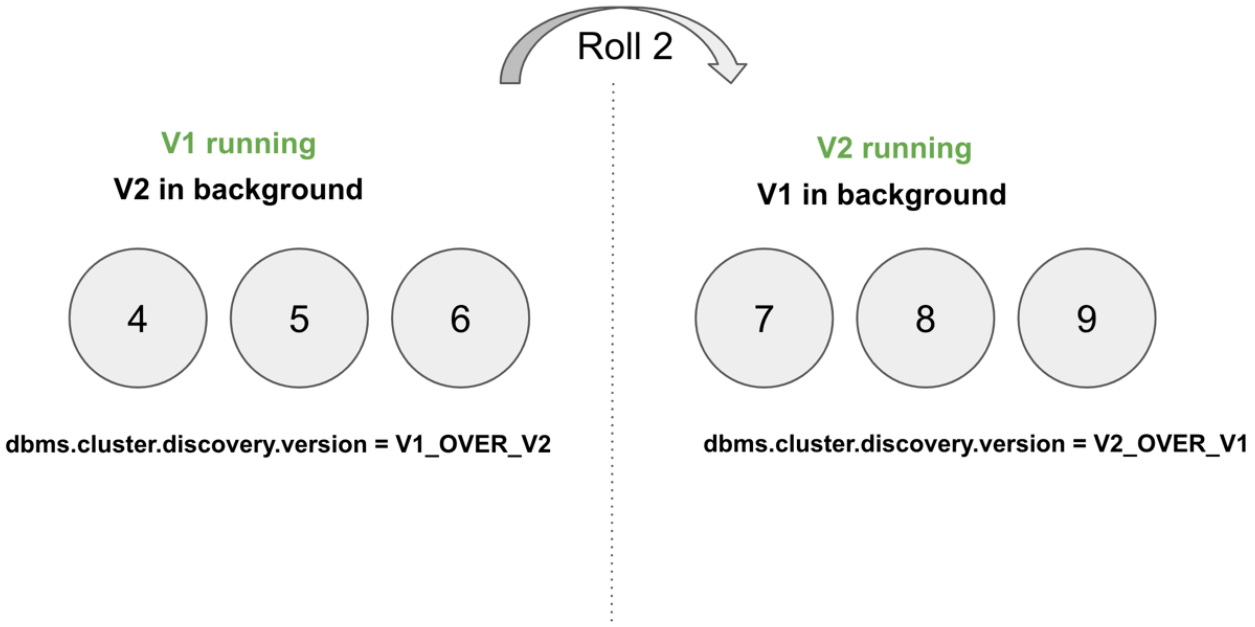
```
CALL dbms.cluster.showParallelDiscoveryState();
```

The expected result should display `v2ServerCount` as 3. `stateComparison` is not expected to match at this stage because the original servers are not visible to the V2 discovery service.

Expected result

```
 +-------------------------------------------------------------------------------------------------------
-----+
 | mode         | stateComparison                                          | v1ServerCount |
v2ServerCount |
 +-------------------------------------------------------------------------------------------------------
-----+
 | "V1_OVER_V2" | "States are not matching after PT55M36.693S: (score:29)" | "6"           | "3"
|
 +-------------------------------------------------------------------------------------------------------
-----+
```

3. Deallocate, drop, and shut down servers 1, 2, 3.
4. Start up servers 7, 8, 9, this time with the setting `dbms.cluster.discovery.version=V2_OVER_V1`.



The discovery addresses in the settings should include addresses of the new members, and the previous members.

As an example, for those using the list resolver, the settings for the new servers should include:

```
dbms.cluster.discovery.resolver_type=LIST
dbms.cluster.discovery.version=V2_OVER_V1

dbms.cluster.discovery.endpoints=server04.example.com:5000,server05.example.com:5000,server06.example.
com:5000,server07.example.com:5000,server08.example.com:5000,server09.example.com:5000
dbms.cluster.discovery.v2.endpoints=server04.example.com:6000,server05.example.com:6000,server06.examp
le.com:6000,server07.example.com:6000,server08.example.com:6000,server09.example.com:6000
```

5. Using `bolt://`, connect to the system database of servers 7, 8, 9 and run the following procedure:

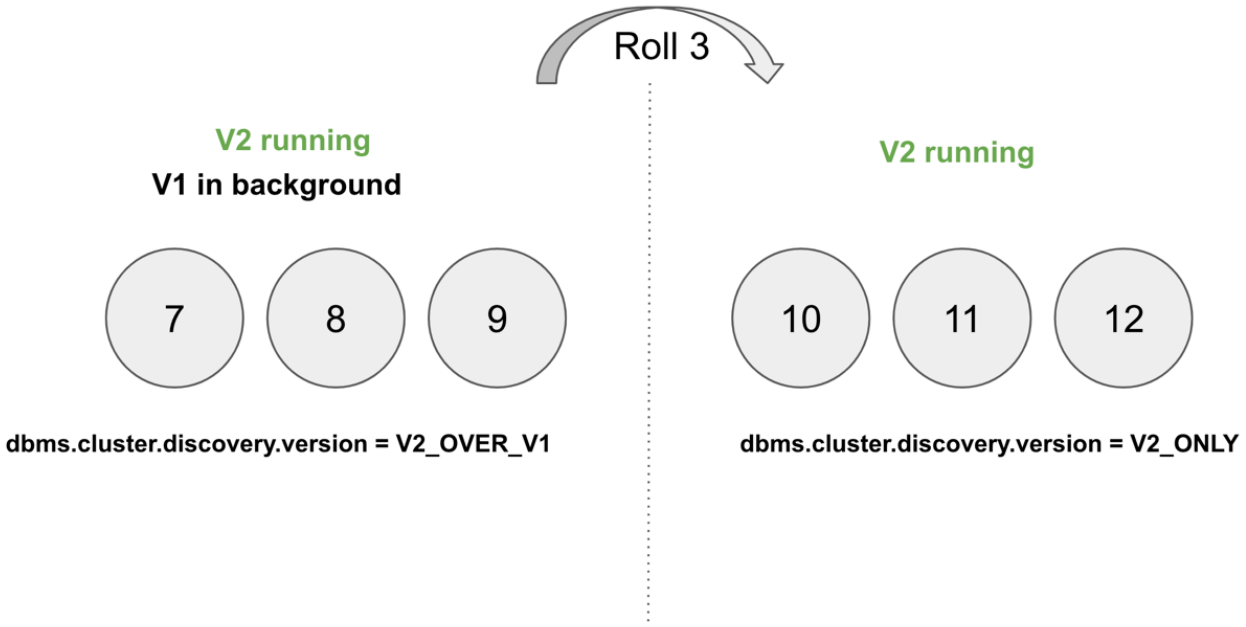
```
CALL dbms.cluster.showParallelDiscoveryState();
```

The output should display `Matching` in the `stateComparison` column. If they are not, wait and try again till matching.

Expected result

```
+-------------------------------------------------------------------+
| mode         | stateComparison | v1ServerCount | v2ServerCount |
+-------------------------------------------------------------------+
| "V2_OVER_V1" | "Matching"      | "6"           | "6"           |
+-------------------------------------------------------------------+
```

6. Deallocate, drop, and shut down servers 4, 5, and 6.
7. Start up servers 10, 11, 12, this time with the setting `dbms.cluster.discovery.version=V2_ONLY`.



The discovery addresses in the settings should include addresses of the new members, and the previous members. Note that only the v2 settings are required.

As an example, for those using the list resolver, the settings for the new servers should include:

```
dbms.cluster.discovery.resolver_type=LIST
dbms.cluster.discovery.version=V2_ONLY

dbms.cluster.discovery.v2.endpoints=server07.example.com:6000,server08.example.com:6000,server09.examp
le.com:6000,server10.example.com:6000,server11.example.com:6000,server12.example.com:6000
```

8. Deallocate, drop, and shut down servers 7, 8, 9.
9. Finally, using `bolt://`, connect to the system database of servers 10, 11, 12, and run the following procedure to check the version of the discovery service:

```
CALL dbms.cluster.showParallelDiscoveryState();
```

Expected result

```
+-------------------------------------------------------------+
| mode      | stateComparison | v1ServerCount | v2ServerCount |
+-------------------------------------------------------------+
| "V2_ONLY" | "N/A"           | "N/A"         | "3"           |
+-------------------------------------------------------------+
```

Monitoring the progress and metrics

When moving from the current discovery service v1 to the new version v2, you can monitor the progress using the procedure `CALL dbms.cluster.showParallelDiscoveryState()`. This procedure shows the current state of the discovery service on the server and the difference score between the states of the v1 and v2 discovery services. The difference score is a measure of how different the states are. The difference score reported by the procedure does not always stay at 0. Here are some scenarios to consider:

- In the case of a cluster, when some members are running just one of the discovery services (v1 or v2) and other members run both, the score will stay permanently high. This is no reason for worry.
- When changes happen in the cluster (like start/stop of a database/server or a leader switch) the difference score will temporarily be greater than 0. It should reach 0 relatively fast again.
- If the difference score is greater than 0 for a longer period, the actual difference is printed in the *debug.log*.

You can also use the following metrics to monitor the discovery service:

- Discovery metrics v1
- Discovery metrics v2

Leadership, routing, and load balancing

Elections and leadership

The servers in a cluster use the Raft protocol to ensure consistency and safety. An implementation detail of Raft is that it uses a *Leader* role to impose an ordering on an underlying log with other instances acting as *Followers* which replicate the leader's state. Specifically in Neo4j, this means that writes to the database are ordered by the server currently playing the *Leader* role for the respective database.

Only servers hosting a database in primary mode can be elected leaders for that database, provided that the cluster contains more than one primary server. If a Neo4j DBMS cluster contains multiple databases, each one of those databases operates within a logically separate Raft group, and therefore each has an individual leader. This means that a server may act both as *Leader* for some databases, and as *Follower* for other databases.

If a follower has not heard from the leader for a while, then it can initiate an election and attempt to become the new leader. The follower makes itself a *Candidate* and asks other servers to vote for it. If it can get a majority of the votes, then it assumes the leader role. Servers do not vote for a candidate which is less up-to-date than itself. There can only be one leader at any time per database, and that leader is guaranteed to have the most up-to-date log.

Elections are expected to occur during the normal running of a cluster and they do not pose an issue in and of itself. If you are experiencing frequent re-elections and they are disturbing the operation of the cluster then you should try to figure out what is causing them. Some common causes are environmental issues (e.g. a flaky networking) and work overload conditions (e.g. more concurrent queries and transactions than the hardware can handle).

Leadership balancing

Write transactions are always routed to the leader for the respective database. As a result, unevenly distributed leaderships may cause write queries to be disproportionately directed to a subset of servers. By default, Neo4j avoids this by automatically transferring database leaderships so that they are evenly distributed throughout the cluster. Additionally, Neo4j automatically transfers database leaderships away from instances where those databases are configured to be read-only using server.databases.read_only or similar.

Client-side routing

Client-side routing is when the database client takes control over which cluster member to send specific requests to. Typically this would be to make sure that write operations are sent to the server that can write for the target database, and that read operations are sent to other servers.

Client-side routing is based on getting a routing table from a cluster member, and then using that information to make the routing decisions. A routing table contains information about the writers, readers, and routers for a specific database. There is usually one *writer*, though there may be none if the database is read only or unhealthy. With the default configuration, all other servers that host the database are considered *readers*, i.e. the writer is not in the list of readers. This is to let it focus on the write load and not have to manage two kinds of interactions. Typically, all servers that host the database are listed as *routers*, which are servers that can be contacted to get a new routing table for that database.

Neo4j Drivers retrieve a routing table the first time they attempt to connect to a database, and fetch a fresh one after the configured time-to-live, or if it seems the routing table has got out of date. For example, if the routing table lists `server-3` as the writer for the database, but write requests get rejected with a *not able to write* error, the driver may decide to get a new routing table, because the writer could be a different server.

For lower level details about getting routing tables, refer to the Bolt protocol documentation.

Routing policies

You can control the routing table that servers provide by using routing policies. Policies filter the full set of possible servers for each category according to the rules you define. For example, this can be used to preferentially route to a local data centre, or to specific large machines, depending on your policies.

Server-side routing

Server-side routing is a complement to the client-side routing.

In a cluster deployment of Neo4j, Cypher queries may be directed to a cluster member that is unable to run the given queries. With server-side routing enabled, such queries are rerouted internally to a cluster member that is expected to be able to run them. This situation can occur for write-transaction queries when they address a database for which the receiving cluster member is not the leader.

The cluster role for cluster members is per database. Thus, if a write-transaction query is sent to a cluster member that is not the leader for the specified database (specified either via the Bolt Protocol or with Cypher `USE` clause), server-side routing is performed if properly configured.

Server-side routing is enabled by the DBMS, by setting `dbms.routing.enabled=true` for each cluster member. The listen address (`server.routing.listen_address`) and advertised address (`server.routing.advertised_address`) also need to be configured for server-side routing communication.

Client connections need to state that server-side routing should be used and this is available for Neo4j Drivers and HTTP API.

> **Note:**

Neo4j Drivers can only use server-side routing when the `neo4j://` URI scheme is used. The Drivers do not perform any routing when the `bolt://` URI scheme is used, instead connecting directly to the specified host.

On the cluster-side you must fulfill the following prerequisites to make server-side routing available:

- Set `dbms.routing.enabled=true` on each member of the cluster.
- Configure `server.routing.listen_address`, and provide the advertised address using `server.routing.advertised_address` on each member.
- Optionally, you can set `dbms.routing.default_router=SERVER` on each member of the cluster.

The last prerequisite enforces server-side routing on the clients by sending out a routing table with exactly one entry to the client. Therefore, `dbms.routing.default_router=SERVER` configures a cluster member to make its routing table behave like a standalone instance. The implication is that if a Neo4j Driver connects to this cluster member, then the Neo4j Driver sends all requests to that cluster member. Please note that the default configuration for `dbms.routing.default_router` is `dbms.routing.default_router=CLIENT`. See `dbms.routing.default_router` for more information.

The HTTP-API of each member benefits from these settings automatically.

Server-side routing connector configuration

Rerouted queries are communicated over the Bolt Protocol using a designated communication channel. The receiving end of the communication is configured using the following settings:

- `dbms.routing.enabled`
- `server.routing.listen_address`
- `server.routing.advertised_address`

Server-side routing driver configuration

Server-side routing uses the Neo4j Java driver to connect to other cluster members. This driver is configured with settings of the format:

- `dbms.routing.driver.<setting>`

Server-side routing encryption

Encryption of server-side routing communication is configured by the cluster SSL policy. For more information, see Cluster Encryption.

Intra-cluster encryption

Caution:

Securing client-to-server communication is not covered in this chapter (e.g. Bolt, HTTPS, Backup).

Introduction

The security solution for cluster communication is based on standard SSL/TLS technology (referred to jointly as SSL). Encryption is just one aspect of security, the other cornerstones are authentication and integrity. A secure solution is based on a key infrastructure which is deployed together with a requirement of authentication.

The SSL support in the platform is documented in detail in SSL framework. This section covers the specifics as they relate to securing a cluster.

Under SSL, an endpoint can authenticate itself using certificates managed by a Public Key Infrastructure (PKI).

Important:

The deployment of a secure key management infrastructure is beyond the scope of this manual, and should be entrusted to experienced security professionals. The example deployment illustrated below is for reference purposes only.

Example deployment

Generate and install cryptographic objects

The generation of cryptographic objects is for the most part outside the scope of this manual. It generally requires having a PKI with a Certificate Authority (CA) within the organization and they should be able to advise here. Note that the information in this manual relating to the PKI is mainly for illustrative purposes.

Tip:

If setting up intra-cluster encryption as part of a cluster configuration, ensure that the certificates used on the cluster endpoint support server and client usage. This is because when connecting between the Neo4j servers for clustering, each server uses its own certificate to authenticate as a client on the connection to another server.

This could be verified from within the certificate details:

```
openssl x509 -in public.crt -noout -text
```

You should see that the X509v3 Extended Key Usage section shows both the usages listed:

```
X509v3 Extended Key Usage:
    TLS Web Server Authentication, TLS Web Client Authentication
```

When the certificates and private keys are obtained they can be installed on each of the servers. Each server has a certificate of its own, signed by a CA, and the corresponding private key. The certificate of the CA is installed into the `trusted` directory, and any certificate signed by the CA is thus trusted. This means that the server now has the capability of establishing trust with other servers.

Caution:

Be sure to exercise caution when using CA certificates in the `trusted` directory, as any certificates signed by that CA are then trusted to join the cluster. Never use a public CA or your internal root CA to sign certificates for your cluster. Instead, use an intermediate certificate or a CA certificate which originates from and is controlled by your organization, and is only used for that specific cluster.

In this example a mutual authentication setup is deployed, which means that both ends of a channel have to authenticate. To enable mutual authentication the SSL policy must have `client_auth` set to `REQUIRE` (which is the default). Servers are by default required to authenticate themselves, so there is no corresponding server setting.

If the certificate for a particular server is compromised, it is possible to revoke it by installing a Certificate Revocation List (CRL) in the `revoked` directory. It is also possible to redeploy using a new CA. For contingency purposes, it is advised to have a separate intermediate CA specifically for the cluster which can be substituted in its entirety should it ever become necessary. This approach would be much easier than having to handle revocations and ensuring their propagation.

Example 64. Generate and install cryptographic objects

In this example, assume that the private key and certificate file are named *private.key* and *public.crt*, respectively. The policy configuration for the key and certificate names/locations can be overridden if different names are desired. For this server, use the default configuration, create the appropriate directory structure, and install the certificate:

```
$NEO4J_HOME> mkdir certificates/cluster
$NEO4J_HOME> mkdir certificates/cluster/trusted
$NEO4J_HOME> mkdir certificates/cluster/revoked

$NEO4J_HOME> cp $some-dir/private.key certificates/cluster
$NEO4J_HOME> cp $some-dir/public.crt certificates/cluster
```

Configure the cluster SSL policy

By default, cluster communication is unencrypted. To configure a cluster to encrypt its intra-cluster communication, set `dbms.ssl.policy.cluster.enabled` to `true`.

An SSL policy utilizes the installed cryptographic objects and additionally allows parameters to be configured. Use the parameters in one of the following configurations:

The following example assumes that an SSL policy is created and configured as per the example deployment and uses the default TLSv1.2.

Add the following content to the *neo4j.conf* file:

```
dbms.ssl.policy.cluster.enabled=true
dbms.ssl.policy.cluster.tls_versions=TLSv1.2 \
```

```
dbms.ssl.policy.cluster.ciphers=TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 \
dbms.ssl.policy.cluster.client_auth=REQUIRE
```

With control of the entire cluster, the default TLS standard can be enforced without any concern for backwards compatibility. It has no known security vulnerabilities and uses the most modern algorithms for key exchanges, etc.

A particular single strong cipher can be enforced and thus remove any doubt about which cipher gets negotiated and chosen. The selected cipher offers Perfect Forward Secrecy (PFS) and uses the Advanced Encryption Standard (AES) for symmetric encryption. AES has great support for hardware acceleration and thus allows performance to be generally negligibly affected.

Setting the cluster client authentication to `REQUIRE` enables mutual authentication, meaning both ends of a channel must authenticate.

The following example assumes that an SSL policy is created and configured as per the example deployment and uses both TLSv1.2 and TLSv1.3.

Add the following content to the *neo4j.conf* file:

```
dbms.ssl.policy.cluster.enabled=true
dbms.ssl.policy.cluster.tls_versions=TLSv1.3,TLSv1.2 \
dbms.ssl.policy.cluster.ciphers=TLS_AES_256_GCM_SHA384,TLS_AES_128_GCM_SHA256,TLS_RSA_WITH_AES_2
56_CBC_SHA,TLS_DHE_DSS_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA \
dbms.ssl.policy.cluster.client_auth=REQUIRE \
```

With control of the entire cluster, the default TLS standard can be enforced without any concern for backwards compatibility. It has no known security vulnerabilities and uses the most modern algorithms for key exchanges, etc.

If you want to specify ciphers for both supported TLS versions, you must specify ciphers for each TLS version not to get more ciphers than expected. The selected ciphers offer Perfect Forward Secrecy (PFS) and use the Advanced Encryption Standard (AES) for symmetric encryption. AES has great support for hardware acceleration and thus allows performance to be generally negligibly affected.

Setting the cluster client authentication to `REQUIRE` enables mutual authentication, meaning both ends of a channel must authenticate. They have no known security vulnerabilities and use the most modern algorithms for key exchanges, etc.

Any user data communicated between servers is now secured. Note that a server that is not correctly setup is not able to communicate with the others.

The policy must be configured on every server with the same settings. The actual cryptographic objects installed are mostly different since they do not share the same private keys and corresponding certificates. However, the trusted CA certificate is shared.

Validate the secure operation of the cluster

To make sure that everything is secured as intended, it makes sense to validate using external tooling such as, for example, the open source assessment tools `nmap` or `OpenSSL`.

Example 65. Validate the secure operation of the cluster

This example uses the `nmap` tool to validate the secure operation of the cluster. A simple test to perform is a cipher enumeration using the following command:

```
nmap --script ssl-enum-ciphers -p <port> <hostname>
```

The hostname and port have to be adjusted according to the example configuration. This can prove that TLS is in fact enabled and that only the intended cipher suites are enabled. All servers and all applicable ports should be tested.

For testing purposes, it is also possible to utilize a separate testing instance of Neo4j which, for example, has an untrusted certificate in place. The expected result of this is that the test server is not able to participate in replication of user data. The debug logs generally indicate an issue by printing an SSL or certificate-related exception.

Managing servers in a cluster

Enterprise Edition

In a clustered environment, server management is completely separate from database management. This section describes how to work with servers in a cluster: adding and removing them, as well as altering their metadata.

For details on server management command syntax, see Server management command syntax.

Server lifecycle

A server can exist in six different states within the DBMS:

- Free
- Enabled
- Cordoned
- Deallocating
- Deallocated Introduced in 5.15
- Dropped

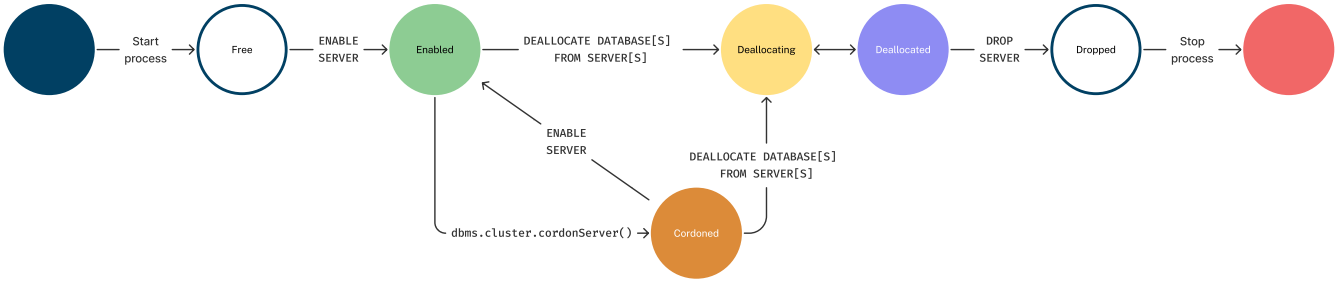


Figure 10. Server's lifecycle

Free state

When a server is discovered by the discovery service (see Setting up a cluster → Cluster server discovery for more information), it is created in the *free* state. Servers in this state have a unique automatically generated ID, but are otherwise unconfigured. These free servers are not yet part of the cluster and cannot be allocated to host any databases.

> **Note:**
>
> When first discovered, a server's name defaults to the value of its generated server ID.

Enabled state

When you deploy a cluster for the first time, at least the minimum number of servers included in the initial deployment are automatically enabled. For details, refer to the example on how to Configure a cluster with three servers.

If you add a new server after the cluster is already running, the server is added in the `Free` state. A server in the `Free` state needs to be explicitly enabled in order to be considered an active member of the cluster.

To transition a server into the *enabled* state, use the command `ENABLE SERVER`:

Syntax

```
ENABLE SERVER 'serverId' [OPTIONS "{" option: value[,...] "}"]
```

The server's initial name is its ID. If the server is already enabled and the command is executed with the same options, nothing is changed.

The possible options when enabling a server are:

Option	Allowed values	Description
modeConstraint	`PRIMARY`, `SECONDARY`, `NONE`	Databases may only be hosted on the server in the mode specified by the constraint. `None` means there is no constraint and any mode is allowed.
allowedDatabases	list of database names, e.g. `["db1", "db2"]`	Only databases matching the specified names may be hosted on the server. This may not be specified in combination with `deniedDatabases`.

Option	Allowed values	Description
deniedDatabases	list of database names, e.g. `["db1", "db2"]`	Only databases **not** matching the specified names may be hosted on the server. This may not be specified in combination with `allowedDatabases`.
tags	list of server tags, e.g. `["tag1", "tag2"]`	List of server tags used for load balancing and routing policies. Introduced in 5.6

Note:

When a server is enabled, if no `OPTIONS` are provided, the default server's values are taken from the settings:

- `initial.server.mode_constraint`
- `initial.server.allowed_databases`
- `initial.server.denied_databases`
- and/or `initial.server.tags`

All these settings are only effective when enabling the relevant server.

Another way to enable servers is to configure the DBMS automatically enable free servers. You can do this in one of two ways:

- By setting `initial.dbms.automatically_enable_free_servers` to `true` before starting the deployment for the first time.
- By running the following procedure after deployment:

```
CALL dbms.cluster.setAutomaticallyEnableFreeServers(true);
```

Once enabled, the server may be allocated databases to host.

If you need to change some of the server's values, use the `ALTER SERVER` command.

Cordoned state

A server in a *cordoned* state cannot be assigned to host any newly created databases; however, the server retains the databases it already hosts.

This state is primarily used for error handling.

To transition a server from the *enabled* to the *cordoned* state, run the `dbms.cluster.cordonServer()` procedure. Keep in mind that when decreasing the number of allocations of a database, allocations on cordoned servers are removed first.

A server in the *cordoned* state can be transitioned to *deallocating* state or back to *enabled*. To re-enable a server, run the `ENABLE SERVER` command.

You can also use the `dbms.cluster.uncordonServer()` procedure. However, note that the procedure is deprecated in Neo4j 5.23.

Deallocating state

A *deallocating* state means a server can no longer host databases. It may be that the server is no longer needed and you want to remove it from the cluster.

To transition servers to the *deallocating* state, run the following command:

```
DEALLOCATE DATABASE[S] FROM SERVER[S] 'name'[, ...]
```

The command reallocates all user databases hosted by the server to other servers in the cluster. This state is **irreversible**. Once a server is in a deallocating state, it subsequently cannot have databases allocated to it.

However, you can deallocate databases from a server in a **reversible manner** by running one of the following procedures:

- `dbms.cluster.deallocateDatabaseFromServer("server-name", "database-name")`
- `dbms.cluster.deallocateDatabaseFromServers(["server-name1", "server-name2"\`, "database-name")]
- `dbms.cluster.deallocateNumberOfDatabases("server-name", number)`

For details, see Managing databases in a cluster → Deallocate databases.

Deallocated state

Introduced in 5.15

The *deallocated* state means that a server no longer hosts any databases besides the `system` database and can be removed from the cluster. Additionally, deallocated servers cannot have any further databases allocated to them.

Note that there is a known situation in which a previously deallocated offline server can transiently show as deallocating when restarting, it will, however, eventually return to the deallocated state without intervention.

Dropped state

The *dropped* state means a server has been removed from the cluster but remains visible to the other cluster members.

Once a server is in the *deallocated* state and is only hosting the `system` database, it is safe to remove it. Use the command `DROP SERVER 'server name'` to remove the server from the cluster.

However, as long as the server's Neo4j process is running, it is still visible to the other cluster members in the *dropped* state. When the Neo4j process is stopped, the server finally disappears. Once dropped, a

server cannot rejoin a cluster.

> **Note:**
>
> To allow the same physical hardware to rejoin the cluster, reset the Neo4j installation by either:
>
> - Reinstalling Neo4j.
> - Running the `neo4j-admin server unbind` command.
>
> This process ensures a new server ID is generated when the server starts again.

Listing servers

The Cypher command `SHOW SERVERS` displays all current servers running in the cluster, including servers yet to be enabled (i.e. servers in the *free* state) in the DBMS as well as dropped servers.

```
neo4j@system> SHOW SERVERS;
+-----------------------------------------------------------------------------------------------------------
---------+
| name                                 | address          | state     | health      | hosting
|
+-----------------------------------------------------------------------------------------------------------
---------+
| "135ad202-5405-4d3c-9822-df39f59b823c" | "localhost:7690" | "Dropped" | "Available" | ["system"]
|
| "25a7efc7-d063-44b8-bdee-f23357f89f01" | "localhost:7689" | "Enabled" | "Available" | ["system", "foo",
"neo4j"] |
| "42a97acc-acf6-40c0-aff2-3993e90db1ff" | "localhost:7691" | "Free"    | "Available" | ["system"]
|
| "782f0ee2-5474-4250-b905-4cd8b8f586ba" | "localhost:7688" | "Enabled" | "Available" | ["system", "foo",
"neo4j"] |
| "8512c9b9-d9e8-48e6-b037-b15b0004ca18" | "localhost:7687" | "Enabled" | "Available" | ["system", "foo",
"neo4j"] |
+-----------------------------------------------------------------------------------------------------------
---------+
```

To display all available information about the servers in the cluster, use `SHOW SERVERS YIELD *`:

```
neo4j@system> SHOW SERVERS YIELD *;
+-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+
| serverId                             | name                                 | address          |
httpAddress      | httpsAddress | state          | health      | hosting                    |
requestedHosting           | tags | allowedDatabases | deniedDatabases | modeConstraint | version
|
+-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+
| "135ad202-5405-4d3c-9822-df39f59b823c" | "135ad202-5405-4d3c-9822-df39f59b823c" | "localhost:7690" |
"localhost:7477" | NULL         | "Deallocating" | "Available" | ["system"]                 | ["system"]
| []   | []               | []               | "NONE"          | "5.0.0-drop09.0" |
| "25a7efc7-d063-44b8-bdee-f23357f89f01" | "25a7efc7-d063-44b8-bdee-f23357f89f01" | "localhost:7689" |
"localhost:7476" | NULL         | "Enabled"      | "Available" | ["system", "foo", "neo4j"] | ["system",
"foo", "neo4j"] | []   | []               | []              | "NONE"          | "5.0.0-drop09.0" |
| "42a97acc-acf6-40c0-aff2-3993e90db1ff" | "42a97acc-acf6-40c0-aff2-3993e90db1ff" | "localhost:7691" |
"localhost:7478" | NULL         | "Free"         | "Available" | ["system"]                 | []
| []   | []               | []               | "NONE"          | "5.0.0-drop09.0" |
| "782f0ee2-5474-4250-b905-4cd8b8f586ba" | "782f0ee2-5474-4250-b905-4cd8b8f586ba" | "localhost:7688" |
"localhost:7475" | NULL         | "Enabled"      | "Available" | ["system", "foo", "neo4j"] | ["system",
"foo", "neo4j"] | []   | []               | []              | "NONE"          | "5.0.0-drop09.0" |
| "8512c9b9-d9e8-48e6-b037-b15b0004ca18" | "8512c9b9-d9e8-48e6-b037-b15b0004ca18" | "localhost:7687" |
```

```
"localhost:7474" | NULL         | "Enabled"      | "Available" | ["system", "foo", "neo4j"] | ["system",
"foo", "neo4j"] | []   | []              | []             | "NONE"         | "5.0.0-drop09.0" |
+-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+
```

The table of results shows information about the servers.

Table 395. The `SHOW SERVERS` output modes

Column	Description	Type	Default output	Full output
name	Name of the server.	`STRING`	✔	✔
serverId	ID of the server.	`STRING`		✔
address	Bolt address of the server (if enabled).	`STRING`	✔	✔
httpAddress	HTTP address of the server (if enabled).	`STRING`		✔
httpsAddress	HTTPS address of the server (if enabled).	`STRING`		✔
state	Information of the state of the server: `Free`, `Enabled`, `Cordoned`, `Deallocating`, `Deallocated`, or `Dropped`.	`STRING`	✔	✔
health	The availability of the server: `Available` or `Unavailable`.	`STRING`	✔	✔
hosting	A list of databases currently hosted on the server.	`LIST<STRING>`	✔	✔
requestedHosting	A list of databases that should be hosted on the server. Composite databases do not currently appear in this list, though they do appear in `hosting` for all servers.	`LIST<STRING>`		✔
tags	Tags are user provided strings that can be used for load balancing and routing policies.	`LIST<STRING>`		✔
allowedDatabases	A list of databases allowed to be hosted on the server.	`LIST<STRING>`		✔
deniedDatabases	A list of databases not allowed to be hosted on the server.	`LIST<STRING>`		✔
modeConstraint	Constraint for the allocator to allocate only databases in this mode on the server.	`STRING`		✔
version	Neo4j version the server is running.	`STRING`		✔

Adding a server to the cluster

To add a server to a running cluster, configure it to discover other existing cluster members.

In the example below, you set `dbms.cluster.discovery.resolver_type=LIST` and use the discovery service v2 introduced in 5.23.

To use the discovery service v2:

- You have to set the configuration of `dbms.cluster.discovery.version` to `V2_ONLY`.
- Instead of `dbms.cluster.discovery.endpoints`, use `dbms.cluster.discovery.v2.endpoints`.
- The port must be `6000`.

For details, see Cluster server discovery.

1. Prepare the *neo4j.conf* file for the new `server.07`.

   neo4j.conf on server07.example.com:

   ```
   server.default_listen_address=0.0.0.0
   server.default_advertised_address=server07.example.com
   dbms.cluster.discovery.resolver_type=LIST
   dbms.cluster.discovery.v2.endpoints=server01.example.com:6000,server02.example.com:6000,server03.examp
   le.com:6000,server04.example.com:6000,server05.example.com:6000,server06.example.com:6000
   dbms.cluster.discovery.version=V2_ONLY
   server.cluster.system_database_mode=PRIMARY
   ```

   Additionally, you can configure a set of initial settings to provide values for the server's options. Those settings will be used as the default input for the `ENABLE SERVER` command. For details, see Enabled state.
   Besides, you can also set the `initial.dbms.automatically_enable_free_servers` to `true`, allowing the DBMS automatically enable a free server.

   Configure the `system` database mode according to the server's purpose:

   - Set the mode to `PRIMARY` if the server is expected to be long-term and handle high-volume workloads.
   - Set the mode to `SECONDARY` if the server is intended to support analytics or read-heavy workloads.
2. Once the new server is configured to discover the cluster's members, start the Neo4j process.
3. Once started, the new server appears in the output of `SHOW SERVERS` with the `Free` state.
4. Copy the server's ID from the `SHOW SERVERS` output and enable the server.

   The `ENABLE SERVER` command can take several options and it will override the initial settings if they are conflicting:

   ```
   neo4j@system> ENABLE SERVER '25a7efc7-d063-44b8-bdee-f23357f89f01' OPTIONS
                 {modeConstraint:'PRIMARY', allowedDatabases:['foo'], tags:['eu','eu-west']};
   ```

   - `modeConstraint` is used to control whether a server can only host standard databases in primary mode, only in secondary mode, or can host databases in either mode.
   - `allowedDatabases` and `deniedDatabases` are collections of database names that filter which databases may be hosted on a server. The `allowedDatabases` and `deniedDatabases` are mutually

exclusive and if both are specified, an error is returned.

- Server `tags` are used when configuring load balancing and replication policies. They cannot contain duplicates, so `tags:['eu', 'eu']` will return an error. Server tags also cannot contain commas. When altering server tags via Cypher, the encoding is done via UTF-8.

> **Tip:**
>
> Neo4j *.conf* files use **Latin1** for their encoding by default. Therefore, for server tags that need a larger character set (e.g. Chinese or Arabic), it is recommended to use Cypher to alter server tags.
>
> From Neo4j 5.8.0 , *.conf* files can use **UTF-8** by setting the environment variable `NEO4J_CONFIG_FILE_CHARSET=utf8`. This allows setting server tags with the larger character set via the config.

Hosting databases on added servers

Once enabled, a server does not automatically host databases unless:

- New databases are created.
- Existing database topologies are altered to request more hosts.
- Another server is transitioned to the *deallocating* state.
- You explicitly rebalance the databases across the cluster, possibly reallocating them to the newly added server(s).

Removing a server from the cluster

Removing a server from the cluster requires two steps: deallocating, then dropping.

Deallocating databases from a server

In preparation for removing a server from the cluster, set it to not host any databases with the `DEALLOCATE DATABASES FROM SERVER 'name'` command.

See deallocating state for more information.

Either the server ID or its name can be used with the `DEALLOCATE DATABASES` command:

```
neo4j@system> DRYRUN DEALLOCATE DATABASES FROM SERVER '135ad202-5405-4d3c-9822-df39f59b823c';
```

When deallocating databases from servers, it is important to be mindful of the topology for each database to ensure that there are sufficient servers left in the cluster to satisfy the topologies of each database. Attempting to deallocate database(s) from a server that would result in less available servers than required fails with an error and no changes are made.

For example, if the cluster contains five servers and a database `foo` has a topology requiring three primary

copies and two secondary copies, then it is **not** possible to deallocate any of the original five servers, without first adding and enabling new unconstrained servers, or altering the desired topology of `foo` to require fewer servers overall.

The command can be used with `DRYRUN` to get a view of how the databases would be moved from the deallocated server(s).

```
neo4j@system> DRYRUN DEALLOCATE DATABASES FROM SERVER '135ad202-5405-4d3c-9822-df39f59b823c';
+-------------------------------------------------------------------------------------------------------------------
----------------------------------+
| database | fromServerName | fromServerId                           | toServerName | toServerId
| mode        |
+-------------------------------------------------------------------------------------------------------------------
----------------------------------+
| "db1"    | "server-3"     | "135ad202-5405-4d3c-9822-df39f59b823c" | "server-5"   | "00000003-b30a-434e-
b9bf-1a5c8009773a" | "secondary" |
+-------------------------------------------------------------------------------------------------------------------
----------------------------------+
```

Note:

Deallocation is currently prevented in the following situations:

- If a database, the server is hosting, is offline.
- If the server is hosting a database with an allocation of one primary.
- If a quorum of servers, hosting the database in primary mode, are cordoned.

Once the command has been executed, the server changes state to `Deallocating` and it cannot readily be enabled again. See dropped state for more information.

Dropping a server

Once `DEALLOCATE DATABASES` is executed against a server, its databases begin being moved. It is important not to attempt the next step before `SHOW SERVERS` reports that the deallocating server is in the deallocated state.

For example, do not drop the server `135ad202-5405-4d3c-9822-df39f59b823c` given the following output:

```
SHOW SERVERS;
+-------------------------------------------------------------------------------------------------------------------
---------+
| name                                   | address          | state          | health      | hosting
|
+-------------------------------------------------------------------------------------------------------------------
---------+
| "135ad202-5405-4d3c-9822-df39f59b823c" | "localhost:7690" | "Deallocating" | "Available" | ["system",
"foo"]     |
+-------------------------------------------------------------------------------------------------------------------
---------+
```

The deallocation process may take some time, as `foo` must be successfully copied and started on a new server before it is stopped on `135ad202-5405-4d3c-9822-df39f59b823c` in order to preserve the availability and fault tolerance of `foo`.

Once `SHOW SERVERS` reflects that the server is `Deallocated` and thus no longer hosts `foo`, the server may be

dropped. Either the server ID or its name can be used:

```
neo4j@system> DROP SERVER '135ad202-5405-4d3c-9822-df39f59b823c';
```

Once this command has been executed successfully, the Neo4j process on the server in question may be stopped.

Controlling a server's metadata

Altering server options

A running server can have its options modified using the `ALTER SERVER 'name' SET OPTIONS { option: value }` command. Either the ID or the name of the server can be used.

Syntax

```
ALTER SERVER 'name' SET OPTIONS "{" option: value[,...] "}"
```

For example, to prevent a server from hosting databases in `PRIMARY`, execute the following:

```
neo4j@system> ALTER SERVER '25a7efc7-d063-44b8-bdee-f23357f89f01' SET OPTIONS {modeConstraint:
'SECONDARY'};
```

Altering servers may cause databases to be moved, and should be performed with care. For example, if the server `25a7efc7-d063-44b8-bdee-f23357f89f01` hosts database `foo` in primary mode when the above command is executed, then another server must begin hosting `foo` in primary mode.

Likewise, if `ALTER SERVER '25a7efc7-d063-44b8-bdee-f23357f89f01' SET OPTIONS {allowedDatabases:['bar','baz']};` is executed, then `foo` is forced to move.

The possible options when altering a server are:

Option	Allowed values	Description
modeConstraint	`PRIMARY`, `SECONDARY`, `NONE`	Databases may only be hosted on the server in the mode specified by the constraint. `NONE` means there is no constraint and any mode is allowed.
allowedDatabases	list of database names, e.g. `["db1", "db2"]`	Only databases matching the specified names may be hosted on the server. This may not be specified in combination with `deniedDatabases`.
deniedDatabases	list of database names, e.g. `["db1", "db2"]`	Only databases **not** matching the specified names may be hosted on the server. This may not be specified in combination with `allowedDatabases`.

Option	Allowed values	Description
tags	list of server tags, e.g. `["tag1", "tag2"]`	List of server tags used for load balancing and routing policies. Introduced in 5.6

> **Note:**
>
> `allowedDatabases` and `deniedDatabases` do not affect composite databases; they are always available everywhere.

As with the `DEALLOCATE DATABASES FROM SERVER …` command, if the alteration of a server's options renders it impossible for the cluster to satisfy one or more of the databases' topologies, then the command fails and no changes are made.

> **Important:**
>
> Input provided to `SET OPTIONS {…}` replaces **all** existing options, rather than being combined with them.
>
> Any previously set values must be specified every time you run the `ALTER SERVER` command; otherwise they will be overwritten to the unset values.
>
> For instance, you run two statements one after the other:
>
> ```
> ALTER SERVER '25a7efc7-d063-44b8-bdee-f23357f89f01' SET OPTIONS {modeConstraint:
> 'SECONDARY'};
> ```
>
> ```
> ALTER SERVER '25a7efc7-d063-44b8-bdee-f23357f89f01' SET OPTIONS {allowedDatabases:['foo']};
> ```
>
> The execution of the second `ALTER SERVER` removes the mode constraint `SECONDARY`, replacing it with `NONE`.
>
> If you want to keep both values `modeConstraint:'SECONDARY'` and `allowedDatabases:['foo']`, you have to explicitly set them in the options for the `ALTER SERVER` command:
>
> ```
> ALTER SERVER '25a7efc7-d063-44b8-bdee-f23357f89f01' SET OPTIONS {modeConstraint:
> 'SECONDARY', allowedDatabases:['foo']};
> ```
>
> Always check the current configuration with `SHOW SERVERS YIELD *` and reapply unchanged options when using `ALTER SERVER`.

Renaming a server

Syntax

```
RENAME SERVER 'name' TO 'newName'
```

When first discovered, a server's name defaults to the value of its generated server ID. However, as long as the server is enabled, this can be changed later using the following command:

```
neo4j@system> RENAME SERVER '25a7efc7-d063-44b8-bdee-f23357f89f01' TO 'eu-server-4';
```

This only affects the name of the server; the ID of the server remains fixed as `25a7efc7-d063-44b8-bdee-f23357f89f01`. Keep in mind that the name of the server must be unique among existing servers.

Error handling

Occasionally, servers in a cluster may suffer issues such as network partitions or process crashes. The easiest way to observe these server failures is by executing `SHOW SERVERS` and checking for `'Unavailable'` in the `health` column.

> **Note:**
>
> An `Available` health status does not indicate that a server is functioning perfectly, only that other servers in the cluster are able to make contact with it.
>
> For more in depth monitoring of cluster and server health, see Monitor servers.

If the issue with the `Unavailable` server proves permanent, then the server should be removed. However, if the issue is temporary then it likely is not desirable to remove these servers entirely as this causes all their hosted databases to be moved. Instead it is preferable to prevent those servers from being allocated any new databases to host, either as a result of databases being created or moved.

This is known as cordoning the server in question, and can be achieved by executing the following procedure against the `system` database:

```
neo4j@system> CALL dbms.cluster.cordonServer('25a7efc7-d063-44b8-bdee-f23357f89f01');
```

`SHOW SERVERS` should then reflect that the server in question is now in *cordoned* state.

Once the issue with the server has been resolved, the server can be returned to its previous *enabled* state as follows:

```
neo4j@system> ENABLE SERVER '25a7efc7-d063-44b8-bdee-f23357f89f01';
```

> **Note:**
>
> An unavailable server which has not been cordoned may still be allocated to host new databases. When the server recovers it observes that it is due to host these databases and begin catching up from some other available servers (if one exists). However, in the meantime those databases have reduced fault tolerance or, worse, reduced availability. See Disaster Recovery for more details.

Managing databases in a cluster

When creating a database or altering it after creation, you can select the number of `primary` and `secondary` allocations for it, i.e., define a topology for a database. To specify the initial database topology, use the Cypher command `CREATE DATABASE`. To change the topology once the database is created, run the `ALTER DATABASE` command. Note that the database role must be aligned with the server's `modeConstraint`, which value can be `PRIMARY`, or `SECONDARY`, or `NONE`.

If a database is no longer needed, the command `DROP DATABASE` deletes the database from the cluster.

The `system` database in a cluster

The `system` database contains metadata on the DBMS and security configuration. When connected to this database you can only perform a specific set of administrative tasks, such as managing databases, servers, and access control.

In the clustered environment, the `system` database can be in a primary or secondary mode. To configure its mode, use the `server.cluster.system_database_mode` setting in the *neo4j.conf* file. This has to be done before starting the Neo4j process. For guidelines on how to choose the mode for the `system` database, see Introduction → Primaries and secondaries for the `system` database.

Deploying a cluster for the first time, consider configuring the `dbms.cluster.minimum_initial_system_primaries_count`. This setting defaults to `3`. See Deploy a basic cluster for details.

For information on how to recover the `system` database, refer to the Disaster recovery guide.

Create databases

The command to create a standard database in a cluster is not significantly different from the command to create a database in a non-clustered environment. See Create, start, and stop databases for more information on database management in a non-clustered environments.

The difference in a clustered environment is that the database topology can be specified, i.e. how many primaries and secondaries are desired for the database. For more details on how many primaries and secondaries it is recommended to have, see Introduction: Neo4j clustering architecture.

To create a database `foo` with three primaries and two secondaries, run the following Cypher command:

```
CREATE DATABASE foo TOPOLOGY 3 PRIMARIES 2 SECONDARIES
```

Alternatively, starting from 5.26, you can use parameters to provide the number of primaries and secondaries:

Parameters

```
{
  "dbname": "foo",
  "primary": 3,
  "secondary": 2
```

```
}
```

Query

```
CREATE DATABASE $dbname TOPOLOGY $primary PRIMARIES $secondary SECONDARIES
```

The command can only be executed successfully if the cluster's servers are able to satisfy the specified topology. If they are not, the command results in an error. For example, if only four servers exist, the command fails with an error. Also, it fails if two servers are set up with a mode constraint of primary, and three servers are set up with a mode constraint of secondary.

If `TOPOLOGY` is not specified, the database is created according to `initial.dbms.default_primaries_count` and `initial.dbms.default_secondaries_count` specified in *neo4j.conf*. After cluster startup, you can overwrite these values using the `dbms.setDefaultAllocationNumbers()` procedure.

A `CREATE DATABASE` command allocates the database, therefore there is no requirement to run `REALLOCATE DATABASES` (described in Hosting databases on added servers).

However, over time, or after several `CREATE DATABASE` commands have been issued, the distribution of databases can become unbalanced. At this point you can run `REALLOCATE DATABASES` to make the cluster re-balance databases across all servers that are part of the cluster.

Modify database topology and read/write access

To alter the topology of or read/write access to a database after its creation, use the command `ALTER DATABASE`.

For information about the command syntax, see Alter databases.

Alter database topology

To change the topology of the database `foo` from the previous example, run the following command:

```
ALTER DATABASE foo SET TOPOLOGY 2 PRIMARIES 1 SECONDARY
```

Alternatively, starting from 5.26, you can use parameters to provide the number of primaries and secondaries:

Parameters

```
{
  "dbname": "foo",
  "primary": 2,
  "secondary": 1
}
```

Query

```
ALTER DATABASE $dbname SET TOPOLOGY $primary PRIMARIES $secondary SECONDARIES
```

Like the `CREATE DATABASE` command, the `ALTER DATABASE` command results in an error if the cluster does

not contain sufficient servers to satisfy the requested topology.

Additionally, `ALTER DATABASE` is optionally idempotent and also results in an error if the database does not exist. It is possible to append the command with `IF EXISTS` to make sure that no error is returned if the database does not exist.

When there is more than one possible permutation of the specified topology, Neo4j uses an allocator to decide how to spread the database across the cluster. Note, like `CREATE DATABASE`, the `ALTER DATABASE` command allocates the database and there is no requirement to execute `REALLOCATE DATABASES` unless there is a desire to re-balance databases across all servers that are part of the cluster.

This normally happens when the cluster is configured with more servers than the sum of the number of primaries and secondaries for any one database.

It is not possible to automatically transition to a topology with a single primary host. Attempting to do so results in an error.

However, it is possible to *manually* do this transition:

1. First, back up the database. See Backup and restore for more information.
2. Once the database is backed up, drop the database. See Delete databases for more information.
3. The last step is to either seed a cluster from the backup with the new topology, or to restore the backup on a single server. See Seed a cluster further on for information on seeding.

Also, it is possible to automatically transition *from* a topology with a single primary host to multiple primary hosts. Keep in mind that during such a transition, the database will be unavailable for a short period of time.

If the `ALTER DATABASE` command decreases the number of allocations of a database, allocations on cordoned servers are removed first.

Query

```
ALTER DATABASE nonExisting IF EXISTS SET TOPOLOGY 1 PRIMARY 0 SECONDARY
```

0 rows

Alter database access

To alter the access to the database `foo`, the syntax looks like this:

```
ALTER DATABASE foo SET ACCESS {READ ONLY | READ WRITE}
```

By default, a newly created database has both read and write access.

Deallocate databases

Introduced in 5.23

To relieve the load of a specific server(s), you can use one of the following procedures to deallocate databases, causing the databases to be removed from the server(s) under pressure:

- `dbms.cluster.deallocateDatabaseFromServer("server-name", "database-name")`
- `dbms.cluster.deallocateDatabaseFromServers(["server-name1", "server-name2"\, "database-name")]`
- `dbms.cluster.deallocateNumberOfDatabases("server-name", number)`

> **Note:**
>
> You must have the `SERVER MANAGEMENT` privilege to execute these procedures.

For example, `server01` hosts two small databases, `foo` and `bar`, and one very large database `baz`, while other servers contain fewer or smaller databases, and `server01` is under pressure.

You can use one of the following approaches to deallocate `baz` from `server01` or to deallocate a number of databases from `server01`:

Deallocating a database from a server

```
// With dry run
neo4j@system> CALL dbms.cluster.deallocateDatabaseFromServer("server01", "baz", true);

// Without dry run
neo4j@system> CALL dbms.cluster.deallocateDatabaseFromServer("server01", "baz");
```

Deallocating a database from multiple servers

```
// With dry run
neo4j@system> CALL dbms.cluster.deallocateDatabaseFromServers(["server01", "server02"], "baz", true);

// Without dry run
neo4j@system> CALL dbms.cluster.deallocateDatabaseFromServers(["server01", "server02"], "baz");
```

Deallocating three databases from a server

```
// With dry run
neo4j@system> CALL dbms.cluster.deallocateNumberOfDatabases("server01", 3, true);

// Without dry run
neo4j@system> CALL dbms.cluster.deallocateNumberOfDatabases("server01", 3);
```

Reallocate databases

To rebalance all database allocations across the cluster, for example, because you added new servers, use either procedures or Cypher commands to reallocate databases onto the new servers.

Reallocate databases using a procedure

Introduced in 5.23

You can use the procedure `dbms.cluster.reallocateDatabase` to rebalance a specific database across the

cluster, or `dbms.cluster.reallocateNumberOfDatabases` to rebalance a number of database allocations across the cluster and relieve overloaded servers. Note that if the cluster is already balanced, no reallocations will happen when running these procedures. These procedures do not require a server name and can be executed with or without a dry run.

> **Note:**
>
> You must have the `SERVER MANAGEMENT` privilege to execute these procedures.

For example, you add three new servers and want to move a very large database, `baz`, from all the servers containing it to the new servers.

Reallocate one database to new servers

```
// With dry run
neo4j@system> CALL dbms.cluster.reallocateDatabase("baz", true);

// Without dry run
neo4j@system> CALL dbms.cluster.reallocateDatabase("baz");
```

Reallocating a number of databases to new servers

```
// With dry run
neo4j@system> CALL dbms.cluster.reallocateNumberOfDatabases(3, true);

// Without dry run
neo4j@system> CALL dbms.cluster.reallocateNumberOfDatabases(3);
```

Reallocate databases using a Cypher command

You can use the Cypher command `REALLOCATE DATABASES` to rebalance all database allocations across the cluster and relieve overloaded servers. This command can also be used with `DRYRUN` to preview the new allocation of databases.

> **Caution:**
>
> `REALLOCATE DATABASES` on a large cluster with many databases has the potential to move a lot of allocations at once, which might stress the cluster. Consider starting with more limited reallocations, such as `dbms.cluster.reallocateNumberOfDatabases` with a small number, and let the databases complete their reallocation before calling it again, until no more reallocations are necessary.

> **Note:**
>
> `DRYRUN` is available from Neo4j 5.2 and later.

```
neo4j@neo4j> DRYRUN REALLOCATE DATABASES;
+-------------------------------------------------------------------------------------------------------------
------------------------------+
| database | fromServerName | fromServerId                          | toServerName | toServerId
| mode      |
+-------------------------------------------------------------------------------------------------------------
```

```
------------------------------+
| "bar"    | "server-1"     | "00000000-27e1-402b-be79-d28047a9418a" | "server-5"   | "00000003-b76c-483f-
b2ca-935a1a28f3db" | "primary" |
| "bar"    | "server-3"     | "00000001-7a21-4780-bb83-cee4726cb318" | "server-4"   | "00000002-14b5-4d4c-
ae62-56845797661a" | "primary" |
+-----------------------------------------------------------------------------------------------------------
------------------------------+
```

Seed a cluster

There are two different ways to seed a cluster with data:

- The first option is to use a *designated seeder*, where a designated server is used to create a backed-up database on other servers in the cluster.
- The other option is to seed the cluster from a URI, where all servers to host the database are seeded with an identical seed from an external source specified by that URI. For more details, see Create a database from a URI.

Keep in mind that using a designated seeder can be problematic in some situations as it is not known in advance how a database is going to be allocated to the servers in a cluster. Also, this method relies on the seed already existing on one of the servers.

Designated seeder

In order to designate a server in the cluster as a seeder, a database backup is transferred to that server using the `neo4j-admin database restore` command. Subsequently, that server is used as the source for other cluster members to copy the backed-up database from.

This example creates a user database called `foo`, hosted on three servers in primary mode. The `foo` database **should not** previously exist on any of the servers in the cluster.

If a database with the same name as your backup already exists, use the command `DROP DATABASE` to delete it and all users and roles associated with it.

1. Restore the `foo` database on one server. In this example, the `server01` member is used.

   ```
   bin/neo4j-admin database restore --from-path=/path/to/foo-backup-dir foo
   ```

2. Find the server ID of `server01` by logging in to Cypher Shell and running `SHOW SERVERS`. Cross-reference the address to find the server ID. Use any database to connect.

   ```
   SHOW SERVERS YIELD serverId, name, address, state, health, hosting;
   ```

   ```
   +-----------------------------------------------------------------------------------------------------------
   ------------------------------------------------+
   | serverId                             | name                                   | address          |
   state     | health      | hosting               |
   +-----------------------------------------------------------------------------------------------------------
   ------------------------------------------------+
   | "25a7efc7-d063-44b8-bdee-f23357f89f01" | "25a7efc7-d063-44b8-bdee-f23357f89f01" | "localhost:7689" |
   "Enabled" | "Available" | ["system",  "neo4j"] |
   | "782f0ee2-5474-4250-b905-4cd8b8f586ba" | "782f0ee2-5474-4250-b905-4cd8b8f586ba" | "localhost:7688" |
   "Enabled" | "Available" | ["system",  "neo4j"] |
   | "8512c9b9-d9e8-48e6-b037-b15b0004ca18" | "8512c9b9-d9e8-48e6-b037-b15b0004ca18" | "localhost:7687" |
   ```

```
"Enabled" | "Available" | ["system",  "neo4j"] |
+-------------------------------------------------------------------------------------------------------------
---------------------------------------------+
```

In this case, the address for `server01` is `localhost:7687` and thus, the server ID is `8512c9b9-d9e8-48e6-b037-b15b0004ca18`.

3. On one of the servers, use the `system` database and create the database `foo` using the server ID of `server01`. The topology of `foo` is stored in the `system` database and when you create it, it is allocated according to the default topology (which can be shown with `CALL dbms.showTopologyGraphConfig`). This may be different from the topology of `foo` when it was backed up. If you want to ensure a certain allocation across the cluster, you can specify the desired topology with the `TOPOLOGY` clause in the `CREATE DATABASE` command. See `CREATE DATABASE` for more information.

```
CREATE DATABASE foo
TOPOLOGY [desired number of primaries] PRIMARIES [desired number of secondaries] SECONDARIES
OPTIONS {existingData: 'use', existingDataSeedServer: '8512c9b9-d9e8-48e6-b037-b15b0004ca18'};
```

4. Verify that the `foo` database is online on the desired number of servers, in the desired roles. If the `foo` database is of considerable size, the execution of the command can take some time.

```
SHOW DATABASE foo;
```

```
+-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------+
| name  | type       | aliases | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage | default | home  | constituents |
+-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------+
| "foo" | "standard" | []      | "read-write" | "localhost:7687" | "primary" | FALSE  | "online"
| "online"      | ""            | FALSE   | FALSE | []           |
| "foo" | "standard" | []      | "read-write" | "localhost:7688" | "primary" | FALSE  | "online"
| "online"      | ""            | FALSE   | FALSE | []           |
| "foo" | "standard" | []      | "read-write" | "localhost:7689" | "primary" | TRUE   | "online"
| "online"      | ""            | FALSE   | FALSE | []           |
+-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------+

9 rows available after 3 ms, consumed after another 1 ms
```

Controlling locations with allowed/denied databases

A database can by default be allocated to run on any server in a cluster. However, it is possible to constrain the servers that specific databases are hosted on. This is done with `ENABLE SERVER` and `ALTER SERVER`, described in Managing servers in a cluster. The following options are available:

- `allowedDatabases` - a set of databases that are allowed to be hosted on a server.
- `deniedDatabases` - a set of databases that are denied to be hosted on a server. Allowed and denied are mutually exclusive.
- `modeConstraint` - controls in what mode (primary, secondary, or none) databases can be hosted on a server. If not set, there are no mode constraints on the server.

Default database in a cluster

The default database, as defined by `initial.dbms.default_database`, is automatically created when the DBMS starts for the first time. This provides a user database to experiment with immediately. However, this creation is 'best effort' for reasons explained below, and users are encouraged to create their own default database for production use. If you create your own default database, even if you just run `DROP DATABASE neo4j` and `CREATE DATABASE neo4j`, you do not have to be aware of the complexities below.

Automatic default database creation

The initial default database is created when the DBMS starts for the first time. It uses the following settings:

- `initial.dbms.default_database` — the name of the database.
- `initial.dbms.default_primaries_count` — the number of primaries requested for the default database.
- `initial.dbms.default_secondaries_count` — the number of secondaries requested for the default database.

However, it uses the default primary and secondary counts (topology) as maximums, not as hard requirements. This is different to normal database creation, where if the requested topology cannot be satisfied, creation fails. For the automatic creation of the default database alone, if the requested topology cannot be satisfied, you get as many of each hosting type as can be satisfied by the current cluster. This means you may end up with a default database with as few as one primary and no secondaries, despite the default values being higher. It is also possible to configure a cluster where automatic creation of the default database is not possible when the DBMS starts up. In this case, creation fails, a warning is logged, and creation **is not be re-attempted**.

Automatic creation of the initial default database works as follows:

- As the cluster starts for the first time, there is a configured threshold for how many servers are required to create the DBMS - `dbms.cluster.minimum_initial_system_primaries_count`.
- Once **a minimum** of this many servers have discovered each other, the `system` database bootstraps, allowing creation of the DBMS.
- The initial default database is created with those servers as the possible hosts.
- If any of the servers block hosting the default database (see `initial.server.denied_databases`), they are not used.
- If any of the servers restrict the mode they can host a database in, that is obeyed (see `initial.server.mode_constraint`).
- If there are too few servers to allocate the requested number of primaries, whichever ones available are used. If there are zero available primaries, automatic creation fails.
- If there are too few servers remaining after the primary allocation to satisfy the requested number of secondaries, whicever ones available are used.

Some possible behaviours that may be observed as a result of the above approach:

- If `initial.dbms.default_primaries_count` is larger than `dbms.cluster.minimum_initial_system_primaries_count`, you are likely to get an initial default

database with fewer primaries than the default. This is because DBMS initialisation only waits for the minimum system primaries.

- If `initial.dbms.default_secondaries_count` plus `initial.dbms.default_primaries_count` is larger than `dbms.cluster.minimum_initial_system_primaries_count`, you are likely to get an initial default database with fewer secondaries than the default. This is because DBMS initialisation only waits for the minimum number of system primaries.
- If you use `initial.server.denied_databases` to prevent the allocation of your default database to any of your initial servers, you may end up with fewer copies of the database than the default request, and possibly even no default database.
- If you use `initial.server.mode_constraint=SECONDARY` for any of your initial servers, you may end up with fewer primary copies of the database than the default request, and possibly even no default database.

Changing default database topology

If the default database is initially created for you with a topology different to what you want, you can update it in the same way as any database, see Alter topology.

Change the default database

You can use the procedure `dbms.setDefaultDatabase("newDefaultDatabaseName")` to change the default database for a DBMS.

1. Ensure that the database to be set as default exists, otherwise create it using the command `CREATE DATABASE <database-name>`.
2. Show the name and status of the current default database by using the command `SHOW DEFAULT DATABASE`.
3. Stop the current default database using the command `STOP DATABASE <database-name>`.
4. Run `CALL dbms.setDefaultDatabase("newDefaultDatabaseName")` against the `system` database to set the new default database.
5. Optionally, you can start the previous default database as non-default by using `START DATABASE <database-name>`.

Handling errors

Databases can get into error states. Typically you can observe this with the `SHOW DATABASES` command, and use the error handling guidance to help.

In more serious cases you may be dealing with a disaster situation, where the whole DBMS may not be responding correctly, or some specific databases cannot be restored without downtime. Refer to the disaster recovery guide for those situations.

Unbind the `system` database

You can use the `neo4j-admin dbms unbind-system-db` command to remove and archive the cluster state

for the `system` database, so that the server can rebind to a new `system` database in the cluster.

> **Caution:**
>
> Executing the `neo4j-admin dbms unbind-system-db` command does not affect the cluster state of other standard databases. However, since all servers in the DBMS need to be shut down to run the command, downtime will occur for all databases inside of the DBMS.

Syntax

The `neo4j-admin dbms unbind-system-db` command has the following syntax:

```
neo4j-admin dbms unbind-system-db [-h] [--expand-commands] [--verbose] [--archive-cluster-
state[=true|false]]
                                  [--additional-config=<file>] [--archive-path=<path>]
```

Description

Removes and archives cluster state for `system` database.

Options

The `neo4j-admin dbms unbind-system-db` command has the following options:

Table 396. `neo4j-admin dbms unbind-system-db` *options*

Option	Description	Default
`--additional-config=<file>`	Configuration file with additional configuration.	
`--archive-cluster-state[=true|false]`	Enable or disable the cluster state archiving.	`false`
`--archive-path=<path>`	Destination (file or folder) of the cluster state archive.	
`--expand-commands`	Allow command expansion in config value evaluation.	
`-h, --help`	Show this help message and exit.	
`--verbose`	Enable verbose output.	

Limitations

The Neo4j process must be shut down on all servers before running the `neo4j-admin dbms unbind-system-db` command.

Usage

The `neo4j-admin dbms unbind-system-db` command removes and archives cluster state for the `system` database, so that the server can rebind to a new `system` database in a cluster.

You must shut down all cluster members before running the command on each of them. For details, see

Disaster recovery guide → Make the `system` database write-available.

The `neo4j-admin dbms unbind-system-db` command does not affect the cluster state of the standard databases. However, since all servers in the DBMS must be shut down before the command is issued, downtime is expected for all databases within the DBMS.

Monitoring

Monitor servers

To monitor the state of individual servers in a cluster, use the `SHOW SERVERS` command.

Listing Servers

Syntax:

```
SHOW SERVERS
```

Returns:

Name	Type	Description
`name`	String	The friendly name of the server, or its UUID if no name is set.
`address`	String	The address of the Bolt port for the server. May be `null`.
`state`	String	The state of the server in the topology.
`health`	String	The current availability of the server.
`hosting`	List<String>	Databases that this server is currently running.

Possible values of `state`

- `Free` - server has been started, but not added to the cluster. It needs to be added with `ENABLE SERVER`.
- `Enabled` - server is part of cluster and can have database allocated to it.
- `Cordoned` - server may be hosting databases, but cannot have any more allocated to it.
- `Deallocating` - server is in the process of removing its databases. This may take some time, because it does not stop its copy of a database, if in primary mode, until another server has fully started its copy, to preserve the requested number of primaries.
- `Deallocated` - server has completed removing its databases.
- `Dropped` - server has been removed from the cluster, but the process has not exited.

Possible values of `health`

- `Available` - server has recently been in contact with the cluster member executing `SHOW SERVERS`.
- `Unavailable` - server has not had successful network communication with the cluster for a while.

Note:

`Unavailable` does not necessarily mean the server is not running, just that there are network problems connecting to it.

Example 66. Listing servers in a cluster

When running `SHOW SERVERS` against a cluster, expect similar output to the following:

```
+-------------------------------------------------------------------------------------------------
--+
| name                                 | address        | state   | health    | hosting
|
+-------------------------------------------------------------------------------------------------
--+
|"f4ae1895-26f1-4b93-bd31-
6f482be80d3d"|"localhost:7681"|"Enabled"|"Available"|["system","foo","neo4j"]|
|"ffa55a5b-2aca-45fc-be09-
2a894067025c"|"localhost:7682"|"Enabled"|"Available"|["system","foo","neo4j"]|
|"server3"                             |"localhost:7683"|"Enabled"|"Available"|["system","neo4j"]
|
+-------------------------------------------------------------------------------------------------
--+
```

Listing more details of servers

If more details about the servers are needed, `SHOW SERVERS` can be appended with `YIELD *`.

Syntax:

```
SHOW SERVERS YIELD *
```

Returns:

Name	Type	Description
`serverId`	String	The UUID of the server
`name`	String	The friendly name of the server, or its UUID if no name is set.
`address`	String	The address of the Bolt port for the server. May be `null`.
`httpAddress`	String	The address of the HTTP port for the server. May be `null`.
`httpsAddress`	String	The address of the HTTPS port for the server. May be `null`.
`state`	String	The state of the server in the topology.
`health`	String	The current availability of the server.
`hosting`	List<String>	Databases that this server is currently running.

Name	Type	Description
`requestedHosting`	List<String>	Databases that this server is supposed to be running. May be fewer databases if the server is in the process of safely stopping one, or more databases if the server is in the process of starting one up. Composite databases do not currently appear in this list, though they do appear in `hosting` for all servers.
`tags`	List<String>	Tags applied to this server. Used for routing policies.
`allowedDatabases`	List<String>	A list of the only databases that are allowed on this server. Empty means all are allowed.
`deniedDatabases`	List<String>	A list of databases that may not be hosted on this server. Empty means all are allowed.
`modeConstraint`	String	A limit on what modes (i.e. `primary` or `secondary`) a database can be in on this server.
`version`	String	The version of Neo4j this server is running.

Note:

Only one of `allowedDatabases` and `deniedDatabases` can be set as they are mutually exclusive.

Possible values of `modeConstraint`

- `NONE` - any modes can be allocated to this server.
- `PRIMARY` - only primary modes can be allocated to this server. These may be the target of writes for the database.
- `SECONDARY` - only secondary modes can be allocated to this server. These will never write to the database, only read.

Example 67. Listing more details about servers in a cluster

When running `SHOW SERVERS YIELD *` in a cluster, expect similar output to the following:

```
+--------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------+
| serverId                             | name                                 | address        |
httpAddress    | httpsAddress | state   | health    | hosting               | requestedHosting
| tags | allowedDatabases| deniedDatabases| modeConstraint| version        |
+--------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------+
```

```
|"f4ae1895-26f1-4b93-bd31-6f482be80d3d"|"f4ae1895-26f1-4b93-bd31-
6f482be80d3d"|"localhost:7681"|"localhost:7471"|null
|"Enabled"|"Available"|["system","foo","neo4j"]|["system","foo","neo4j"]|[]    |[]              |[]
|"NONE"         |"5.0.0-SNAPSHOT"|
|"ffa55a5b-2aca-45fc-be09-2a894067025c"|"ffa55a5b-2aca-45fc-be09-
2a894067025c"|"localhost:7682"|"localhost:7472"|null
|"Enabled"|"Available"|["system","foo","neo4j"]|["system","foo","neo4j"]|[]    |[]              |[]
|"NONE"         |"5.0.0-SNAPSHOT"|
|"72bd3d0f-c1d1-4d39-9da7-015f5656e40b"|"server3"
|"localhost:7683"|"localhost:7473"|null          |"Enabled"|"Available"|["system","neo4j"]
|["system","neo4j"]      |[]    |[]               |[]              |"NONE"         |"5.0.0-SNAPSHOT"|
+-------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
```

Monitor databases

In addition to the system-wide metrics and logs described in previous sections, to monitor the state of individual databases hosted in a cluster, use the `SHOW DATABASES` command.

Listing Databases

Syntax:

```
SHOW DATABASES
```

Returns:

Name	Type	Description
`name`	String	The human-readable name of the database.
`type`	String	`standard`, `system` or `composite`.
`aliases`	List<String>	The names of any aliases the database may have.
`access`	String	The database access mode, either `read-write` or `read-only`.
`address`	String	The bolt address of the server hosting the database.
`role`	String	The cluster role which the server fulfills for this database.
`writer`	Boolean	`true` for the database node that accepts writes. This node is either the leader for this database in a cluster or this is a standalone server.
`requestedStatus`	String	The state that an operator has requested the database to be in.
`currentStatus`	String	The state the database is actually in on this server.

Name	Type	Description
`statusMessage`	String	A message explaining the current state of the database, which could be an error encountered by the Neo4j server when transitioning the database to `requestedStatus`, if any.
`default`	Boolean	Whether this database is the default for this DBMS.
`home`	Boolean	Whether this database is the home database for this user.
`constituents`	List<String>	A list of alias names making up this Composite database, null for non-Composite databases.

Note that for failed databases, `currentStatus` and `requestedStatus` are different. This can imply an error. For example:

- A database may take a while to transition from `offline` to `online`, due to performing recovery.
- During normal operation, the `currentStatus` of a database may be transiently different from its `requestedStatus`, due to a necessary automatic process, such as one server copying store files from another.

The possible values for `currentStatus` are `online`, `offline`, `starting`, `stopping`, `store copying`, `initial`, `deallocating`, `dirty`, `quarantined`, and `unknown`. The `requestedStatus` can only be `online` or `offline`. See Database states for more information.

Additionally, note that databases hosted on servers that are offline are also returned by the `SHOW DATABASES` command. For such databases the `address` column displays `NULL`, the `currentStatus` column displays `unknown`, and the `statusMessage` displays `Server is unavailable`.

Example 68. Listing databases in standalone Neo4j

When executing `SHOW DATABASES` against a standalone server, the following output is expected:

```
+--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------+
| name    | type      | aliases | access     | address        | role      | writer | requestedStatus
| currentStatus | statusMessage | default | home | constituents |
+--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------+
|"neo4j"  |"standard" |[]       |"read-write"|"localhost:7687"| "primary" | true   | "online"
| "online"      | ""            |true     |true  |[]           |
|"system" |"system"   |[]       |"read-write"|"localhost:7687"| "primary" | true   | "online"
| "online"      | ""            |false    |false |[]           |
+--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------+
```

Note that the `role`, `writer`, and `address` columns are primarily intended to distinguish between the states of a given database, across multiple servers deployed in a cluster. In a standalone deployment with a single server, the `address` field should be the same for every database, the `role` field should

always be "primary", and the `writer` field should be true.

Example 69. Listing databases in a cluster

When running `SHOW DATABASES` against a cluster, expect similar output to the following:

```
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------+
| name    | type      | aliases | access     | address        | role      | writer | requestedStatus
| currentStatus | statusMessage | default | home | constituents |
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------+
|"neo4j"  |"standard" |[]       |"read-write"|"localhost:7681"|"primary"  |false   |"online"
|"online"       |""             |true     |true  |[]            |
|"neo4j"  |"standard" |[]       |"read-write"|"localhost:7682"|"primary"  |false   |"online"
|"online"       |""             |true     |true  |[]            |
|"neo4j"  |"standard" |[]       |"read-write"|"localhost:7683"|"primary"  |true    |"online"
|"online"       |""             |true     |true  |[]            |
|"neo4j"  |"standard" |[]       |"read-write"|"localhost:7684"|"secondary"|false   |"online"
|"online"       |""             |true     |true  |[]            |
|"system" |"system"   |[]       |"read-write"|"localhost:7681"|"primary"  |true    |"online"
|"online"       |""             |false    |false |[]            |
|"system" |"system"   |[]       |"read-write"|"localhost:7682"|"primary"  |false   |"online"
|"online"       |""             |false    |false |[]            |
|"system" |"system"   |[]       |"read-write"|"localhost:7683"|"primary"  |false   |"online"
|"online"       |""             |false    |false |[]            |
|"system" |"system"   |[]       |"read-write"|"localhost:7684"|"secondary"|false   |"online"
|"online"       |""             |false    |false |[]            |
|"foo"    |"standard" |[]       |"read-write"|"localhost:7681"|"primary"  |true    |"online"
|"online"       |""             |false    |false |[]            |
|"foo"    |"standard" |[]       |"read-write"|"localhost:7684"|"secondary"|false   |"online"
|"online"       |""             |false    |false |[]            |
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------+
```

Note that `SHOW DATABASES` does **not** return 1 row per database. Instead, it returns 1 row per database, per server that hosts it in the cluster. Therefore, if the cluster has four servers hosting 3 databases each with 3 primaries and one secondary, 12 rows are displayed. In addition this means that if all the servers that host a database are offline, the database will not appear in the results of `SHOW DATABASES`.

The possible roles are "primary", "secondary", and "unknown".

Note that different servers may have different roles for each database, and a server may have different roles for different databases.

If a database is offline on a particular server, either because it was stopped by an operator, or an error has occurred, its cluster `role` is "unknown".

Listing a single database

The number of rows returned by `SHOW DATABASES` can be quite large, especially when run in a cluster. You can filter the rows returned by database name (e.g. "foo") by using the command `SHOW DATABASE foo`.

Syntax:

```
SHOW DATABASE databaseName
```

Arguments:

Name	Type	Description
`databaseName`	String	The name of the database whose status to report.

Returns:

Name	Type	Description
`name`	String	The human-readable name of the database.
`type`	String	`standard`, `system`, or `composite`.
`aliases`	List<String>	The names of any aliases the database may have.
`access`	String	The database access mode, either `read-write` or `read-only`.
`address`	String	The bolt address of the server hosting the database.
`role`	String	The cluster role which the server fulfills for this database.
`writer`	Boolean	`true` for the database node that accepts writes. This node is either the leader for this database in a cluster or this is a standalone server.
`requestedStatus`	String	The state that an operator has requested the database to be in.
`currentStatus`	String	The state the database is actually in on this server.
`statusMessage`	String	A message explaining the current state of the database, which could be an error encountered by the Neo4j server when transitioning the database to `requestedStatus`, if any.
`default`	Boolean	Whether this database is the default for this DBMS.
`home`	Boolean	Whether this database is the home database for this user.
`constituents`	List<String>	A list of alias names making up this Composite database, null for non-Composite databases.

Example 70. Listing statuses for database foo

When running `SHOW DATABASE foo` in a cluster, expect similar output to the following:

```
+-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------
------------------+
| name    | type      | aliases   | access      | address         | role     | writer  |
requestedStatus  | currentStatus  | statusMessage                                    | default  |
home  | constituents |
+-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------
------------------+
| "foo"    | "standard" | []        | "read-write" | "localhost:7681" | "primary" | false    |
"online"          | "online"        | ""                                                | true      |
true   | []           |
| "foo"    | "standard" | []        | "read-write" | "localhost:7682" | "unknown" | false    |
"online"          | "dirty"         | "An error occurred! Unable to start database ..." | true      |
true   | []           |
| "foo"    | "standard" | []        | "read-write" | "localhost:7683" | "primary" | true     |
"online"          | "online"        | ""                                                | true      |
true   | []           |
| "foo"    | "standard" | []        | "read-write" | "localhost:7684" | "unknown" | false    |
"online"          | "dirty"         | "An error occurred! Unable to start database ..." | true      |
true   | []           |
+-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------
------------------+
```

Listing more details about databases

If more details about the databases are needed, `SHOW DATABASES` can be appended with `YIELD *`.

Syntax:

```
SHOW DATABASES YIELD *
```

Returns:

Name	Type	Description	Example value
`name`	String	The human-readable name of the database.	"foo"
`type`	String	`standard`, `system` or `composite`	"standard"
`aliases`	List<String>	Aliases of the database.	"[]"
`access`	String	`read-write` or `read-only`	"read-write"
`databaseID`	String	The ID for the database.	"CC573A1DF4...."
`serverID`	String	The friendly name or UUID of the server hosting this database.	"server3"
`address`	String	The Bolt address of the server hosting the database.	"localhost:7683"
`role`	String	The cluster role which the server fulfills for this database.	"primary"

Name	Type	Description	Example value
`writer`	Boolean	Whether the database accepts writes on this server.	true
`requestedStatus`	String	The state that an operator has requested the database to be in.	"online"
`currentStatus`	String	The state the database is actually in on this server.	"online"
`statusMessage`	String	Error encountered by the server when transitioning the database to `requestedStatus`, if any.	""
`default`	Boolean	Whether this database is the default for this DBMS.	false
`home`	Boolean	Whether this database is the user's home database.	true
`currentPrimariesCount`	Integer	Number of primaries for this database reported as running currently. It is the same as the number of rows where `role` = `primary` and `name` =this database	3
`currentSecondariesCount`	Integer	Number of secondaries for this database reported as running currently. It is the same as the number of rows where `role` = `secondary` and `name` =this database	0
`requestedPrimariesCount`	Integer	The requested number of primaries for this database. May be lower than current if the DBMS is currently reducing the number of copies of the database, or higher if it is currently increasing the number of copies.	3
`requestedSecondariesCount`	Integer	The requested number of secondaries for this database. May be lower than current if the DBMS is currently reducing the number of copies of the database, or higher if it is currently increasing the number of copies.	1
`creationTime`	Datetime	The timestamp of the creation of this database.	"2022-09-09T12:58:21.923000000Z"

Name	Type	Description	Example value
`lastStartTime`	Datetime	The timestamp of the most recent time this database was started It is the same as creation time unless the database has been stopped at some point.	"2022-09-09T12:58:21.923000000Z"
`lastStopTime`	Datetime	The timestamp of the most recent time this database was stopped (`STOP DATABASE`)	null
`store`	String	The store format.	"record-aligned-1.1"
`lastCommittedTxn`	Integer	The latest committed transaction number on this database server. May be different between members when changes have not propagated.	2342
`replicationLag`	Integer	The difference in transaction numbers between this server and the writer of this database. If this is persistently high, there may be a problem.	1
`constituents`	List<String>	A list of alias names making up this Composite database, null for non-Composite databases.	"[]"

Note:

Deviating values for `currentPrimariesCount` and `requestedPrimariesCount`, and for `currentSecondariesCount` and `requestedSecondariesCount` may not be a cause for concern. These values can vary while the cluster is allocating, reallocating, and/or deallocating databases.

Monitor cluster endpoints for status information

A cluster exposes some HTTP endpoints which can be used to monitor the health of the cluster. This section describes these endpoints and explains their semantics.

Adjusting security settings for clustering endpoints

If authentication and authorization is enabled in Neo4j, the clustering status endpoints also require authentication credentials. The setting `dbms.security.auth_enabled` controls whether the native auth provider is enabled. For some load balancers and proxy servers, providing authentication credentials with the request is not an option. For those situations, consider disabling authentication of the clustering status endpoints by setting `dbms.security.cluster_status_auth_enabled=false` in *neo4j.conf*.

Unified endpoints

A unified set of endpoints exist, both on primary and secondary servers, with the following behavior:

- `/db/<databasename>/cluster/writable` — Used to direct `write` traffic to specific instances.
- `/db/<databasename>/cluster/read-only` — Used to direct `read` traffic to specific instances.
- `/db/<databasename>/cluster/available` — Available for the general case of directing arbitrary request types to instances that are available for processing read transactions.
- `/db/<databasename>/cluster/status` — Gives a detailed description of this instance's view of its status within the cluster, for the given database.
- `/dbms/cluster/status` — Gives a detailed description of this instance's view of its status within the cluster, for all databases. Useful for monitoring and coordinating rolling upgrades. See Status endpoints for further details.

Every `/db/<databasename>/*` endpoint targets a specific database. The `databaseName` path parameter represents the name of the database. By default, a fresh Neo4j installation with two databases `system` and `neo4j` has the following cluster endpoints:

```
http://localhost:7474/dbms/cluster/status

http://localhost:7474/db/system/cluster/writable
http://localhost:7474/db/system/cluster/read-only
http://localhost:7474/db/system/cluster/available
http://localhost:7474/db/system/cluster/status

http://localhost:7474/db/neo4j/cluster/writable
http://localhost:7474/db/neo4j/cluster/read-only
http://localhost:7474/db/neo4j/cluster/available
http://localhost:7474/db/neo4j/cluster/status
```

Note:

Attempting to access endpoints for a database not hosted on a server produces a `404` response.

Table 397. Unified HTTP endpoint responses

Endpoint	Instance state	Returned code	Body text
`/db/<databasename>/cluster/writable`	Leader	`200 OK`	`true`
	Follower	`404 Not Found`	`false`
	Secondary	`404 Not Found`	`false`
`/db/<databasename>/cluster/read-only`	Leader	`404 Not Found`	`false`
	Follower	`200 OK`	`true`
	Secondary	`200 OK`	`true`

Endpoint	Instance state	Returned code	Body text
`/db/<databasename>/cluster/available`	Leader	`200 OK`	`true`
	Follower	`200 OK`	`true`
	Secondary	`200 OK`	`true`
`/db/<databasename>/cluster/status`	Leader	`200 OK`	JSON - See Status endpoint for details.
	Follower	`200 OK`	JSON - See Status endpoint for details.
	Secondary	`200 OK`	JSON - See Status endpoint for details.
`/dbms/cluster/status`	Leader	`200 OK`	JSON - See Status endpoint for details.
	Follower	`200 OK`	JSON - See Status endpoint for details.
	Secondary	`200 OK`	JSON - See Status endpoint for details.

Example 71. Use a clustering monitoring endpoint

From the command line, a common way to ask those endpoints is to use `curl`. With no arguments, `curl` does an HTTP `GET` on the URI provided and outputs the body text, if any. If the response code is desired, just add the `-v` flag for verbose output. Here are some examples:

- Requesting `writable` endpoint on a primary database allocation that is currently elected leader with verbose output:

```
#> curl -v localhost:7474/db/neo4j/cluster/writable
* About to connect() to localhost port 7474 (#0)
*   Trying ::1...
* connected
* Connected to localhost (::1) port 7474 (#0)
> GET /db/neo4j/cluster/writable HTTP/1.1
> User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
> Host: localhost:7474
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/plain
< Access-Control-Allow-Origin: *
< Transfer-Encoding: chunked
< Server: Jetty(9.4.17)
<
* Connection #0 to host localhost left intact
true* Closing connection #0
```

Status endpoints

The status endpoint, available at `/db/<databasename>/cluster/status`, is to be used to assist with rolling upgrades. For more information, see Upgrade and Migration Guide → Clusters.

Typically, it is desired to have some guarantee that a primary is safe to shutdown for each database before removing it from a cluster. Counter-intuitively, that a primary is safe to shutdown means that a majority of the **other** primaries are healthy, caught up, and have recently heard from that database's leader. The status endpoints provide the following information in order to help resolve such issues.

Note:

Several of the fields in status endpoint responses refer to details of Raft, the algorithm used in Neo4j clusters to provide highly available transactions. In a Neo4j cluster each database has its own independent Raft group. Therefore, details such as `leader` and `raftCommandsPerSecond` are database-specific.

Example 72. Example status response

```
{
  "lastAppliedRaftIndex":0,
  "votingMembers":["30edc1c4-519c-4030-8348-7cb7af44f591","80a7fb7b-c966-4ee7-88a9-35db8b4d68fe"
,"f9301218-1fd4-4938-b9bb-a03453e1f779"],
  "memberId":"80a7fb7b-c966-4ee7-88a9-35db8b4d68fe",
  "leader":"30edc1c4-519c-4030-8348-7cb7af44f591",
  "millisSinceLastLeaderMessage":84545,
  "participatingInRaftGroup":true,
  "core":true,
  "isHealthy":true,
  "raftCommandsPerSecond":124
}
```

Table 398. Status endpoint descriptions

Field	Type	Optional	Example	Description
`core`	boolean	no	`true`	Used to distinguish between if the server is hosting the database in primary (core) or secondary mode.
`lastAppliedRaftIndex`	number	no	`4321`	Every transaction in a cluster is associated with a raft index. Gives an indication of what the latest applied raft log index is.
`participatingInRaftGroup`	boolean	no	`false`	A participating member is able to vote. A primary is considered participating when it is part of the voter membership and has kept track of the leader.
`votingMembers`	string[]	no	`[]`	A member is considered a voting member when the leader has been receiving communication with it. List of member's `memberId` that are considered part of the voting set by this primary.

Field	Type	Optional	Example	Description
`isHealthy`	boolean	no	`true`	Indicates that the local database on this cluster member has not encountered a critical error that can block database operation. **Caution:** The `isHealthy` status applies only to this member's local database. It does not reflect the overall health of the database if it is clustered. A cluster member may still report `"isHealthy": true` even if the database currently has no leader and so cannot accept write transactions.
`memberId`	string	no	`30edc1c4-519c-4030-8348-7cb7af44f591`	Every member in a cluster has it's own unique member id to identify it. Use `memberId` to distinguish between primary and secondary servers.
`leader`	string	yes	`80a7fb7b-c966-4ee7-88a9-35db8b4d68fe`	Follows the same format as `memberId`, but if it is null or missing, then the leader is unknown.
`millisSinceLastLeaderMessage`	number	yes	`1234`	The number of milliseconds since the last heartbeat-like leader message. Not relevant to secondaries, and hence is not included.
`raftCommandsPerSecond` Deprecated in 5.4	number	yes	`124`	An estimate of the average Raft state machine throughput over a sampling window configurable via `clustering.status_throughput_window` setting. `raftCommandsPerSecond` is not an effective way to monitor that servers are not falling behind in updated and is hence deprecated and will be removed in the next major release of Neo4j. It is recommended to use the metric `<prefix>.cluster.raft.commit_index` on each server and look for divergence instead.

After an instance has been switched on, the status endpoint can be accessed in order to make sure all the guarantees listed in the table below are met.

To get the most accurate view of a cluster it is strongly recommended to access the *status endpoint* on all primary members and compare the result. The following table explains how results can be compared.

Table 399. Measured values, accessed via the status endpoint

Name of check	Method of calculation	Description
`allServersAreHealthy`	Every primary's status endpoint indicates `isHealthy` == `true`.	To ensure the data across the entire cluster is healthy. Whenever any primaries are false that indicates a larger problem.
`allVotingSetsAreEqual`	For any 2 primaries (A and B), status endpoint A's `votingMembers` == status endpoint B's `votingMembers`.	When the voting begins, all the primaries are equal to each other, and all members agree on membership.
`allVotingSetsContainAtLeastTargetCluster`	For all primaries (S), excluding primary Z (to be switched off), every member in S contains S in their voting set. Membership is determined by using the `memberId` and `votingMembers` from the status endpoint.	Sometimes network conditions are not perfect and it may make sense to switch off a different primary than the one originally was to be switched off. If this check is run for all primaries, the ones that match this condition can be switched off (providing other conditions are also met).
`hasOneLeader`	For any 2 primaries (A and B), `A.leader == B.leader && leader!=null`.	If the leader is different then there may be a partition (alternatively, this could also occur due to bad timing). If the leader is unknown, that means the leader messages have actually timed out.
`noMembersLagging`	For primary A with `lastAppliedRaftIndex` = `min` and primary B with `lastAppliedRaftIndex` = `max` `B.lastAppliedRaftIndex-A.lastAppliedRaftIndex<raftIndexLagThreshold`.	If there is a large difference in the applied indexes between primaries, then it could be dangerous to switch off a primary.

Note:

`raftIndexLagThreshold` helps you to monitor the lag in applying Raft log entries across a cluster and set appropriate thresholds. You should pick a `raftIndexLagThreshold` appropriate to your particular cluster and workload. Measuring the reported lag under normal circumstances and selecting a threshold slightly above that would be a good way to select an appropriate value. For example, you observe the metric (the difference between the maximum and minimum `lastAppliedRaftIndex`) during all phases of the specific workload and see that it spends all of the working hours around 100 or less, but on Saturdays it spikes to 5,000 for a few hours. Then, depending on your monitoring needs or capabilities, you either set a weekday threshold of 120 and a weekend threshold of 6,000 or just an overall threshold of 6,000. These thresholds can help in identifying performance issues.

Combined status endpoints

When using the status endpoints to support a rolling upgrade, it is required to assess whether a primary is safe to shut down for **all** databases. To avoid having to issue a separate request to each

`/db/<databasename>/cluster/status` endpoint, use the `/dbms/cluster/status` instead.

This endpoint returns a json array, the elements of which contain the same fields as the single database version, along with fields for for `databaseName` and `databaseUuid`.

Example 73. Example combined status response


```
[
  {
    "databaseName": "neo4j",
    "databaseUuid": "f4dacc01-f88a-4512-b3bf-68f7539c941e",
    "databaseStatus": {
      "lastAppliedRaftIndex": -1,
      "votingMembers": [
        "0cff51ad-7cee-44cc-9102-538fc4544b95",
        "90ff5df1-f5f8-4b4c-8289-a0e3deb2235c",
        "99ca7cd0-6072-4387-bd41-7566a98c6afc"
      ],
      "memberId": "90ff5df1-f5f8-4b4c-8289-a0e3deb2235c",
      "leader": "90ff5df1-f5f8-4b4c-8289-a0e3deb2235c",
      "millisSinceLastLeaderMessage": 0,
      "raftCommandsPerSecond": 0.0,
      "core": true,
      "participatingInRaftGroup": true,
      "healthy": true
    }
  },
  {
    "databaseName": "system",
    "databaseUuid": "00000000-0000-0000-0000-000000000001",
    "databaseStatus": {
      "lastAppliedRaftIndex": 7,
      "votingMembers": [
        "0cff51ad-7cee-44cc-9102-538fc4544b95",
        "90ff5df1-f5f8-4b4c-8289-a0e3deb2235c",
        "99ca7cd0-6072-4387-bd41-7566a98c6afc"
      ],
      "memberId": "90ff5df1-f5f8-4b4c-8289-a0e3deb2235c",
      "leader": "90ff5df1-f5f8-4b4c-8289-a0e3deb2235c",
      "millisSinceLastLeaderMessage": 0,
      "raftCommandsPerSecond": 0.0,
      "core": true,
      "participatingInRaftGroup": true,
      "healthy": true
    }
  }
]
```


Monitor replication status

Enterprise Edition | Introduced in 5.24

Neo4j 5.24 introduces the `dbms.cluster.statusCheck()` procedure, which can be used to monitor the ability to replicate in clustered databases. In most cases this means a clustered database is write available. The procedure identifies which members of a clustered database are up-to-date and can participate in successful replication. Therefore, it is useful in determining the fault tolerance of a clustered database. Additionally, you can use the procedure to identify the leader of a clustered database within the cluster.

> **Note:**
>
> The procedure replicates a dummy transaction within the cluster and verifies that it can be

replicated and applied. Since the status check does not replicate an actual transaction, it does not guarantee write availability, as other factors in the write path (e.g., database issues) may block transactions. However, a healthy status typically indicates write availability in most cases.

Cluster status check

Syntax

```
CALL dbms.cluster.statusCheck(databases :: LIST<STRING>, timeoutMilliseconds = null :: INTEGER)
```

Input arguments

Name	Type	Description
`databases`	List<String>	Databases for which the status check should run. Providing an empty list runs the status check for all **clustered** databases on that server, i.e. it does not run on singles or secondaries.
`timeoutMilliseconds`	Integer	How long to allow for replication, before returning it was unsuccessful. Default value is 1000 milliseconds.

Return arguments

The procedure returns a row for all primary members of all the requested databases where each row consists of:

Name	Type	Description
`database`	String	The database for which a `status check entry` was replicated.
`serverId`	String	The UUID of the server, which did or did not participate in a successful replication of the `status check entry`.
`serverName`	String	The friendly name of the server, or its UUID if no name is set.
`address`	String	The address of the Bolt port for the server.
`replicationSuccessful`	Boolean	Indicates if the server (on which the procedure is run) can replicate a transaction.
`memberStatus`	String	The status of each primary member.
`recognisedLeader`	String	The server id of the perceived leader of each primary member.

Name	Type	Description
`recognisedLeaderTerm`	Integer	The term of the perceived leader of each primary member. If the members report different leaders, the one with the highest term should be trusted.
`requester`	Boolean	Whether a server is the requester or not.
`error`	String	Contains the error message if one is present. An example of an error is that one or more of the requested databases do not exist on the requester.

Possible values of `replicationSuccessful`

- `TRUE` — if this server managed to replicate the dummy transaction to a majority of cluster members within the given timeout.
- `FALSE` — if it failed to replicate within the timeout. The value is the same column-wise. A failed replication can either indicate a real issue in the cluster (e.g., no leader) or that this server is too far behind in applying updates and can't replicate.

Possible values of `memberStatus`

- `APPLYING` means that the member can replicate and is actively applying transactions.
- `REPLICATING` means that the member can participate in replicating but cannot apply. This state is uncommon, but may happen while waiting for the database to start and accept transactions.
- `UNAVAILABLE` means that the member is either too far behind the leader or unreachable. They are unhealthy and cannot add to the fault-tolerance.

Possible values of `requester`

- `TRUE` — for the server on which the procedure is run.
- `FALSE` — on the remaining servers.

In general, you can use the `replicationSuccessful` field to determine overall write-availability, whereas the `memberStatus` field can be checked in order to see whether the database is fault-tolerant or not.

Example

Running the status check

When running the cluster status check against a server, expect similar output to the following:

```
+-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
----------------------+
| database | serverId                              | serverName                              | address
| replicationSuccessful | memberStatus | recognisedLeader                       | recognisedLeaderTerm |
requester | error |
+-----------------------------------------------------------------------------------------------------------
```

```
-----------------------------------------------------------------------------------------------------------------
----------------------+
| "neo4j"  | "d3fe2e6a-494d-4ab8-81b1-7de2ce31ce11" | "d3fe2e6a-494d-4ab8-81b1-7de2ce31ce11" |
"localhost:7682" | TRUE                  | "APPLYING"   | "565130e8-b8f0-41ad-8f9d-c660bd8d5519" | 4
| FALSE     | NULL  |
| "neo4j"  | "565130e8-b8f0-41ad-8f9d-c660bd8d5519" | "565130e8-b8f0-41ad-8f9d-c660bd8d5519" |
"localhost:7681" | TRUE                  | "APPLYING"   | "565130e8-b8f0-41ad-8f9d-c660bd8d5519" | 4
| TRUE      | NULL  |
| "neo4j"  | "58c70f4b-910d-4d0e-b0f2-3084554079ec" | "58c70f4b-910d-4d0e-b0f2-3084554079ec" |
"localhost:7683" | TRUE                  | "APPLYING"   | "565130e8-b8f0-41ad-8f9d-c660bd8d5519" | 4
| FALSE     | NULL  |
| "system" | "565130e8-b8f0-41ad-8f9d-c660bd8d5519" | "565130e8-b8f0-41ad-8f9d-c660bd8d5519" |
"localhost:7681" | TRUE                  | "APPLYING"   | "d3fe2e6a-494d-4ab8-81b1-7de2ce31ce11" | 1
| TRUE      | NULL  |
| "system" | "58c70f4b-910d-4d0e-b0f2-3084554079ec" | "58c70f4b-910d-4d0e-b0f2-3084554079ec" |
"localhost:7683" | TRUE                  | "APPLYING"   | "d3fe2e6a-494d-4ab8-81b1-7de2ce31ce11" | 1
| FALSE     | NULL  |
| "system" | "d3fe2e6a-494d-4ab8-81b1-7de2ce31ce11" | "d3fe2e6a-494d-4ab8-81b1-7de2ce31ce11" |
"localhost:7682" | TRUE                  | "APPLYING"   | "d3fe2e6a-494d-4ab8-81b1-7de2ce31ce11" | 1
| FALSE     | NULL  |
+-----------------------------------------------------------------------------------------------------------------
----------------------+
```

Resilient multi-region cluster deployment

Designing a resilient multi-data center cluster

Overview

The goal of deploying a resilient multi-data center cluster is to achieve high availability, disaster recovery, and tolerance against the loss of a data center.

You should take into account cluster architecture and topology and decide where database primaries and secondaries are located, balancing performance and fault tolerance. See Introduction: Neo4j clustering architecture for recommendations on database topologies.

Pay attention to networking and traffic routing:

- If database primaries are distant from each other, that will increase your write latency.
- To commit a change, the writer primary must get confirmation from a quorum of members, including itself. If primaries are far apart, network latency adds to commit time.

Recommended cluster design patterns

Read resilience with user database secondaries

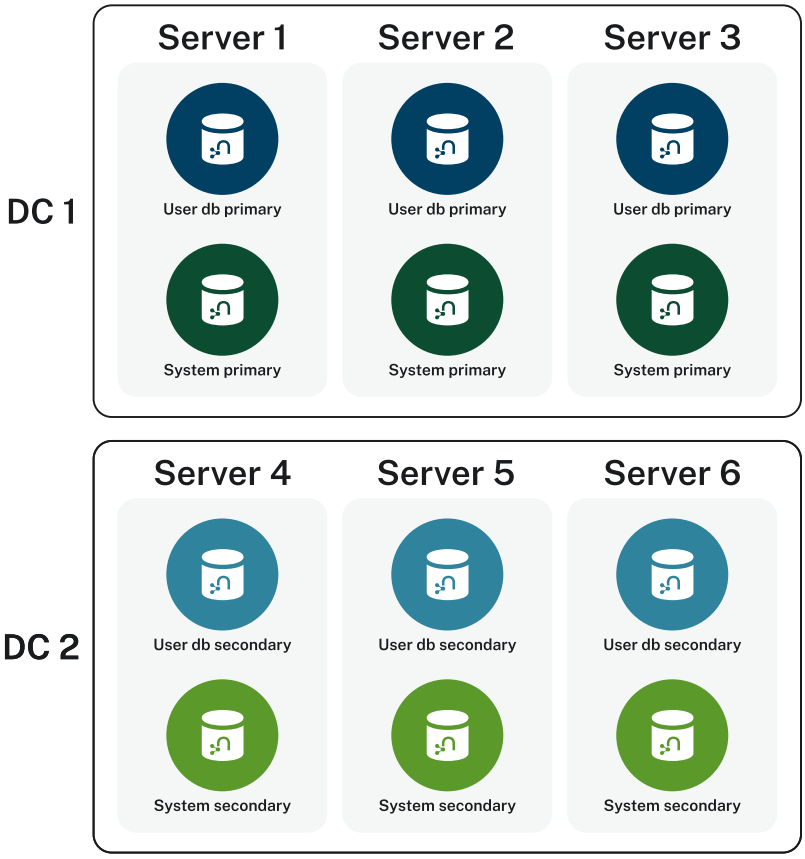


Figure 11. Cluster design with database secondaries for better read performance

For better read performance, you can locate all database primaries in one data center (DC) and the

secondaries in another DC. This setup also provides fast writes, because they will be performed within the single DC.

However, if the DC with primaries goes down, your cluster loses write availability. Though read availability may remain via the secondaries.

Recovering from the loss of a data center

You can restore the cluster write availability without the failed DC:

- If you have enough secondary members of the database in another DC, you can switch their mode to primary and not have to store a copy or wait a long time for primary copies to restore.
- You can use secondaries to re-seed databases if needed. See the `dbms.recreateDatabase()` procedure for more details.

  Example recovery steps

  1. Promote secondary copies of the `system` database to primaries to make the `system` database write-available. This requires restarting processes. For other scenarios, see the steps in the Disaster recovery guide on how to make the `system` database write-available again.
  2. Mark missing servers as not available by cordoning them. For each `Unavailable` server, run `CALL dbms.cluster.cordonServer("unavailable-server-id")` on the remaining cluster.
  3. Recreate each user database, letting it choose the existing servers as seeders. You need to accept a smaller topology that will fit in the remaining DC.

For detailed scenarios, see the Disaster recovery guide.

Geo-distribution of primary databases

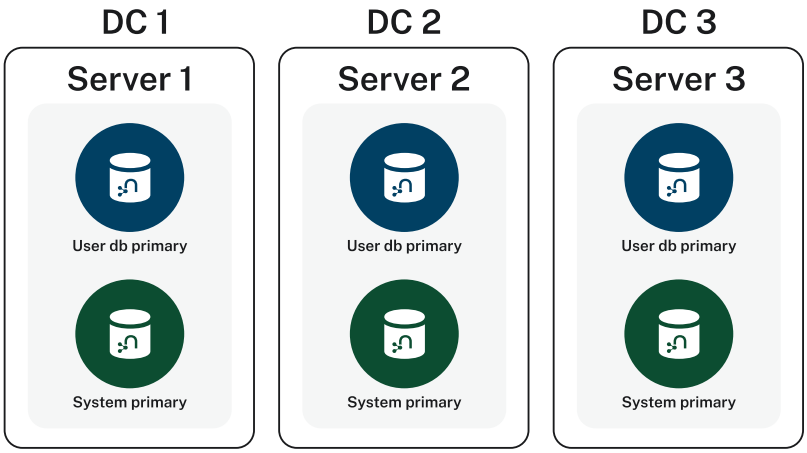


Figure 12. Cluster design with primaries distributed across three data centers

You can place each primary copy in a different data center (DC), using at least three data centers.

Therefore, if one DC fails, only a single primary member is lost, and the cluster can continue operating without data loss.

However, you always pay cross-data center latency times for every write operation.

Recovering from the loss of a data center

This setup has no loss of quorum, so the cluster keeps running — only with reduced fault tolerance (with no room for extra failures).

To restore fault tolerance, you can either wait until the affected DC is back online or start a new primary member somewhere else that will provide resilience and re-establish three-DC fault tolerance.

Example recovery steps

1. Start and enable a new server. See How to add a server to the cluster for details.
2. Remove the unavailable server from the cluster:
    a. First, deallocate databases from it.
    b. Then drop the server.

    For more information, visit the Managing servers in a cluster.

For detailed scenarios, see the Disaster recovery guide.

Exclusive geo-distribution for the `system` database

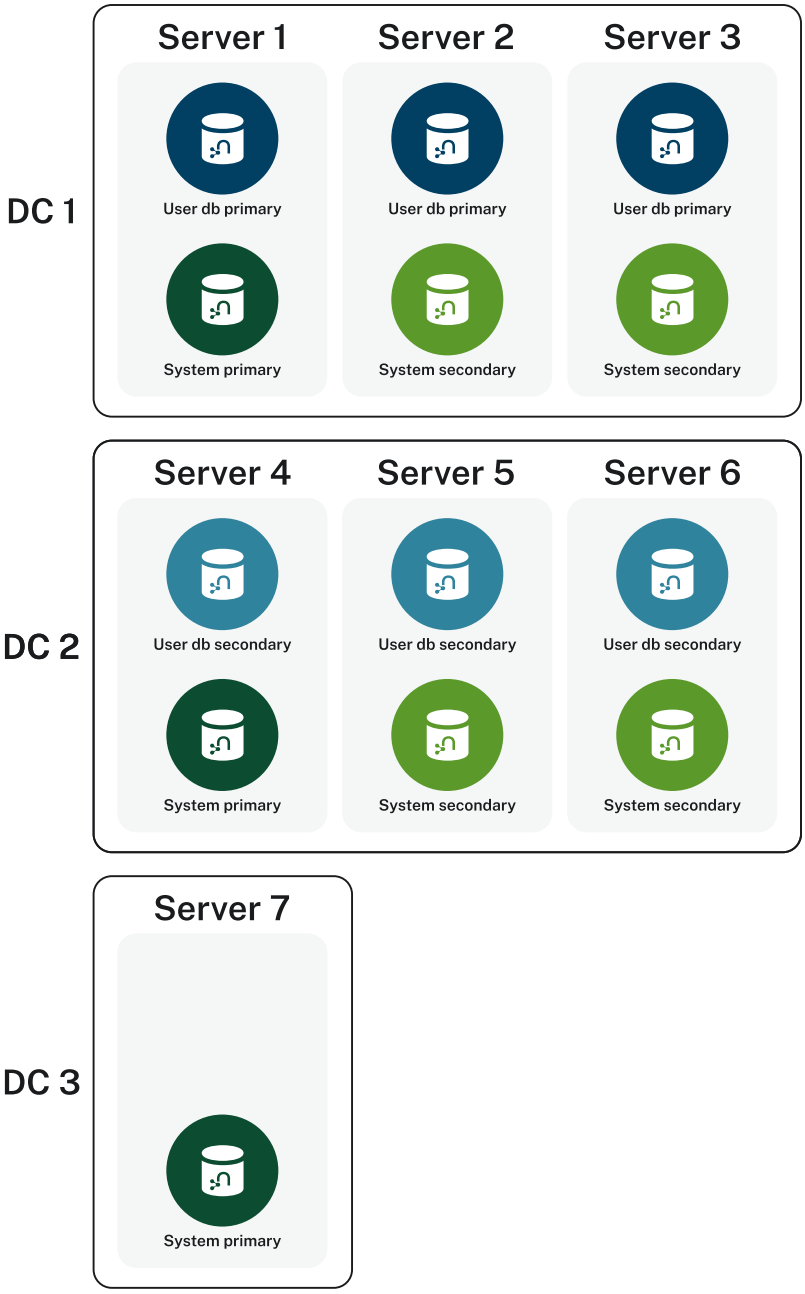


Figure 13. Primaries for the `system` database distributed across three data centers

You can place all primaries for user databases in one data center (DC) and all secondaries in another.

In a third DC, deploy a server that only hosts a primary member of the `system` database (in addition to those in the first two data centers).

- This server can be a small machine, since the `system` database has minimal resource requirements.
- To prevent user databases from being allocated to it, set the `allowedDatabases` constraint to some name that will never be used.

Your writes will be fast, because they occur within the single DC.

If a DC goes down, you retain write availability for the `system` database, which makes restoring write availability to the user databases easier.

However, if the DC with primaries goes down, the user databases will become write-unavailable. Though read availability may still be maintained via the secondaries.

If you lose the DC with primaries in, the user databases will go write-unavailable, though the secondaries should continue to provide read availability. Because of the third DC, the `system` database remains write-available, so you will be able to get the user databases back to write-available without process downtime.

However, if you need to use the `dbms.recreateDatabase()` procedure, it will involve downtime for the user database.

Example recovery steps

1. Mark missing servers as not present by cordoning them. For each `Unavailable` server, run `CALL dbms.cluster.cordonServer("unavailable-server-id")` on one of the available servers.
2. Recreate each user database, letting it select the existing servers as seeders. You need to accept a smaller topology that will fit in the remaining data center.

For detailed scenarios, see the Disaster recovery guide.

Fault tolerance to the failure of any two primaries

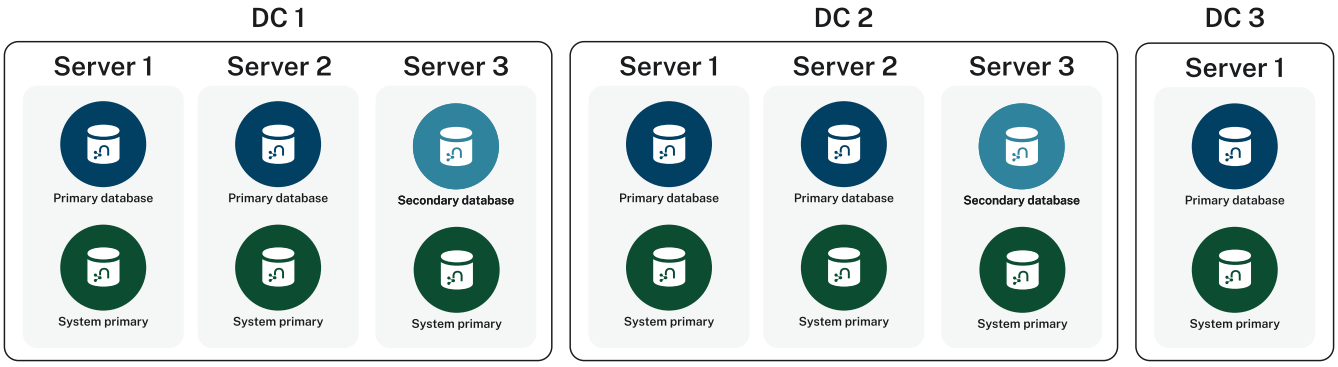


Figure 14. Five primary allocations distributed across three data centers

To tolerate the failure of any two arbitrary servers (or two primaries), the database must be configured with five primary allocations.

The configuration allows temporary removal of one primary while retaining fault tolerance, for example, when restarting one server for maintenance.

For multi-data center deployments, it is recommended to distribute five primaries across three data centers using the `2P+2P+1P` layout. You can also add secondaries, for example, to run backup jobs. See *Figure 4. Five primary allocations distributed across three data centers*. With this configuration, the cluster can tolerate the loss of any single data center.

Deployments using only two data centers (for example, `3P+2P` or `4P+1P`) or three data centers with an uneven distribution (such as `3P+1P+1P`) are not recommended. These configurations can only tolerate the loss of specific data centers rather than **any** data center.

If you possess a sufficient infrastructure, you can distribute five primaries across five data centers.

This setup allows you to lose up to two primary database allocations or any single data center. Quorum is

not lost, so the cluster keeps running.

If you lose three primaries, your database needs to be recreated since it has lost a majority of primary allocations and is therefore write-unavailable. However, the recreation can be based on the primary and secondary allocations still present on healthy servers, so a backup is not required.

Example recovery steps

1. Start and enable new servers. See How to add a server to the cluster for details.
2. Mark missing servers as not present by cordoning them. For each `Unavailable` server, run `CALL dbms.cluster.cordonServer("unavailable-server-id")` on one of the available servers.
3. For each `Cordoned` server, run `DEALLOCATE DATABASES FROM SERVER cordoned-server-id` on one of the available servers. This will move all database allocations from this server to an available server in the cluster.
4. Remove unavailable servers from the cluster:
   a. First, deallocate databases from it.
   b. Then drop the server.

   For more information, visit the Managing servers in a cluster.

For detailed scenarios, see the Disaster recovery guide.

Cluster design patterns to avoid

Two data centers with unbalanced membership

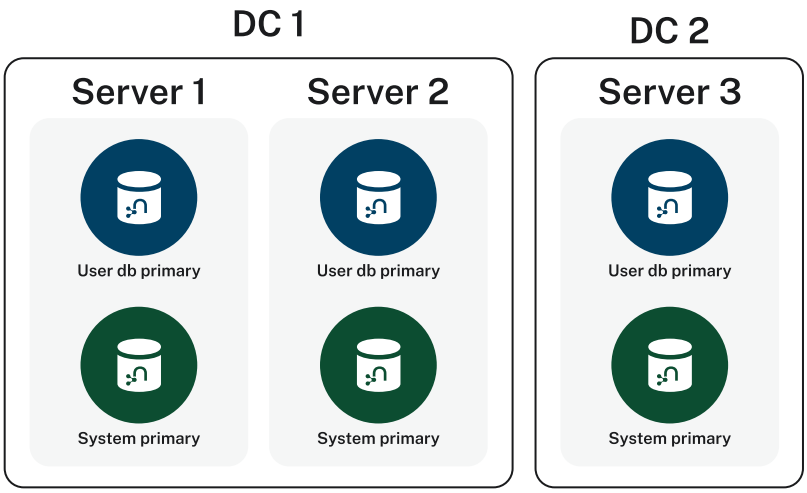


Figure 15. Unbalanced data center primary distribution

Suppose, you decide to set up just two data centers, placing two primaries in data center 1 (DC1) and one primary in the data center 2 (DC2).

If the writer primary is located in DC1, then writes can be fast because a local quorum can be reached.

This setup can tolerate the loss of one data center — but only if the failure is in DC2. If DC1 fails, you lose two primary members, which means the quorum is lost and the cluster becomes unavailable for writes.

Keep in mind that any issue could push the system back to cross–data center write latencies. Worse,

because of the latency, the member in DC2 may fall behind. In that case a failure of a member in DC1 means the database is write-unavailable until the DC2 member has caught up.

If leadership shifts to DC2, this makes all writes slow.

Finally, there is no guarantee against data loss if DC1 goes down. Because the primary member in DC2 may not be up to date with writes, even in append.

Two data centers with balanced membership

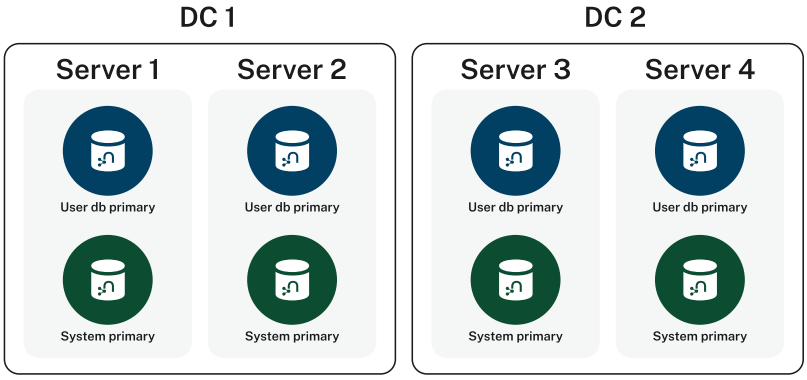


Figure 16. Symmetric primaries across two data centers

The worst scenario is to operate with just two data centers and place two or three primaries in each of them.

This means the failure of either data center leads to loss of quorum and, therefore, to loss of the cluster write-availability.

Besides, all writes have to pay the cross-data center latency cost.

This design pattern is strongly recommended to avoid.

Summary

Table 400. Comparison of cluster designs

Setup	Design	Pros	Cons	Best use case
Recommended patterns				
Secondaries for read resilience	Primaries in one data center, secondaries in other data centers	• Fast writes (local quorum). • Local reads in remote data centers.	• Loss of write availability if DC with primaries fails. • Recovery requires reseeding. • Process restarts required if DC with primaries fails.	Applications needing fast writes. The cluster can tolerate downtime during recovery.

Setup	Design	Pros	Cons	Best use case
Geo-distributed data centers (3DC)	Each primary in a different data center (≥3)	• Survives loss of one DC without data loss. • Quorum remains intact.	• Higher write latency (cross-data center).	Critical systems needing continuous availability even if a full data center fails.
Full geo-distribution for the `system` database only (3DC)	User database primaries in one DC, secondaries in another, `system` primaries across three data centers	• Fast user database writes (local). • The `system` database is always available, which means smoother recovery. • Reads available if primaries fail.	• Loss of user database writes if DC with primaries fails. • Recovery requires reseeding.	Balanced approach: fast normal operations, easier recovery, some downtime acceptable.
Tolerance to the failure of any two primaries (3DC)	Five primaries distributed across three data centers	• Tolerance to the loss up to two primaries. • With the `2P+2P+1P` layout, the cluster tolerates the loss of any single data center.	• The recommended fault tolerance depends on careful distribution of primaries; uneven or two–data center layouts reduce resilience and are not recommended. • Higher write latency (cross-data center).	Allows removal of one primary while retaining fault tolerance, e.g. restarting one server for maintenance.
Non-recommended patterns				
Two DCs – Unbalanced membership	Two primaries are in DC1, one primary is in DC2.	Fast writes if a leader is in DC1.	• Quorum lost if DC1 fails. • Risk of data loss. • Cross-DC latency if leader is in DC2.	Should be avoided.
Two DCs – Balanced membership	Equal primaries in two DCs.	(none significant)	• Quorum lost if either DC fails. • All writes pay cross-DC latency.	Should be avoided.

Multi-data center routing

This section describes the following:

Introduction

When deploying a multi-data center cluster it is often desirable to take advantage of locality to reduce latency and improve performance. For example, it is preferrable that graph-intensive workloads are executed in the local data center at LAN latencies rather than in a faraway data center at WAN latencies. Neo4j's *load balancing* and *catch-up strategy plugins* for multi-data center scenarios facilitates precisely this.

Neo4j's load balancing is a cooperative system where the driver asks the cluster on a recurring basis where it should direct the different classes of its workload (e.g., writes and reads). This allows the driver to work independently for long stretches of time, yet check back from time to time to adapt to changes such as a new server having been added for increased capacity. There are also failure situations where the driver asks again immediately, when it cannot use any of its allocated servers for example.

This is mostly transparent from the perspective of a client. On the server side, the load balancing behaviors are configured using a simple Domain Specific Language, *DSL*, and exposed under a named *load balancing policy* which the driver can bind to. All server-side configuration is performed on the Primary servers.

Catch-up strategy plugins are sets of rules that define how secondary servers contact upstream servers in the cluster in order to synchronize transaction logs. Neo4j comes with a set of pre-defined strategies, and also user-defined strategies can be created using the same DSL. Finally, Neo4j supports an API which advanced users may use to enhance upstream recommendations.

Once a catch-up strategy plugin resolves a satisfactory upstream server, it is used for pulling transactions to update the local secondary for a single synchronization. For subsequent updates, the procedure is repeated so that the most preferred available upstream server is always resolved.

Prerequisite configuration

Server tags

Both load balancing across multiple data centers and user-defined catch-up strategies are predicated on the *Server Tag* concept.

In order to optimize the use of the cluster's servers according to the specific requirements, they are sorted using *Server Tags*. Server tags can map to data centers, availability zones, or any other significant topological elements from the operator's domain, e.g., `us`, `us-east`. Applying the same tag to multiple

servers logically groups them together. Note that servers can have mulitple tags.

Server tags are defined as a key that maps onto a set of servers in a cluster. Server tags are defined on each server using the `initial.server.tags` parameter in *neo4j.conf*. Each server in a cluster can be tagged with to zero or more server tags. Server tags can be altered at runtime via the `ALTER SERVER` command, see Altering server options for more details.

Example 74. Definition of grouping servers using server tags

Grouping servers using server tags is achieved in `neo4j.conf` as in the following examples:

Tag the current instance with `us` *and* `us-east`

```
initial.server.tags=us,us-east
```

Tag the current instance with `london`

```
initial.server.tags=london
```

Tag the current instance with `eu`

```
initial.server.tags=eu
```

Note that membership of a group implied by a server tag is explicit. For example, a server tagged with `gb-london` is not automatically part of the same tag group as a server that is tagged with `gb` or `eu` unless that server is also explicitly tagged with those tags.

Primaries for reading

Depending on the deployment and the available number of servers in the cluster, different strategies make sense for whether or not the reading workload should be routed to the primary servers. The following configuration allows the routing of read workload to primary servers. Valid values are `true` and `false`.

```
dbms.routing.reads_on_primaries_enabled=true
```

The load balancing framework

There are different topology-aware load balancing options available for client applications in a multi-data center Neo4j deployment. There are different ways to configure the load balancing for the cluster so that client applications can direct its workload at the most appropriate cluster members, such as those nearby.

The load balancing system is based on a plugin architecture for future extensibility and for allowing user customizations. The current version ships with exactly one such canned plugin called the *server policies* plugin.

The server policies plugin is selected by setting the following property:

```
dbms.routing.load_balancing.plugin=server_policies
```

Under the server policies plugin, a number of load balancing policies can be configured server-side and be exposed to drivers under unique names. The drivers, in turn, must on instantiation select an appropriate policy by specifying its name. Common patterns for naming policies are after geographical regions or intended application groups.

Important:

It is crucial to define the exact same policies on all servers since this is to be regarded as cluster-wide configuration and failure to do so leads to unpredictable behavior. Similarly, policies in active use should not be removed or renamed since it breaks applications trying to use these policies. It is perfectly acceptable and expected however, that policies be modified under the same name.

If a driver asks for a policy name that is not available, then the driver is not able to use the cluster. A driver that does not specify any name at all gets the behavior of the default policy as configured. The default policy, if left unchanged, distributes the load across all servers. It is possible to change the default policy to any behavior that a named policy can have.

A misconfigured driver or load balancing policy results in suboptimal routing choices and can even prevent successful interactions with the cluster entirely.

Note:

The details of how to write a custom plugin are not documented here. Please contact Neo4j Professional Services if you think that you need a custom plugin.

Note:

Use load balancing from Neo4j drivers

Once enabled and configured, the custom load balancing feature is used by drivers to route traffic as intended. See the Neo4j Drivers manuals for instructions on how to configure drivers to use custom load balancing.

Policy definitions

The configuration of load balancing policies is transparent to client applications and expressed via a simple DSL. The syntax consists of a set of rules which are considered in order. The first rule to produce a non-empty result is the final result.

```
rule1; rule2; rule3
```

Each rule in turn consists of a set of filters which limit the considered servers, starting with the complete set. Note that the evaluation of each rule starts fresh with the complete set of available servers.

There is a fixed set of filters which composes a rule and they are chained together using arrows.

```
filter1 -> filter2 -> filter3
```

If there are any servers still left after the last filter then the rule evaluation has produced a result and this is returned to the driver. However, if there are no servers left then the next rule is considered. If no rule is able to produce a usable result then the driver is signalled a failure.

Policy names

The policies are configured under the namespace of the `server_policies` plugin and named as desired. You can find them in the *neo4j.conf* file.

Policy names can contain alphanumeric characters and underscores, and they are case sensitive. Below is the property key for a policy with the name `mypolicy`:

```
dbms.routing.load_balancing.config.server_policies.mypolicy=
```

The actual policy is defined in the value part using the DSL.

The `default` policy name is reserved for the default policy. It is possible to configure this policy like any other and it is used by driver clients that do not specify a policy.

Additionally, any number of policies can be created using unique policy names. The policy name can suggest a particular region or an application for which it is intended to be used.

Filters

There are four filters available for specifying rules, detailed below. The syntax is similar to a method call with parameters.

- `tags(name1, name2, …)`
  - Only servers that are tagged with any of the specified tags pass the filter.
  - The defined names must match those of the server tags.
  - Prior to 5.4 `tags()` were referred to as `groups()`, which continue to work but are now deprecated.
- `min(count)`
  - Only the minimum amount of servers are allowed to pass (or none).
  - Allows overload conditions to be managed.
- `all()`
  - No need to specify since it is implicit at the beginning of each rule.
  - Implicitly the last rule (override this behavior using halt).
- `halt()`
  - Only makes sense as the last filter in the last rule.
  - Stops the processing of any more rules.

The tags filter is essentially an OR-filter, e.g. `tags(A,B)` which passes any server in with either tag A, B or both (the union of the server tags). An AND-filter can also be created by chaining two filters as in `tags(A)`

`-> tags(B)`, which only passes servers with both tags (the intersect of the server tags).

Load balancing examples

The discussion on multi-data center clusters introduced a four region, multi-data center setup. The cardinal compass points for regions and numbered data centers within those regions were used there and the same hypothetical setup is used here as well.

Figure 17. Mapping regions and data centers onto server tags

The behavior of the load balancer is configured in the property `dbms.routing.load_balancing.config.server_policies.<policy-name>`. The specified rules allows for fine-tuning how the cluster routes requests under load.

The examples make use of the line continuation character `\` for better readability. It is valid syntax in *neo4j.conf* as well and it is recommended to break up complicated rule definitions using this and a new rule on every line.

The most restrictive strategy is to insist on a particular data center to the exclusion of all others:

Example 75. Specific data center only

```
dbms.routing.load_balancing.config.server_policies.north1_only=\
tags(north1)->min(2); halt();
```

This case states that the intention is to send queries to servers tagged with `north1`, which maps onto a specific physical data center, provided there are two of them available. If at least two servers tagged with `north1` cannot be provided, then the operation should `halt()`, i.e. not try any other data center.

While the previous example demonstrates the basic form of load balancing rules, it is possible to be a little more expansive:

Example 76. Specific data center preferably

```
dbms.routing.load_balancing.config.server_policies.north1=\
tags(north1)->min(2);
```

In this case if at least two servers are tagged with `north1` then the load is balanced across them. Otherwise, any server in the whole cluster is used, falling back to the implicit, final `all()` rule.

The previous example considered only a single data center before resorting to the whole cluster. If there is a hierarchy or region concept exposed through the server groups, the fall back can be more graceful:

Example 77. Gracefully falling back to neighbors

```
dbms.routing.load_balancing.config.server_policies.north_app1=\
tags(north1,north2)->min(2);\
tags(north);\
all();
```

This example says that the cluster should load balance across servers with the `north1` and `north2` tags provided there are at least two machines available across them. Failing that, any server in the `north` region can be used, and if the whole of the north is offline, any server in the cluster can be used.

Catch-up strategy plugins

Catch-up strategy plugins are sets of rules that define how secondaries contact upstream servers in the cluster in order to synchronize transaction logs. Neo4j comes with a set of pre-defined strategies, and also leverages the DSL to flexibly create user-defined strategies. Finally, Neo4j supports an API which advanced users may use to enhance upstream server recommendations.

Once a catch-up strategy plugin resolves a satisfactory upstream server, it is used for pulling transactions to update the local secondary for a single synchronization. For subsequent updates, the procedure is repeated so that the most preferred available upstream server is always resolved.

Configuring upstream selection strategy using pre-defined catch-up strategies

Neo4j ships with the following pre-defined catch-up strategy plugins. These provide coarse-grained algorithms for selecting an upstream server:

Plugin name	Resulting behavior
`connect-to-random-primary-server`	Connect to any **primary server** selecting at random from those currently available.
`typically-connect-to-random-secondary`	Connect to any available **secondary server**, but around 10% of the time connect to any random primary server.

Plugin name	Resulting behavior
`connect-randomly-to-server-tags`	Introduced in 5.5 Connect at random to any available **secondary server** tagged with any of the server tags specified in the comma-separated list `server.cluster.catchup.connect_randomly_to_server_tags` introduced in Neo4j 5.5.
`leader-only`	Connect only to the current Raft leader of the **primary servers**.
`connect-randomly-to-server-group`	Deprecated in 5.5 Connect at random to any available **secondary server** in the server groups specified in the comma-separated list `server.cluster.catchup.connect_randomly_to_server_group`. Replaced by `connect-randomly-to-server-tags` and server.cluster.catchup.connect_randomly_to_server_tags.
`connect-randomly-within-server-group`	Deprecated in 5.5 Connect at random to any available **secondary server** in any of the server groups to which this server belongs. Replaced by `connect-randomly-within-server-tags`.

Pre-defined strategies are used by configuring the `server.cluster.catchup.upstream_strategy` option. Doing so allows for specification of an ordered preference of strategies to resolve an upstream provider of transaction data. A comma-separated list of strategy plugin names with preferred strategies is provided earlier in that list. The catch-up strategy is selected by asking each of the strategies in list-order whether they can provide an upstream server from which transactions can be pulled.

Example 78. Define an upstream server selection strategy

Consider the following configuration example:

```
server.cluster.catchup.upstream_strategy=connect-randomly-to-server-tags,typically-connect-to-random-secondary
```

With this configuration the secondary server first tries to connect to any other server with tag(s) specified in `server.cluster.catchup.connect_randomly_to_server_tags`. Should it fail to find any live servers with those tags, then it connects to a random secondary server.

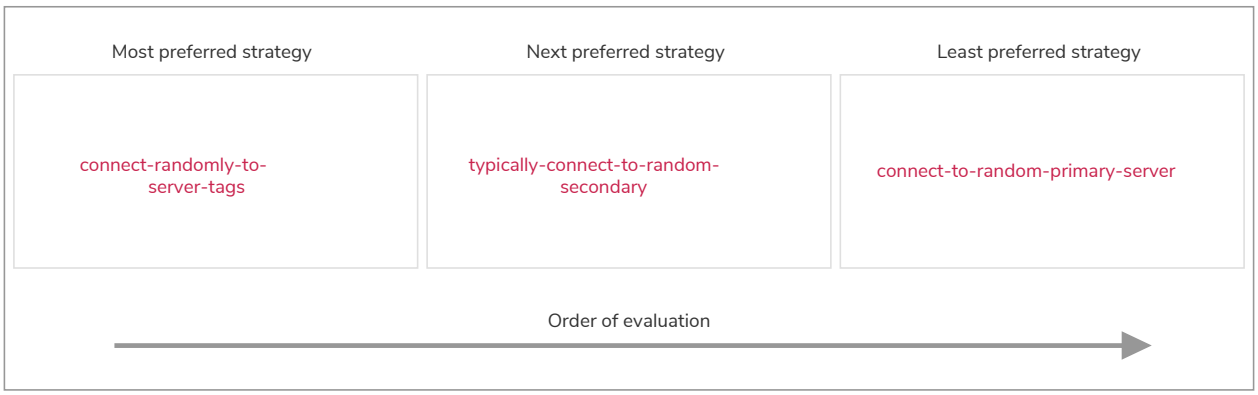


Figure 18. The first satisfactory response from a strategy will be used.

To ensure that downstream servers can still access live data in the event of upstream failures, the last resort of any server is always to contact a random primary server. This is equivalent to ending the `server.cluster.catchup.upstream_strategy` configuration with `connect-to-random-primary-server`.

Configuring user-defined catch-up strategies

Neo4j clusters support a small DSL for the configuration of client-cluster load balancing. This is described in detail in Domain Specific Language and Filters. The same DSL is used to describe preferences for how a server binds to another server to request transaction updates.

The DSL is made available by selecting the `user-defined` catch-up strategy as follows:

```
server.cluster.catchup.upstream_strategy=user-defined
```

Once the user-defined strategy has been specified, you can add configuration to the `server.cluster.catchup.user_defined_upstream_strategy` setting based on the server tags that have been set for the cluster.

This functionality is described with two examples:

Example 79. Defining a user-defined strategy

For illustrative purposes four regions are proposed: `north`, `south`, `east`, and `west` and within each region there is a number of data centers such as `north1` or `west2`. The server tags are configured so that each data center maps to its own server tag. Additionally it is assumed that each data center fails independently from the others and that a region can act as a supergroup of its constituent data centers. So a server in the `north` region might have configuration like `initial.server.tags=north2,north` which puts it in two groups that match to your physical topology as shown in the diagram below.

Figure 19. Mapping regions and data centers onto server tags

Once the servers are tagged, the next task is to define some upstream selection rules based on them. For design purposes, assume that any server in one of the `north` region data centers prefers to catch-up within the data center if it can, but resorts to any northern instance otherwise. To configure that behavior, add:

```
server.cluster.catchup.user_defined_upstream_strategy=tags(north2); tags(north); halt()
```

The configuration is in precedence order from left to right. The `tags()` operator yields a server tag from which to catchup. In this case, only if there are no servers tagged with `north2` does the operation proceed to the `tags(north)` rule which yields any server tagged with `north`. Finally, if no servers can be resolved with any of the previous tags, then the rule chain is stopped via `halt()`.

Note that the use of `halt()` ends the rule chain explicitly. If a `halt()` is not used at the end of the rule chain, then the `all()` rule is implicitly added. `all()` is expansive: it offers up all servers and so increases the likelihood of finding an available upstream server. However `all()` is indiscriminate and the servers it offers are not guaranteed to be topologically or geographically local, potentially increasing the latency of synchronization.

The example above shows a simple hierarchy of preferences expressed through the use of server tags. But the hierarchy can be more sophisticated. For example, conditions can be placed on the tagged catch-up servers.

Example 80. User-defined strategy with conditions

In this example it is desired to roughly qualify cluster health before selecting from where to catchup. For this, the `min()` filter is used as follows:

```
server.cluster.catchup.user_defined_upstream_strategy=tags(north2)->min(3), tags(north)->min(3);
all();
```

`tags(north2)->min(3)` states that catch-up from servers tagged with `north2` should be performed

only if there are three available servers, which here is interpreted as an indicator of good health. If `north2` can't meet that requirement then catch-up should be attempted from any server tagged with `north` provided there are at least three of them available as per `tags(north)->min(3)`. Finally, if catch-up cannot be performed from a sufficiently healthy `north` region, then the operation (explicitly) falls back to the whole cluster with `all()`.

The `min()` filter is a simple but reasonable health indicator of a set of servers with the same tag.

Favoring data centers

In a multi-data center scenario, while it remains a rare occurrence, it is possible to bias where writes for the specified database should be directed. `db.cluster.raft.leader_transfer.priority_tag` can be applied to specify a set of servers with a given tag which should have priority when selecting the leader for a given database. The priority tag can be set on one or multiple databases and it means that the cluster attempts to keep the leadership for the configured database on a server tagged with the configured server tag.

A database for which `db.cluster.raft.leader_transfer.priority_tag` has been configured is excluded from the automatic balancing of leaderships across a cluster. It is therefore recommended to not use this configuration unless it is necessary.

Disaster recovery

A database can become unavailable due to issues on different system levels. For example, a data center failover may lead to the loss of multiple servers, which may cause a set of databases to become unavailable.

This section contains a step-by-step guide on how to recover **unavailable databases** that are incapable of serving writes and/or reads. The guide recovers the unavailable databases and make them fully operational, with minimal impact on the other databases in the cluster. However, if a database is not performing as expected for other reasons, this section cannot help.

Caution:

If **all** servers in a Neo4j cluster are lost in a disaster, it is not possible to recover the current cluster. You have to create a new cluster and restore the databases, see Deploy a basic cluster and Seed a database for more information.

Faults in clusters

Databases in clusters may be allocated differently within the cluster and may also have different numbers of primaries and secondaries.

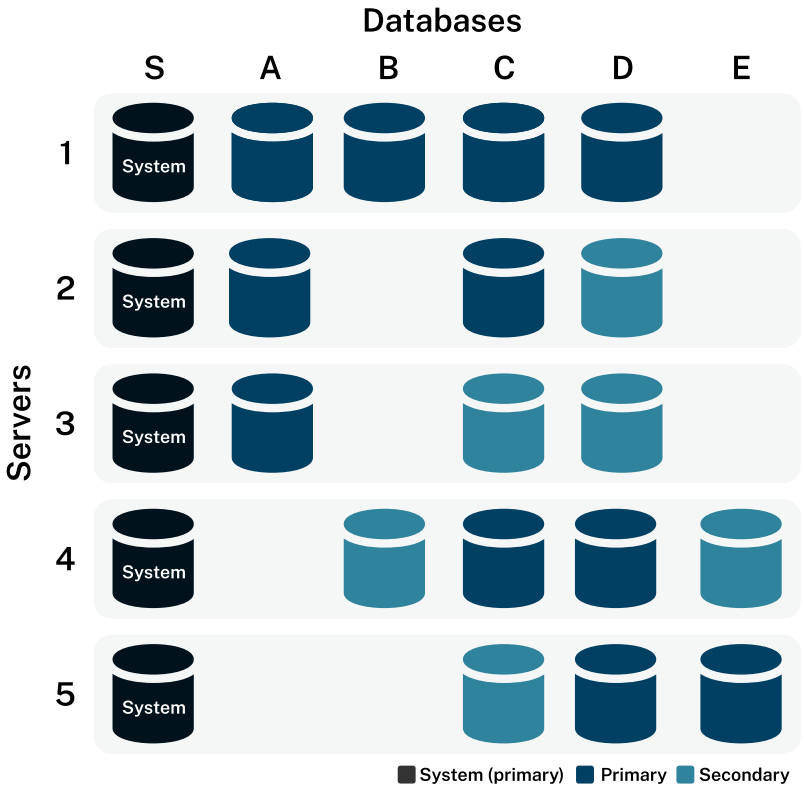


Figure 20. A healthy cluster

The consequence of this is that all servers may be different in which databases they are hosting. Losing a server in a cluster may cause some databases to lose a member while others are unaffected. Therefore, in a disaster where one or more servers go down, some databases may keep running with little to no impact, while others may lose all their allocated resources.

Figure 2 shows the disaster when three servers are lost, demonstrating that this situation impacts databases in different ways.

Figure 21. Example of a cluster disaster

Table 401. Disaster scenarios and recovery strategies

Database	Disaster scenario	Recovery strategy
Database A	All allocations are lost.	The database needs to be recreated from a backup since there are no available allocations left in the cluster.
Database B	The primary allocation is lost, and the secondary allocation is available.	The database needs to be recreated since it has lost a majority of primary allocations and is therefore write-unvailable. However, the recreation can be based on the secondary allocation still present on a healthy server, so a backup is not required. The recreated database will be as up-to-date as the secondary allocation was at the time of the disaster.
Database C	Two primary allocations and a secondary one are lost.	The database needs to be recreated since it has lost a majority of primary allocations and is therefore write-unavailable. However, the recreation can be based on the primary and secondary allocations still present on healthy servers, so a backup is not required. The recreated database will reflect the state of the most up-to-date surviving primary or secondary allocation.

Database	Disaster scenario	Recovery strategy
Database D	One primary allocation and two secondary allocations are lost.	The database remains write-available, allowing it to automatically move allocations from lost servers to available ones when the lost servers are deallocated. Therefore, the database does not need to be recreated even though some allocations have been lost.
Database E	Stays unaffected.	None of the database's allocations were affected by the disaster, so no action is required.

Although databases C and D share the same topology, their primaries and secondaries are allocated differently, requiring distinct recovery strategies in this disaster example.

Guide overview

Note:

In this guide the following terms are used:

- An *offline* server is a server that is not running but may be restartable.
- A *lost* server, however, is a server that is currently not running and cannot be restarted.
- A *write-available* database is able to serve writes, while a *write-unavailable* database is not.

There are four steps to recovering a cluster from a disaster:

1. Start the Neo4j process on all servers which are not *lost*. See Start the Neo4j process for more information.
2. Make the `system` database able to serve write operations, so that the cluster can be modified. See Make the `system` database write-available for more information.
3. Detach any potential lost servers from the cluster and replace them by new ones. See Make servers available for more information.
4. Finish disaster recovery by starting or continuing to manage databases and verify that they are write-available. See Make databases write-available for more information.

Each step is described in the following three sections:

1. Objective — a state that the cluster needs to be in, with optional motivation.
2. Verifying the state — an example of how the state can be verified.
3. Path to correct state — a proposed series of steps to get to the correct state.

Caution:

Verifying each state before continuing to the next step, regardless of the disaster scenario, is recommended to ensure the cluster is fully operational.

Disaster recovery steps

Note:

Disasters may sometimes affect the routing capabilities of the driver and may prevent the use of the `neo4j` scheme for routing. One way to remedy this is to connect directly to the server using `bolt` instead of `neo4j`. See Server-side routing for more information on the `bolt` scheme.

Start the Neo4j process

Objective

The Neo4j process is started on all servers that are not *lost*.

Path to correct state

Start the Neo4j process on all servers that are *offline*. If a server is unable to start, inspect the logs and contact support personnel. The server may have to be considered indefinitely lost.

Make the `system` database write-available

Objective

The `system` database is able to serve write operations.

The `system` database contains the view of the cluster. This includes which servers and databases are present, where they live and how they are configured. During a disaster, the view of the cluster might need to change to reflect a new reality, such as removing lost servers. Databases might also need to be recreated to regain write availability. Because both of these steps are executed by modifying the `system` database, making the `system` database write-available is a vital first step during disaster recovery.

Verifying the state

The `system` database's write availability can be verified by using the Status check procedure.

```
CALL dbms.cluster.statusCheck(["system"]);
```

The status check procedure cannot verify the write availability of a database configured to have a single primary. Instead, check that the primary is allocated on an available server and that it has `currentStatus` = `online` by running `SHOW DATABASES`.

Path to correct state

Use the following steps to regain write availability for the `system` database if it has been lost. They create a new `system` database from the most up-to-date copy of the `system` database that can be found in the cluster. It is important to get a `system` database that is as up-to-date as possible, so it corresponds to the view before the disaster closely.

> This section of the disaster recovery guide uses `neo4j-admin` commands. For more information about the used commands, see neo4j-admin commands.

1. Shut down the Neo4j process on all servers. This causes downtime for all databases in the cluster until the processes are started again at the end of this section.
2. On each server, run `bin/neo4j-admin dbms unbind-system-db` to reset the `system` database state on the servers.
3. On each server, run `bin/neo4j-admin database info system` and compare the `lastCommittedTransaction` to find out which server has the most up-to-date copy of the `system` database.
4. On the most up-to-date server, run `bin/neo4j-admin database dump system --to-path=[path-to-dump]` to take a dump of the current `system` database and store it in an accessible location.
5. For every *lost* server, add a new **unconstrained** one according to Add a server to the cluster. It is important that the new servers are unconstrained, or deallocating servers in the next step of this guide might be blocked, even though enough servers were added.

   In the current example, the new unconstrained servers are added in this step.

   > While recommended, it is not strictly necessary to add new servers in this step. There is also an option to change the `system` database mode (`server.cluster.system_database_mode`) on secondary allocations to make them primary allocations for the new `system` database. The number of primary allocations needed is defined by `dbms.cluster.minimum_initial_system_primaries_count`. See the Configuration settings for more information. Be aware that not replacing servers can cause cluster overload when databases are moved from lost servers to available ones in the next step of this guide.

6. On each server, run `bin/neo4j-admin database load system --from-path=[path-to-dump] --overwrite-destination=true` to load the current `system` database dump.

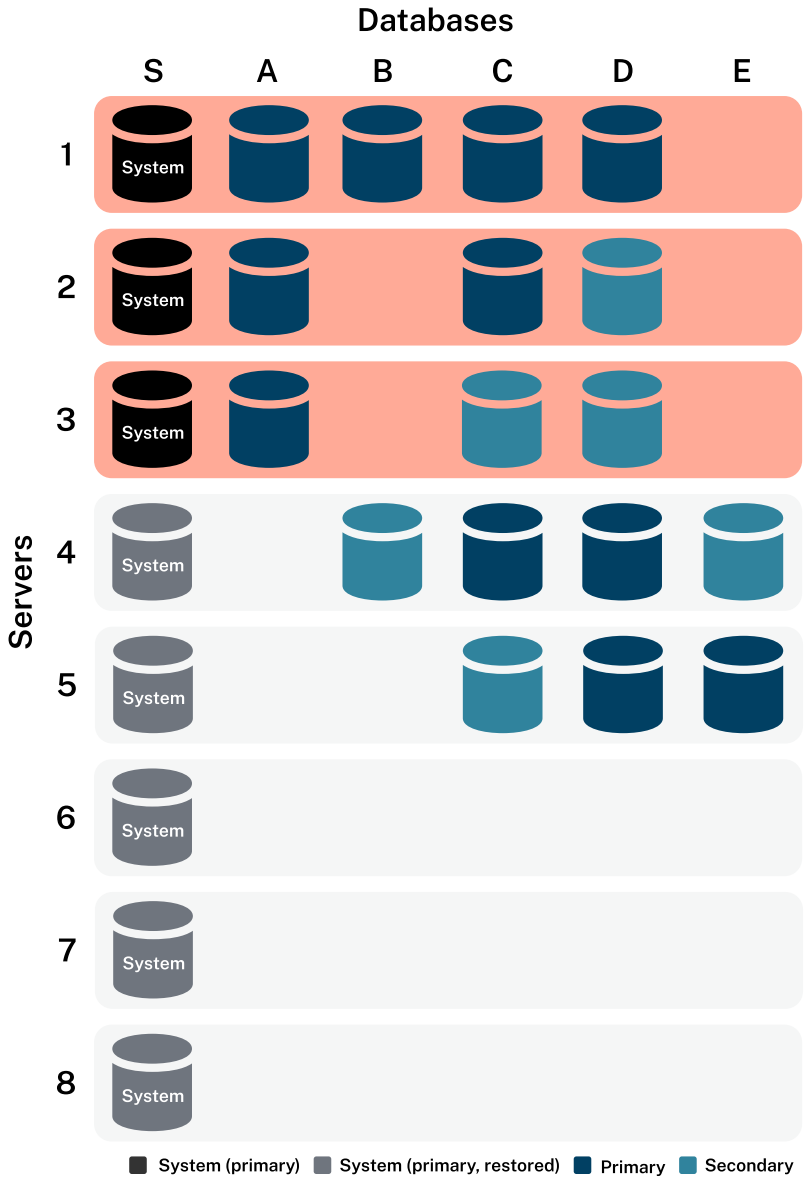


Figure 22. The unconstrained servers are added and the `system` database is restored

7. On each server, ensure that the discovery settings are correct. See Cluster server discovery for more information.
8. Start the Neo4j process on all servers.

Make servers available

Objective

> All servers in the cluster's view are available and enabled.

A lost server will still be in the `system` database's view of the cluster, but in an unavailable state. Furthermore, according to the view of the cluster, these lost servers are still hosting the databases they had before they became lost. Therefore, informing the cluster of servers which are lost is not enough. The databases hosted on lost servers also need to be moved onto available servers in the cluster, before the lost servers can be removed.

Verifying the state

The cluster's view of servers can be seen by listing the servers. See Listing servers for more information. The state has been verified if **all** servers show `health` = `Available` and `status` = `Enabled`.

```
SHOW SERVERS;
```

Path to correct state

Use the following steps to remove lost servers and add new ones to the cluster. To remove lost servers, any allocations they were hosting must be moved to available servers in the cluster. This is done in two different steps:

- Any allocations that cannot move by themselves require the database to be recreated so that they are forced to move.
- Any allocations that can move will be instructed to do so by deallocating the server.

1. For each `Unavailable` server, run `CALL dbms.cluster.cordonServer("unavailable-server-id")` on one of the available servers. This prevents new database allocations from being moved to this server.

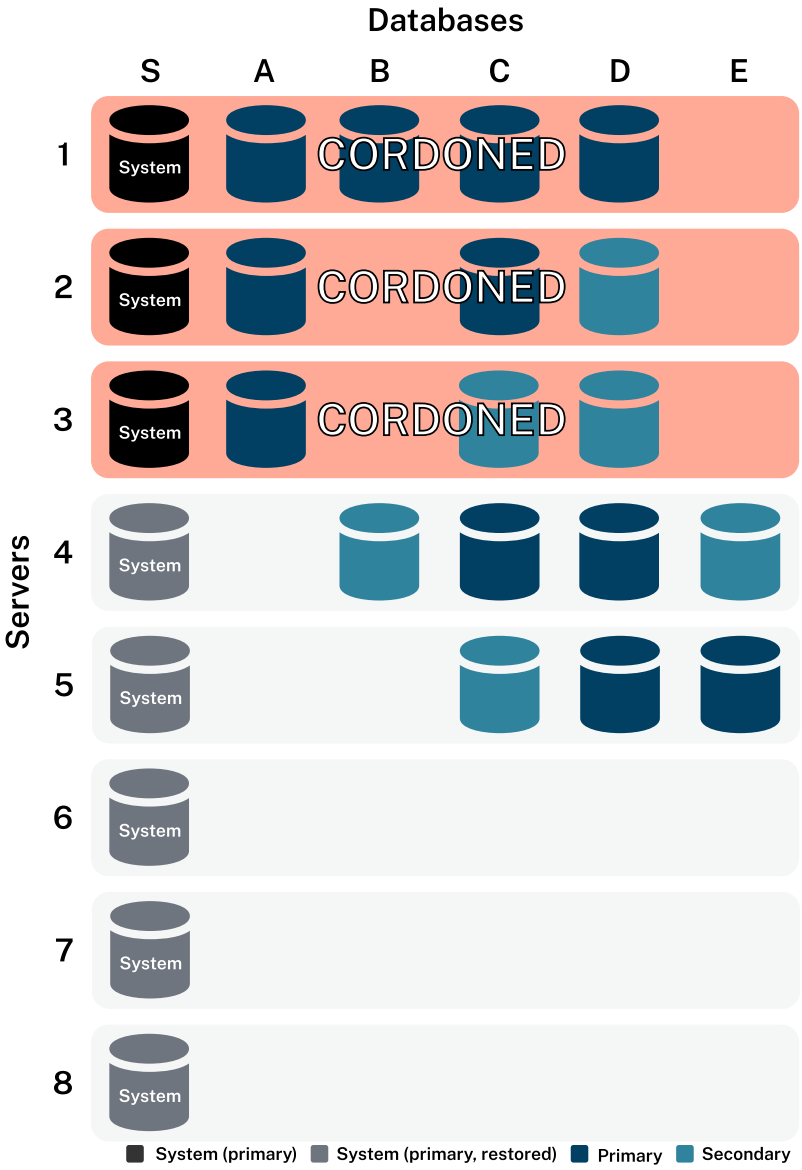


Figure 23. Cordon unavailable servers

Figure 4 shows that new unconstrained servers have been added already. It was done in the Make the `system` database write-available step of this guide, and additional servers might not be needed here.

2. If you have not yet added new **unconstrained** servers, add one for each `Cordoned` server that needs to be replaced. See Add a server to the cluster for more information. It is important that the new servers are unconstrained, or deallocating servers might be blocked even though enough servers were added.

> While recommended, it is not strictly necessary to add new servers in this step. However, not adding new servers reduces the capacity of the cluster to handle work. Furthermore, it might require the topology for a database to be altered to make deallocating servers and recreating databases possible.

3. For each stopped database (`currentStatus` = `offline`), start them by running `START DATABASE stopped-db`. This is necessary since stopped databases cannot be deallocated from a server. It is also necessary for the status check procedure to accurately indicate if this database should be recreated or not. Verify that all allocations are in `currentStatus` = `online` on servers which are not

lost before moving to the next step. If a database fails to start, leave it to be recreated in the next step of this guide.

> A database can be set to `READ-ONLY` before it is started to avoid updates on the database with the following command: `ALTER DATABASE database-name SET ACCESS READ ONLY`.

4. On each server, run `CALL dbms.cluster.statusCheck([])` to check the write availability for all databases running in primary mode on this server. See Monitoring replication for more information.

> The status check procedure cannot verify the write availability of a database configured to have a single primary. Instead, check that the primary is allocated on an available server and that it has `currentStatus` = `online` by running `SHOW DATABASES`.

5. For each database that is not write-available, recreate it to move it from lost servers and regain write availability. Go to Recreate a database for more information about recreate options. Remember to make sure there are recent backups for the databases before recreating them. See Online backup for more information. If any database has `currentStatus` = `quarantined` on an available server, recreate them from backup using Backup as seed.

> If you recreate databases using undefined servers or undefined servers with fallback backup, the store might not be recreated as up-to-date as possible in certain edge cases where the `system` database has been restored.

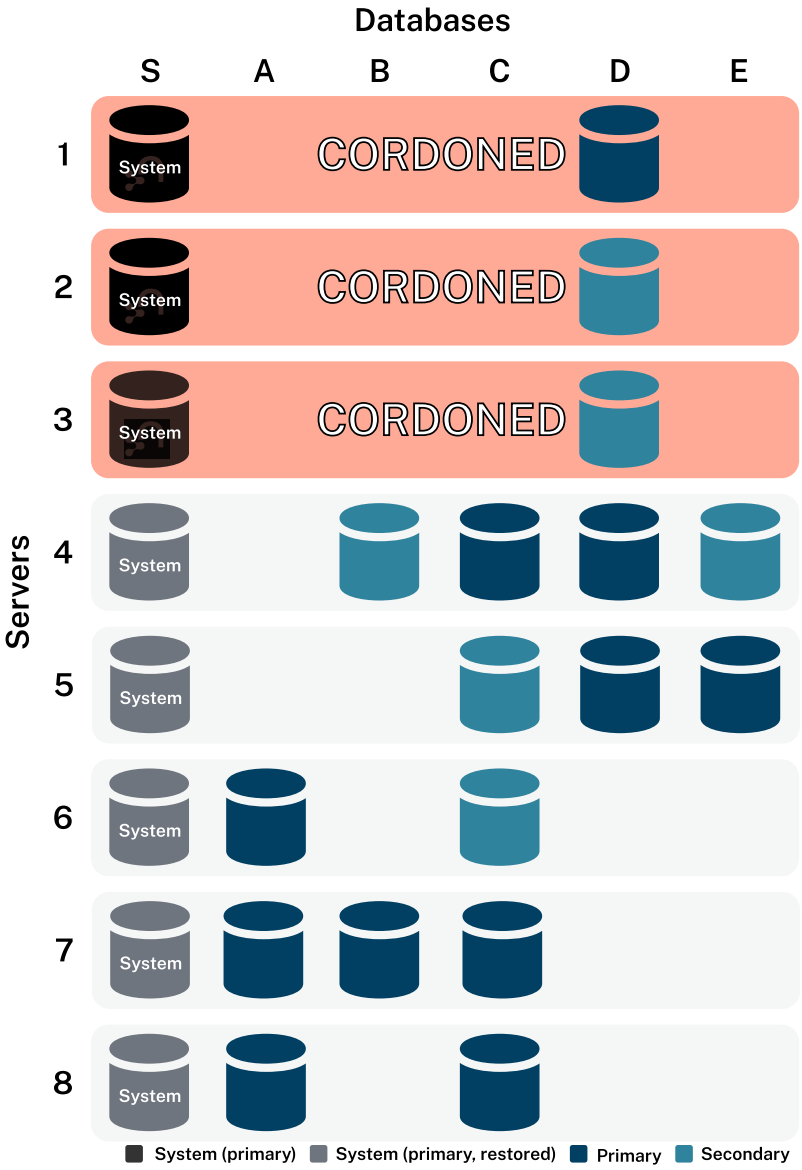


Figure 24. All write-unavailable databases were recreated

6. For each `Cordoned` server, run `DEALLOCATE DATABASES FROM SERVER cordoned-server-id` on one of the available servers. This will move all database allocations from this server to an available server in the cluster.

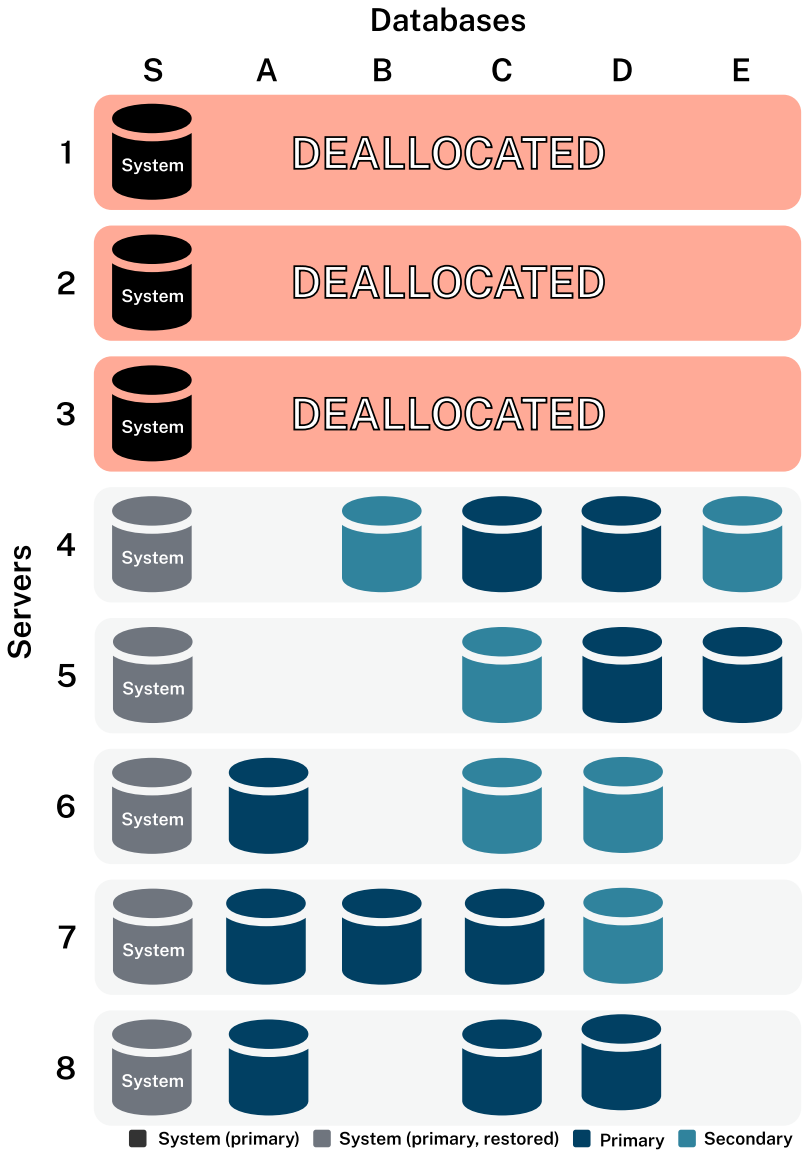


Figure 25. Deallocate databases from unavailable servers

Note that the database D was still write-available, which means the allocations can be moved from lost servers to available ones when the lost servers are deallocated.

> This operation might fail if enough unconstrained servers were not added to the cluster to replace lost servers. Another reason is that some available servers are also `Cordoned`.

7. For each deallocating or deallocated server, run `DROP SERVER deallocated-server-id`. This removes the server from the cluster's view.

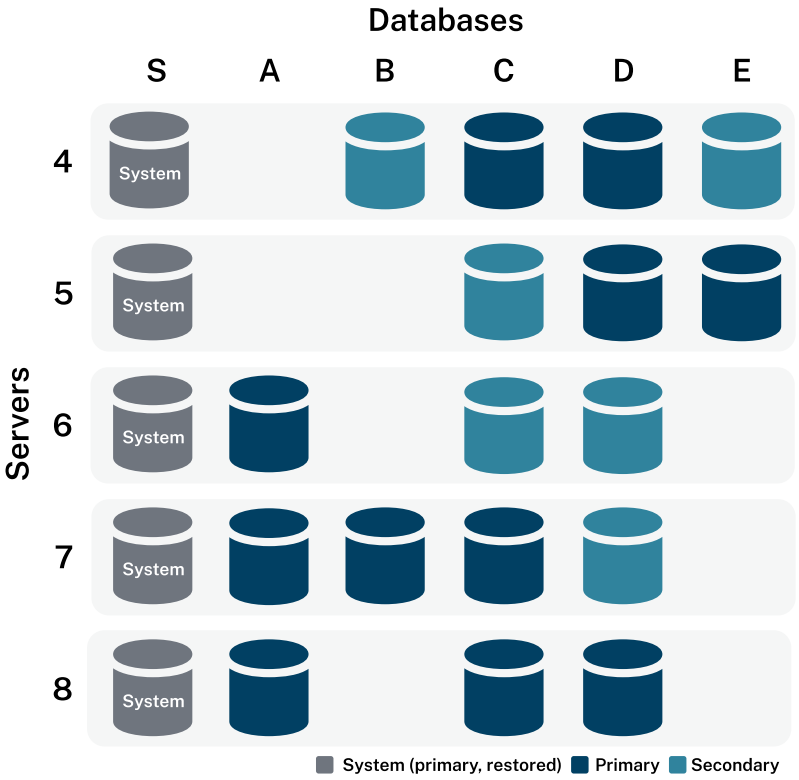


Figure 26. The fully recovered cluster

After dropping the deallocated servers, you still have to ensure that all moved and recreated databases are write-available. For this purpose, follow the steps below.

Make databases write-available

Objective

All databases that are desired to be started are write-available.

Once this state is verified, disaster recovery is complete. However, remember that previously stopped databases might have been started during this process. If they are still desired to be in stopped state, run `STOP DATABASE started-db WAIT`.

> **Caution:**
>
> Remember, recreating a database takes an unbounded amount of time since it may involve copying the store to a new server, as described in Recreate a database. Therefore, an allocation with `currentStatus` = `starting` will probably reach the `requestedStatus` given some time.

Verifying the state

You can verify all clustered databases' write availability by using the status check procedure.

```
CALL dbms.cluster.statusCheck([]);
```

> The status check procedure cannot verify the write availability of a database configured to have a single primary. Instead, check that the primary is allocated on an available server and that it has `currentStatus` = `online` by running `SHOW DATABASES`.

A stricter verification can be done to verify that all databases are in their desired states on all servers. For the stricter check, run `SHOW DATABASES` and verify that `requestedStatus` = `currentStatus` for all database allocations on all servers.

Path to correct state

Use the following steps to make all databases in the cluster write-available again. They include recreating any databases that are not write-available and identifying any recreations that will not complete. Recreations might fail for different reasons, but one example is that the checksums do not match for the same transaction on different servers.

1. Identify all write-unavailable databases by running `CALL dbms.cluster.statusCheck([])` as described in the Example verification part of this disaster recovery step. Filter out all databases desired to be stopped, so that they are not recreated unnecessarily.
2. Recreate every database that is not write-available and has not been recreated previously. See Recreate a database for more information. Remember to make sure there are recent backups for the databases before recreating them. See Online backup for more information. If any database has `currentStatus` = `quarantined` on an available server, recreate them from backup using Backup as a seed.

   > If you recreate databases using undefined servers or undefined servers with fallback backup, the store might not be recreated as up-to-date as possible in certain edge cases where the `system` database has been restored.

3. Run `SHOW DATABASES` and check any recreated databases that are not write-available. Recreating a database will not complete if one of the following messages is displayed in the message field:
   - `Seeders ServerId1 and ServerId2 have different checksums for transaction TransactionId. All seeders must have the same checksum for the same append index.`
   - `Seeders ServerId1 and ServerId2 have incompatible storeIds. All seeders must have compatible storeIds.`
   - `No store found on any of the seeders ServerId1, ServerId2…`
4. For each database which will not complete recreation, recreate them from backup using Backup as a seed.

Settings reference

Parameter	Explanation
`initial.server.mode_constraint`	This setting constrains the operating mode of the database to be used only in primary or secondary mode. Default setting is `NONE`, ie. no constraint. As an initial setting, the value set here is used when a server is first enabled. Once enabled, a server's mode constraint can only be changed with `ALTER SERVER 'name' SET OPTIONS {modeConstraint:'PRIMARY'}`. **Example**: a server configured with `initial.server.mode_constraint=SECONDARY` is only allocated databases whose topologies contain 1 or more secondary. This server always only hosts those databases in `SECONDARY` mode.
`initial.dbms.automatically_enable_free_servers`	This setting allows for auto-enable of servers in the `FREE` state. After startup, it can be changed with the `dbms.cluster.setAutomaticallyEnableFreeServers` procedure.
`server.cluster.system_database_mode`	Every cluster member hosts the `system` database. This config controls what mode a given instance hosts the `system` database in: `PRIMARY` or `SECONDARY`. **Example:** `server.cluster.system_database_mode=SECONDARY` means that this instance holds only a secondary copy of the `system` database. **Note:** There should be a relatively high number (5-7) of `system` primaries, spread across availability zones. However, if enabling more than 10 servers, it is recommended to start making the later ones secondaries.
`dbms.cluster.minimum_initial_system_primaries_count`	Minimum number of servers configured with `server.cluster.system_database_mode=PRIMARY` required to form a cluster. **Example:** `dbms.cluster.minimum_initial_system_primaries_count=3` specifies that the cluster is considered bootstrapped and the DBMS online when at least 3 `system` database primaries have discovered one another.

Parameter	Explanation

`dbms.cluster.discovery.resolver_type`	This setting specifies the strategy that the instance uses to determine the addresses for other instances in the cluster to contact for *bootstrapping*. Possible values are:
	
	`LIST`
	Treats `dbms.cluster.discovery.endpoints` (or `dbms.cluster.discovery.v2.endpoints`, if you use discovery service v2 available as of Neo4j 5.23) as a list of addresses of servers to contact for discovery.
	
	`DNS`
	Treats `dbms.cluster.discovery.endpoints` (or `dbms.cluster.discovery.v2.endpoints`, if you use discovery service v2 available as of Neo4j 5.23) as a domain name to resolve via DNS. Expect DNS resolution to provide A records with hostnames or IP addresses of servers to contact for discovery, on the port specified by `dbms.cluster.discovery.endpoints`.
	
	`SRV`
	Treats `dbms.cluster.discovery.endpoints` (or `dbms.cluster.discovery.v2.endpoints`, if you use discovery service v2 available as of Neo4j 5.23) as a domain name to resolve via DNS. Expect DNS resolution to provide SRV records with hostnames or IP addresses and ports, of servers to contact for discovery.
	
	`K8S`
	Accesses the Kubernetes list service API to derive addresses of servers to contact for discovery. Requires `dbms.kubernetes.label_selector` to be a Kubernetes label selector for Kubernetes services running a server each and `dbms.kubernetes.service_port_name` to be a service port name identifying the discovery port of cluster servers services. The value of `dbms.cluster.discovery.endpoints` (or `dbms.cluster.discovery.v2.endpoints`, if you use discovery service v2 available as of Neo4j 5.23) is ignored for this option. For more details, see Discovery in Kubernetes.
	
	From Neo4j 5.23, depending on which version of the discovery service you are using, you need to set either `dbms.cluster.discovery.endpoints` or `dbms.cluster.discovery.v2.endpoints` in the *neo4j.conf* file. Detailed information about discovery and discovery configuration options is given in Methods for server discovery.

Parameter	Explanation
`dbms.cluster.discovery.endpoints` Deprecated in 5.23	One or more network addresses used to discover other servers in the cluster. The exact method by which endpoints are resolved to other cluster members is determined by the value of `dbms.cluster.discovery.resolver_type`. In the default case, the initial discovery members are given as a comma-separated list of address/port pairs, and the default port for the discovery service is `:5000`. It is good practice to set this parameter to the same value on all servers in the cluster. **Example:** `dbms.cluster.discovery.resolver_type=LIST` combined with `server01.example.com:5000,server02.example.com:5000,server03.example.com:5000` attempt to reach Neo4j instances listening on *server01.example.com*, *server02.example.com* and *server03.example.com*; all on port `5000`.
`dbms.cluster.discovery.v2.endpoints` Introduced in 5.22	A comma-separated list of endpoints that a server should contact in order to discover other cluster members. Typically, all cluster members, including the current server, must be specified in this list. The setting configures the endpoints for discovery service v2. **Example:** `dbms.cluster.discovery.resolver_type=LIST` combined with `server01.example.com:6000,server02.example.com:6000,server03.example.com:6000` attempt to reach Neo4j instances listening on *server01.example.com*, *server02.example.com* and *server03.example.com*; all on port `6000`.
`dbms.cluster.discovery.version` Introduced in 5.22	This setting allows you to select which discovery service should be started. Possible values are: • V1_ONLY -— it runs only discovery service v1. • V1_OVER_V2 -— it runs both discovery service v1 and discovery service v2, where v1 is the main service and v2 runs in the background. • V2_OVER_V1 -— it runs both discovery service v1 and discovery service v2, where v2 is the main service and v1 runs in the background. • V2_ONLY — it runs only discovery service v2. The default value is `V1_ONLY`. Discovery services v1 and v2 are designed to run in parallel. They are completely independent of each other, thus allowing you to keep the cluster functioning while switching over from v1 to v2. For details on how to move from discovery service v1 to v2, see Moving from discovery service v1 to v2.

Parameter	Explanation
`server.discovery.advertised_address` Deprecated in 5.23	The address/port setting that specifies where the instance advertises that it listens for discovery protocol messages from other members of the cluster. If this server is included in the `discovery.endpoints` of other cluster members, the value there must **exactly** match this advertised address. **Example:** `server.discovery.advertised_address=192.168.33.21:5001` indicates that other cluster members can communicate with this server using the discovery protocol at host `192.168.33.20` and port `5001`.
`server.cluster.raft.advertised_address`	The address/port setting that specifies where the Neo4j server advertises to other members of the cluster that it listens for Raft messages within the cluster. **Example:** `server.cluster.raft.advertised_address=192.168.33.20:7000` listens for cluster communication in the network interface bound to `192.168.33.20` on port `7000`.
`server.cluster.advertised_address`	The address/port setting that specifies where the instance advertises it listens for requests for transactions in the transaction-shipping catch-up protocol. **Example:** `causal_clustering.transaction_advertised_address=192.168.33.20:6001` listens for transactions from cluster members on the network interface bound to `192.168.33.20` on port `6001`.
`server.discovery.listen_address` Deprecated in 5.23	The address/port setting that specifies which network interface and port the Neo4j instance binds to for the cluster discovery protocol. **Example:** `server.discovery.listen_address=0.0.0.0:5001` listens for cluster membership communication on any network interface at port `5001`.
`server.cluster.raft.listen_address`	The address/port setting that specifies which network interface and port the Neo4j instance binds to for cluster communication. This setting must be set in coordination with the address this instance advertises it listens at in the setting `server.cluster.raft.advertised_address`. **Example:** `server.cluster.raft.listen_address=0.0.0.0:7000` listens for cluster communication on any network interface at port `7000`.
`server.cluster.listen_address`	The address/port setting that specifies which network interface and port the Neo4j instance binds to for cluster communication. This setting must be set in coordination with the address this instance advertises it listens at in the setting `server.cluster.advertised_address`. **Example:** `server.cluster.listen_address=0.0.0.0:6001` listens for cluster communication on any network interface at port `6001`.

Server management command syntax

Servers can be added and managed using a set of Cypher administrative commands executed against the `system` database.

When connected to the DBMS over `bolt`, administrative commands are automatically routed to the `system` database.

Server management command syntax

> **Note:**
>
> More details about the syntax descriptions can be found on the page Database management command syntax → Reading the administrative commands syntax.

Enable a server

Command	`ENABLE SERVER`
Syntax	`ENABLE SERVER 'serverId' [OPTIONS "{" option: value[,...] "}"]`
Description	Adds a server that has been discovered to the cluster. For more information see Enabled state.
Required privilege	`GRANT SERVER MANAGEMENT` (see SERVER MANAGEMENT privileges)

List servers

Command	`SHOW SERVERS`
Syntax	`SHOW SERVER[S] [YIELD { * | field[, ...] } [ORDER BY field[, ...]] [SKIP n] [LIMIT n]] [WHERE expression] [RETURN field[, ...] [ORDER BY field[, ...]] [SKIP n] [LIMIT n]]`
Description	Lists all servers visible to the cluster. For more information see Listing servers.
Required privilege	`GRANT SHOW SERVERS` (see SERVER MANAGEMENT privileges)

Alter server's options

Command	`ALTER SERVER`
Syntax	`ALTER SERVER 'name' SET OPTIONS "{" option: value[,...] "}"`
Description	Changes the constraints for a server. For more information see Altering server options.
Required privilege	`GRANT SERVER MANAGEMENT` (see SERVER MANAGEMENT privileges)

Rename a server

Command	`RENAME SERVER`
Syntax	`RENAME SERVER 'name' TO 'newName'`
Description	Changes the name of a server. For more information see Renaming a server.
Required privilege	`GRANT SERVER MANAGEMENT` (see SERVER MANAGEMENT privileges)

Remove a server

Command	`DROP SERVER`
Syntax	`DROP SERVER 'name'`
Description	Removes a server not hosting any databases from the cluster. For more information see Dropping a server.
Required privilege	`GRANT SERVER MANAGEMENT` (see SERVER MANAGEMENT privileges)

Reallocate databases in a cluster

Command	`REALLOCATE DATABASES`
Syntax	`[DRYRUN] REALLOCATE DATABASE[S]`
Description	Re-balances databases among the servers in the cluster. For more information see Hosting databases on added servers. Note that is a cluster-level command, not a server-level command.
Required privilege	`GRANT SERVER MANAGEMENT` (see SERVER MANAGEMENT privileges)

Deallocate databases from a server

Command	`DEALLOCATE DATABASES`
Syntax	`[DRYRUN] DEALLOCATE DATABASE[S] FROM SERVER[S] 'name'[, ...]`
Description	Removes all user databases from the given servers. For more information see Deallocating databases from server.
Required privilege	`GRANT SERVER MANAGEMENT` (see SERVER MANAGEMENT privileges)

Clustering glossary

Term	Description
Allocator	A component in the cluster that allocates databases to servers according to the topology constraints specified and an allocation strategy.
Asynchronous replication	Enables efficient scale-out of secondary database copies but offers no guarantees under fault conditions. The data present in the secondary copy is not guaranteed to be up-to-date with a majority of the database's primary copies.
Availability	The ability to access data in a database. A database can be available for read-write, read-only, or altogether unavailable. A clustered database is fault-tolerant, i.e. it can maintain both read and write availability if some primaries fail (see Fault tolerance for more information). If the number of failed primaries exceeds the fault tolerance limit, the database becomes read-only. Should all copies fail, the database becomes unavailable.
Bookmark	A marker the client can request from the cluster to ensure that it is able to read its own writes so that the application's state is consistent and only databases that have a copy of the bookmark are permitted to respond.

Term	Description
Causal consistency	When a client (driver) creates a session and executes a query, the responding server issues the client a bookmark. This reflects the state of the database copy on that server at the time the query was executed. The bookmark is passed along and updated by all subsequent queries in the session, regardless of which server executes what query. A bookmark can only be updated monotonically increasing. If a server is behind the state in the bookmark, it waits until it has caught up, or time out the query. Thus, clients executing queries within a session are guaranteed to read their own writes, and only see successively later states of the database. This is sometimes also referred to as session consistency.
Cluster	A collection of servers running Neo4j that are configured to communicate with each other. These may be used to host databases and the databases may be configured to replicate across servers in the cluster thus achieving read scalability or high availability. A minimum of three servers is required for the cluster to be fault-tolerant.
Database	The data store for the nodes, relationships, and properties that make up the graph. Multiple databases can be hosted on a Database Management Server (DBMS).
Database Management System (DBMS)	The Neo4j services and system database running on an instance of a single server or cluster to provide one or more databases.
Deallocate	An act of safely removing (i.e. without loss of data or reduced fault tolerance) a database from a server, or removing a server from a cluster.
Disaster recovery	A manual intervention to restore availability of a cluster, or databases within a cluster.

Term	Description
Election	In the event that a leader becomes unresponsive, followers automatically trigger an election and vote for a new leader. A majority is required for the vote to be successful.
Fault tolerance	A guarantee that a database can maintain persistence and availability in the event of one or more failures. The number of failures `f` that can be tolerated is dependent on the number of primaries `n` for the database and follows the formula `f = (n-1)/2`. In the event that more than `f` primaries fail, the database can no longer process write transactions and becomes read-only.
Follower	A primary copy of a database acting as a follower, receives and acknowledges synchronous writes from the leader.
Leader	A single primary copy of a database is designated as the leader. It receives all write transactions from clients and replicates writes synchronously to followers and asynchronously to secondary copies of the database. Each database can have a different leader within the cluster.
Primary	A copy of the database that is able to process write transactions and is eligible to be elected as a leader. It participates in fault tolerant writes as it is part of the majority required to acknowledge and commit write transactions.
Read scaling	Adding secondary copies of the database to the cluster can offload read queries from the primary databases and thus reduce the load and aid write performance of the cluster.

Term	Description
Secondary	An asynchronously replicated copy of the database that provides read scaling within the cluster. It is also suitable for running graph analytic workloads in a cluster using Graph Data Science and taking backups without incurring load on the primary.
Seed	A file used to create a copy of a database on a single instance or on a member of a cluster. This can be a database dump or a database backup. Seed can also be used as a verb to describe the act seeding a cluster from a backup.
Server	A physical machine, a virtual machine, or a container running Neo4j DBMS. The server can be standalone or part of a cluster.
Session consistency	An alternative name for Neo4j's causal consistency.
Standalone server	A single server, or container, running Neo4j DBMS and not part of a cluster.
Synchronous replication	When attempting to commit a transaction, the leader primary replicates the transaction and block, requiring the follower primaries to acknowledge the replication before allowing the commit to proceed. This blocking replication is known as *synchronous*, and ensures data durability and consistency within the cluster. See also asynchronous replication.
Topology	A configuration that describes how the copies of a database should be spread across the servers in a cluster.

Backup and restore

This chapter describes the following:

- Backup and restore planning — What to consider when designing your backup and restore strategy.
- Backup modes — The supported backup modes.
- **Neo4j Community Edition includes the following commands:**
  - Back up an offline database — How to back up an offline database.
  - Restore a database dump — How to restore a database dump in a live Neo4j deployment.
- **In the Neo4j Enterprise Edition, the following commands are available in addition to those in the Community Edition:**
  - Back up an online database — How to back up an online database.
  - Aggregate a database backup chain - How to aggregate a backup chain into a single backup.
  - Introduced in 5.25 Inspect the metadata of a database backup file — How to inspect the metadata of a database backup file.
  - Restore a database backup — How to restore a database backup in a live Neo4j deployment.
  - Copy a database store — How to copy data store from an existing database to a new database.

Backup and restore planning

There are two main reasons for backing up your Neo4j databases and storing them in a safe, off-site location:

- to be able to quickly recover your data in case of failure, for example related to hardware, human error, or natural disaster.
- to be able to perform routine administrative operations, such as moving a database from one instance to another, upgrading, or reclaiming space.

Backup and restore strategy

Depending on your particular deployment and environment, it is important to design an appropriate backup and restore strategy.

There are various factors to consider when deciding on your strategy, such as:

- Type of environment – development, test, or production.
- Data volumes.
- Number of databases.
- Available system resources.
- Downtime tolerance during backup and restore.
- Demands on Neo4j performance during backup and restore. This factor might lead your decision

towards performing these operations during an off-peak period.

- Tolerance for data loss in case of failure.
- Tolerance for downtime in case of failure. If you have zero tolerance for downtime and data loss, you might want to consider performing an online or even a scheduled backup.
- Frequency of updates to the database.
- Type of backup and restore method (online or offline), which may depend on whether you want to:
  - perform full backups (online or offline).
  - perform differential backups (online only).
  - use SSL/TLS for the backup network communication (online only).
  - keep your databases as archive files (online or offline).
- How many backups you want to keep.
- Where the backups will be stored — drive or remote server, cloud storage, different data center, different location, etc.

> **Important:**
>
> Verify the target directory permissions before running backup or dump command. The backup directory must never be world-readable or world-executable. Even if individual backup files inside are restricted, a world-executable directory can still expose sensitive information.
>
> For details and intructions, refer to Security considerations.

> **Tip:**
>
> It is recommended to store your database backups on a separate off-site server (drive or remote) from the database files. This ensures that if for some reason your Neo4j DBMS crashes, you will be able to access the backups and perform a restore.

- How you will test recovery routines, and how often.

Backup and restore options

Neo4j supports backing up and restoring both online and offline databases. It uses Neo4j Admin tool commands that can be executed on a Neo4j DBMS, whether it is running or offline. All `neo4j-admin` commands must be invoked as the `neo4j` user to ensure the appropriate file permissions.

- `neo4j-admin database backup/restore` Enterprise Edition – used for performing online backup (full and differential) and restore operations.
  - The database to be backed up must be in **online** mode.
  - The command produces an immutable artifact, which has an inspectable API to aid management and operability.

- This command is suitable for production environments, where you cannot afford downtime.
- The command can also be invoked over the network if access is enabled using `server.backup.listen_address`.

> **Note:**
>
> Make sure to limit access to the backup server port to fully trusted, specific devices. Firewall policies should be considered. For more information, refer to the Server configurations section.

> **Tip:**
>
> When using `neo4j-admin database backup` in a cluster, it is recommended to back up from an external instance as opposed to reuse instances that form part of the cluster.

- `neo4j-admin database dump/load` -- used for performing offline dump and load operations.
  - The database to be dumped must be in **offline** mode.
  - The dump command can only be invoked from the server command line and is suitable for environments where downtime is not a factor.
  - The command produces an archive file that follows the format *<databasename><timestamp>.dump*.
- `neo4j-admin database copy` -- used for copying an offline database or backup. This command can be used for cleaning up database inconsistencies and reclaiming unused space.

> **Warning:**
>
> File system copy-and-paste of databases is not supported and may result in unwanted behavior, such as corrupt stores.

The following table summarizes the commands' capabilities and usage.

Table 402. `neo4j-admin` commands for backing up and restoring databases

Capability/ Usage	`backup/restore`	`dump/load`	`copy`
Neo4j Edition	Enterprise	all	Enterprise
Run from an online Neo4j DBMS	✓	✓ Enterprise Only	✓
Run from an offline Neo4j DBMS	✗	✓	✓
Run against a user database	✓	✓	✓

Capability/ Usage	`backup/restore`	`dump/load`	`copy`
Run against the `system` database	✔	✔	✖
Run against a composite database	✖	✖	✖
Perform full backups	✔	✔	n/a
Perform differential backups	✔	✖	n/a
Applied to an online database	✔	✖	✖
Applied to an offline database	only `restore`	✔	✔
Can be run remotely	only `backup`	✖	✔
Command input	database/archive (.backup)	database/archive (.dump)	database
Command output	archive (.backup)/database	archive (.dump)/database	database; no schema store
Clean up database inconsistencies	✖	✖	✔
Compact data store	✖	✖	✔

Note:

The Neo4j Admin commands `backup`, `restore`, `dump`, `load`, `copy`, and `check-consistency` are not supported for use on Composite databases. They must be run directly on the databases that are associated with that Composite database.

Considerations for backing up and restoring databases in a cluster

Backing up a database in a clustered environment is not essentially different from a standalone backup, apart from the fact that you must know which server in a cluster to connect to. Use `SHOW DATABASE <database>` to learn which servers are hosting the database you want to back up. See Listing a single database for more information.

Restoring from the command line involves putting a copy of the database on disk on each server that will need it. That can be awkward to achieve. The recommended way to restore a database in a cluster is to seed from URI.

Important:

By default, when backing up a database using the `neo4j-admin database backup` command (Enterprise edition), the backup archive includes both the database contents and the users and roles metadata. You can control whether you want to include the `users`, `roles`, `all`, or `none` using the argument `--include-metadata`.

When restoring, you have the flexibility to define the target topology (how many primaries and secondaries are desired for the database), which may differ from the topology at backup time. The database will then be allocated across the available servers according to that topology.

Databases to back up

A Neo4j DBMS can host multiple databases. Both Neo4j Community and Enterprise Editions have a default user database named `neo4j` and a `system` database. The `system` database contains configurations, e.g., operational states of databases, security configuration, schema definitions, login credentials, and roles.

In the Enterprise Edition, you can also create multiple user databases. Each of these databases is backed up independently of one another.

It is very important to store a recent backup of your databases, including the `system` database, in a safe location.

Additional files to back up

The following files must be backed up separately from the databases:

- The *neo4j.conf* file. If you have a cluster deployment, you should back up the configuration file for each cluster member.
- All the files used for encryption, i.e., private key, public certificate, and the contents of the *trusted* and *revoked* directories. The locations of these are described in SSL framework. If you have a cluster, you should back up these files for each cluster member.
- If using custom plugins, make sure that you have the plugins in a safe location.
- If using Bloom or GDS Enterprise, back up license key files for these products as well.

Storage considerations

For any backup, it is important that you store your data separately from the production system, where there are no common dependencies, and preferably off-site. If you are running Neo4j in the cloud, you may use a different availability zone or even a separate cloud provider. Since backups are kept for a long time, the longevity of archival storage should be considered as part of backup planning.

Backup modes

The backup client can operate in two different modes – a *full backup* and a *differential backup*.

Full backup

A full backup is always required initially for the very first backup into a target location.

The full backup can be run against an **online** (using the Enterprise edition command — `neo4j-admin`

`database backup`) and an **offline** (using `neo4j-admin database dump`) database.

`neo4j-admin database backup`

Example 81. Backing up an online database

```
export HEAP_SIZE=2G
mkdir /mnt/backups
bin/neo4j-admin database backup --from=192.168.1.34 --to-path=/mnt/backups/neo4j --pagecache=4G neo4j
Doing full backup...
2017-02-01 14:09:09.510+0000 INFO  [o.n.c.s.StoreCopyClient] Copying neostore.nodestore.db.labels
2017-02-01 14:09:09.537+0000 INFO  [o.n.c.s.StoreCopyClient] Copied neostore.nodestore.db.labels 8.00
kB
2017-02-01 14:09:09.538+0000 INFO  [o.n.c.s.StoreCopyClient] Copying neostore.nodestore.db
2017-02-01 14:09:09.540+0000 INFO  [o.n.c.s.StoreCopyClient] Copied neostore.nodestore.db 16.00 kB
...
...
...
```

The `neo4j-admin database backup` command with the `--type=full` argument creates a full backup artifact file in the format of *<databasename><timestamp>.backup*, in the target location specified by `--to-path`.

The `neo4j-admin database backup` command allows you to include metadata, enabling the backup to store the role-based access control (RBAC) metadata associated with the database.

For more information about online backup options and how to control memory usage, see Back up an online database.

`neo4j-admin database dump`

The `neo4j-admin database dump` command creates a single-file archive that follows the format *<databasename><timestamp>.dump*, and stores it in the *<NEO4J_HOME>/data* directory unless specified by `--to-path=<path>`.

When using `neo4j-admin database dump`, only the database data is backed up. The associated users and roles metadata is not included and must be recreated manually after restore. See Authentication and authorization for details.

For more information about performing a full backup against an **offline** database, see Back up an offline database.

Differential backup

In the online backup version, after the initial full backup, subsequent backups attempt to use differential mode, where only the delta of the transaction logs since the last backup is transferred and used to create a differential backup artifact (stored in the target location). Those differential backup artifacts form a backup chain. If the required transaction logs are not available on the backup server, then the backup client falls back on performing a full backup instead.

Example 82. Differential backup against an online database

```
export HEAP_SIZE=2G
```

```
bin/neo4j-admin database backup --from=192.168.1.34 --to-path=/mnt/backups/neo4j --pagecache=4G neo4j
Destination is not empty, doing differential backup...
Backup complete.
```

Back up an online database

> **Caution:**
>
> Remember to plan your backup carefully and back up each of your databases, including the `system` database.
>
> Note that it is not allowed to take a backup of a database alias, only physical databases can be backed up.

Command

A Neo4j database can be backed up in **online mode** using the `backup` command of `neo4j-admin`. The command must be invoked as the `neo4j` user to ensure the appropriate file permissions.

It is best practice, but not mandatory, to perform the backup from a server on the same network as the database, but that is not part of the cluster. You should install Neo4j on that machine to make the `neo4j-admin` command available. This machine is known as a backup client.

> **Note:**
>
> Neo4j 5 introduces a new version of the backup command which produces immutable backup artifacts (as opposed to mutable folders as in previous versions).

Backup artifact

The `neo4j-admin database backup` command produces one backup artifact file per database each time it is run. A backup artifact file is an immutable file containing the backup data of a given database along with some metadata like the database name and ID, the backup time, the lowest/highest transaction ID, etc.

Backup artifacts can be of two types:

1. a *full backup* containing the whole database store or
2. a *differential backup* containing a log of transactions to apply to a database store contained in a full backup artifact.

Backup chain

The first time the backup command is run, a full backup artifact is produced for a given database. On the other hand, differential backup artifacts are produced by the subsequent runs.

A *backup chain* consists of a full backup optionally followed by a sequence of n contiguous differential backups.

Figure 27. Backup chain

Usage

The `neo4j-admin database backup` command can be used for performing an online full or differential backup from a running Neo4j Enterprise server. The produced differential backup artifact contains transaction logs that can be replayed and applied to stores contained in full backup artifacts when restoring a backup chain.

Neo4j's backup service must have been configured on the server beforehand. The command can be run both locally and remotely. However, it uses a significant amount of resources, such as memory and CPU. Therefore, it is recommended to perform the backup on a separate dedicated machine. The `neo4j-admin database backup` command also supports SSL/TLS. For more information, see Online backup configurations.

> **Note:**
>
> `neo4j-admin database backup` is not supported in Neo4j Aura.

Syntax

```
neo4j-admin database backup [-h] [--expand-commands] [--verbose]
                            [--compress[=true|false]] [--keep-failed[=true|false]]
                            [--parallel-recovery[=true|false]]
                            [--additional-config=<file>]
                            [--include-metadata=none|all|users|roles]
                            [--inspect-path=<path>] [--pagecache=<size>] [--temp-path=<path>]
                            [--to-path=<path>] [--type=<type>] [--from=<host:port>[,<host:
                            port>...]]... [<database>...]
```

Description

Perform an online backup from a running Neo4j enterprise server. Neo4j's backup service must have been configured on the server beforehand.

Parameters

Table 403. `neo4j-admin database backup` parameters

Parameter	Description	Default
`[<database>…]`	Name(s) of the remote database(s) to backup. Supports globbing inside of double quotes, for example, "data*". (<database> is required unless `--inspect-path` is used.)	`neo4j`

Tip:

If <database> is "*", `neo4j-admin` will attempt to back up all databases of the DBMS.

Options

Table 404. `neo4j-admin database backup` *options*

Option	Description	Default
`--additional-config=<file>`[1]	Configuration file with additional configuration.	
`--compress[=true|false]`	Request backup artifact to be compressed. Compression can yield a backup artefact many times smaller, but the exact reduction depends upon many factors, including the database format and the kind of data stored. If disabled, the size of the produced artifact will be approximately equal to the size of the backed-up database. The speed of the backup operation is affected by compression, but which is faster depends upon the relative performance of CPU and storage. If backup speed is important, consider evaluating both options - with compression enabled and disabled.	`true`
`--expand-commands`	Allow command expansion in config value evaluation.	
`--from=<host:port>[,<host:port>…]`	Comma-separated list of host and port of Neo4j instances, each of which are tried in order.	
`-h, --help`	Show this help message and exit.	

Option	Description	Default
`--include-metadata=none|all|users|roles`	Include metadata in the file. This cannot be used for backing up the `system` database. Possible values are: • `roles` - include commands to create the roles and privileges (for both database and graph) that affect the use of the database. • `users` - include commands to create the users that can use the database and their role assignments. • `all` - include both `roles` and `users`. • `none` - does not include any metadata. Privileges specific to the DBMS and not to the backed-up database are not included in the backup. For instance, `GRANT ROLE MANAGEMENT ON DBMS TO $role` will not be backed up. Accordingly, `roles` and `users` that do not have database-related privileges are not included in the backup (e.g. those with only DBMS or no privileges). It is recommended to use `SHOW USERS`, `SHOW ROLES`, and `SHOW ROLE $role PRIVILEGES AS COMMANDS` to get the complete list of users, roles and privileges in these situations.	`all`
`--inspect-path=<path>`	List and show the metadata of the backup artifact(s). Accepts a folder or a file.	
`--keep-failed[=true|false]`	Request failed backup to be preserved for further post-failure analysis. If enabled, a directory with the failed backup database is preserved.	`false`
`--pagecache=<size>`	The size of the page cache to use for the backup process.	
`--parallel-recovery[=true|false]`	Allow multiple threads to apply pulled transactions to a backup in parallel. For some databases and workloads, this may reduce backup times significantly. Note: this is an EXPERIMENTAL option. Consult Neo4j support before use.	`false`

Option	Description	Default
`--temp-path=<path>`	Introduced in 5.24 Provide a path to a temporary empty directory for storing backup files until the command is completed. The files will be deleted once the command is finished.	
`--to-path=<path>`	Directory to place backup in (required unless `--inspect-path` is used). It is possible to back up databases into AWS S3 buckets, Google Cloud storage buckets, and Azure using the appropriate URI as the path.	
`--type=<type>`	Type of backup to perform. Possible values are: `FULL`, `DIFF`, `AUTO`. If none is specified, the type is automatically determined based on the existing backups. If you want to force a full backup, use `FULL`.	`AUTO`
`--verbose`	Enable verbose output.	

Note:

The `--to-path=<path>` option can also back up databases into AWS S3 buckets (from Neo4j 5.19), Google Cloud storage buckets (from Neo4j 5.21), and Azure buckets (from 5.24). For more information, see Back up a database to a cloud storage.

Note:

Neo4j 5.24 introduces the `--temp-path` option to address potential issues related to disk space when performing backup-related commands, especially when cloud storage is involved.

If `--temp-path` is not set, a temporary directory is created inside the directory specified by the `--path` option.

If you don't provide the `--path` option or if your provided path points to a cloud storage bucket, a temporary folder is created inside the current working directory for Neo4j. This fallback option can cause issues because the local filesystem (or the partition where Neo4j is installed) may not have enough free disk to accommodate the intermediate computation.

Therefore, it is strongly recommended to provide a `--temp-path` option when executing a backup-related command, especially if the folder provided in the `--path` option points to a cloud storage bucket.

Exit codes

Depending on whether the backup was successful or not, `neo4j-admin database backup` exits with different codes. The error codes include details of what error was encountered.

Table 405. Neo4j Admin backup exit codes when backing up one database

Code	Description
`0`	Success.
`1`	Backup failed, or succeeded but encountered problems such as some servers being uncontactable. See logs for more details.

Table 406. Neo4j Admin backup exit codes when backing multiple databases

Code	Description
`0`	All databases are backed up successfully.
`1`	One or several backups failed, or succeeded with problems.

Online backup configurations

Checkpointing

When a full backup is requested, it always triggers a checkpoint. The backup cannot proceed until the checkpoint finishes.

While the server is checkpointing, the backup job receives no data, which may lead to the backup timeout. To extend the backup timeout, modify the `dbms.cluster.catchup.client_inactivity_timeout` setting, which restricts the network inactivity. It controls the timeout duration of the catchup protocol, which is the underlying protocol of multiple catchup processes, including backups.

You can also tune up the Checkpoint settings or check that your disks are performant enough to handle the load. For more information, see Checkpoint IOPS limit.

To read more about checkpointing, see Database internals → Checkpointing and log pruning.

Server configuration

The table below lists the basic server parameters relevant to backups. Note that by default, the backup service is enabled but only listens on localhost (127.0.0.1). This needs to be changed if backups are to be taken from another machine.

Table 407. Server parameters for backups

Parameter name	Default value	Description
`server.backup.enabled`	`true`	Enable support for running online backups.
`server.backup.listen_address`	`127.0.0.1:6362`	Listening server for online backups.

Memory configuration

You can configure the memory allocated to the backup client in several ways:

- Configure heap size:

  `HEAP_SIZE` configures the maximum heap size allocated for the backup process. Define the variable `HEAP_SIZE` before starting the operation. If not specified, the Java Virtual Machine chooses a value based on the server resources.

- Configure page cache:

  Set the page cache size with the `--pagecache` option of the `neo4j-admin database backup` command.

- Override the memory settings:

  Use the `--additional-config` option of the `neo4j-admin database backup` command to override the memory configurations in the *neo4j.conf* file.

Tip:

You should give the Neo4J page cache as much memory as possible, as long as it satisfies the following constraint:

Neo4J page cache + OS page cache < available RAM, where 2 to 4GB should be dedicated to the operating system's page cache.

For example, if your current database has a `Total mapped size` of `128GB` as per the *debug.log*, and you have enough free space (meaning you have left aside 2 to 4 GB for the OS), then you can set `--pagecache` to `128GB`.

Computational resource configurations

Transaction log files

The transaction log files, which keep track of recent changes, are rotated and pruned based on a provided configuration. For example, setting `db.tx_log.rotation.retention_policy=3` files keeps 3 transaction log files in the backup. Because recovered servers do not need all of the transaction log files that have already been applied, it is possible to further reduce storage size by reducing the size of the files to the bare minimum. This can be done by setting `db.tx_log.rotation.size=1M` and `db.tx_log.rotation.retention_policy=3` files. You can use the `--additional-config` parameter to override the configurations in the *neo4j.conf* file.

Warning:

Removing transaction logs manually can result in a broken backup.

Security configurations

Securing your backup network communication with an SSL policy and a firewall protects your data from unwanted intrusion and leakage. When using the `neo4j-admin database backup` command, you can configure the backup server to require SSL/TLS, and the backup client to use a compatible policy. For more information on how to configure SSL in Neo4j, see SSL framework.

Configuration for the backup server should be added to the *neo4j.conf* file and configuration for backup client to the *neo4j-admin.conf* file. The easiest way to ensure compatibility is to use the same SSL policy configuration for both the server and the client. For production environments, it is recommended to use Certificate Authorities (CAs) to sign certificates rather than self-signed certificates.

The default backup port is 6362, configured with key `server.backup.listen_address`. The SSL configuration policy has the key of `dbms.ssl.policy.backup`.

As an example, add the following content to your *neo4j.conf* and *neo4j-admin.conf* files:

Server configuration in neo4j.conf

```
dbms.ssl.policy.backup.enabled=true
dbms.ssl.policy.backup.client_auth=REQUIRE
dbms.ssl.policy.backup.tls_versions=TLSv1.2,TLSv1.3
dbms.ssl.policy.backup.ciphers=TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA3
84
```

Client configuration in neo4j-admin.conf

```
dbms.ssl.policy.backup.enabled=true
dbms.ssl.policy.backup.client_auth=REQUIRE
dbms.ssl.policy.backup.tls_versions=TLSv1.2,TLSv1.3
dbms.ssl.policy.backup.ciphers=TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA3
84
```

Tip:

Neo4j 5.24 also supports TLSv1.3. To use both TLSv1.2 and TLSv1.3 versions, you must specify which ciphers to be enforced for each version. Otherwise, Neo4j could use every possible cipher in the JVM for those versions, leading to a less secure configuration.

For a detailed list of recommendations regarding security in Neo4j, see Security checklist.

Note:

It is very important to ensure that there is no external access to the port specified by the setting `server.backup.listen_address`. Failing to protect this port may leave a security hole open by which an unauthorized user can make a copy of the database onto a different machine. In production environments, external access to the backup port should be blocked by a firewall.

Cluster configurations

In a cluster topology, it is possible to take a backup from any server hosting the database to backup, and each server has two configurable ports capable of serving a backup. These ports are configured by `server.backup.listen_address` and `server.cluster.listen_address` respectively. Functionally, they are equivalent for backups, but separating them can allow some operational flexibility, while using just a single port can simplify the configuration.

It is generally recommended to select secondary database copies to act as backup resources since they are

more numerous than primary copies in typical cluster deployments. Furthermore, the possibility of performance issues on a secondary database allocation, caused by a large backup, does not affect the performance or redundancy of the primaries. If a secondary is not available, then a primary can be selected based on factors, such as its physical proximity, bandwidth, performance, and liveness.

Note:

Use the `SHOW DATABASES` command to learn which database is hosted on which server.

Note:

To avoid taking a backup from a database member that is lagging behind, you can look at the transaction IDs by exposing Neo4j metrics or via Neo4j Browser. To view the latest processed transaction IDs (and other metrics) in Neo4j Browser, type `:sysinfo` at the prompt.

Targeting multiple servers

It is recommended to provide a list of multiple target servers when taking a backup from a cluster, since that may allow a backup to succeed even if some server is down, or not all databases are hosted on the same servers. If the command finds one or more servers that do not respond, it continues trying to backup from other servers and continues backing up other requested databases, but the exit code of the command is non-zero, to alert the user to the fact there is a problem. If a name pattern is used for the database together with multiple target servers, all servers contribute to the list of matching databases.

Examples

The following are examples of how to perform a backup of a single database and multiple databases. The target directory */mnt/backups/neo4j* must exist before calling the command and the database(s) must be online.

Back up a single database

You do not need to use the `--type` option to specify the type of backup. By default, the type is automatically determined based on the existing backups.

```
bin/neo4j-admin database backup --to-path=/path/to/backups/neo4j neo4j
```

Perform a forced full backup of a single database

If you want to force a full backup after several differential backups, you can use the `--type=full` option.

```
bin/neo4j-admin database backup --type=full --to-path=/path/to/backups/neo4j neo4j
```

Back up multiple databases

To back up multiple databases that match a name pattern, you can use name globbing. For example, to backup all databases that start with **n** in your three-server cluster, run:

```
bin/neo4j-admin database backup --from=192.168.1.34:6362,192.168.1.35:6362,192.168.1.36:6362 --to-path
=/mnt/backups/neo4j --pagecache=4G "n*"
```

Back up a list of databases

To back up several databases by name, you can provide a list of database names.

```
neo4j-admin database backup --from=192.168.1.34:6362,192.168.1.35:6362,192.168.1.36:6362 --to-path
=/mnt/backups/neo4j --pagecache=4G "test*" "neo4j"
```

Back up a database to a cloud storage

The following examples show how to back up a database to a cloud storage bucket using the `--to-path` option.

Note:

Neo4j uses the AWS SDK v2 to call the APIs on AWS using AWS URLs. Alternatively, you can override the endpoints so that the AWS SDK can communicate with alternative storage systems, such as Ceph, Minio, or LocalStack, using the system variables `aws.endpointUrls3`, `aws.endpointUrlS3`, or `aws.endpointUrl`, or the environments variables `AWS_ENDPOINT_URL_S3` or `AWS_ENDPOINT_URL`.

1. Install the AWS CLI by following the instructions in the AWS official documentation — Install the AWS CLI version 2.
2. Create an S3 bucket and a directory to store the backup files using the AWS CLI:

   ```
   aws s3 mb --region=us-east-1 s3://myBucket
   aws s3api put-object --bucket myBucket --key myDirectory/
   ```

   For more information on how to create a bucket and use the AWS CLI, see the AWS official documentation — Use Amazon S3 with the AWS CLI and Use high-level (s3) commands with the AWS CLI.

3. Verify that the `~/.aws/config` file is correct by running the following command:

   ```
   cat ~/.aws/config
   ```

   The output should look like this:

   ```
   [default]
   ```

```
region=us-east-1
```

4. Configure the access to your AWS S3 bucket by setting the `aws_access_key_id` and `aws_secret_access_key` in the `~/.aws/credentials` file and, if needed, using a bucket policy. For example:

   a. Use `aws configure set aws_access_key_id aws_secret_access_key` command to set your IAM credentials from AWS and verify that the `~/.aws/credentials` is correct:

   ```
   cat ~/.aws/credentials
   ```

   The output should look like this:

   ```
   [default]
   aws_access_key_id=this.is.secret
   aws_secret_access_key=this.is.super.secret
   ```

   b. Additionally, you can use a resource-based policy to grant access permissions to your S3 bucket and the objects in it. Create a policy document with the following content and attach it to the bucket. Note that both resource entries are important to be able to download and upload files.

   ```
   {
       "Version": "2012-10-17",
       "Id": "Neo4jBackupAggregatePolicy",
       "Statement": [
           {
               "Sid": "Neo4jBackupAggregateStatement",
               "Effect": "Allow",
               "Action": [
                   "s3:ListBucket",
                   "s3:GetObject",
                   "s3:PutObject",
                   "s3:DeleteObject"
               ],
               "Resource": [
                   "arn:aws:s3:::myBucket/*",
                   "arn:aws:s3:::myBucket"
               ]
           }
       ]
   }
   ```

5. Run the `neo4j-admin database backup` command to back up your database to your AWS S3 bucket:

   ```
   bin/neo4j-admin database backup --to-path=s3://myBucket/myDirectory/ mydatabase
   ```

1. Ensure you have a Google account and a project created in the Google Cloud Platform (GCP).

   a. Install the `gcloud` CLI by following the instructions in the Google official documentation — Install the gcloud CLI.

   b. Create a service account and a service account key using Google official

documentation — Create service accounts and Creating and managing service account keys.

c. Download the JSON key file for the service account.

d. Set the `GOOGLE_APPLICATION_CREDENTIALS` and `GOOGLE_CLOUD_PROJECT` environment variables to the path of the JSON key file and the project ID, respectively:

```
export GOOGLE_APPLICATION_CREDENTIALS="/path/to/keyfile.json"
export GOOGLE_CLOUD_PROJECT=YOUR_PROJECT_ID
```

e. Authenticate the `gcloud` CLI with the e-mail address of the service account you have created, the path to the JSON key file, and the project ID:

```
gcloud auth activate-service-account service-account@example.com --key-file
=$GOOGLE_APPLICATION_CREDENTIALS --project=$GOOGLE_CLOUD_PROJECT
```

For more information, see the Google official documentation — gcloud auth activate-service-account.

f. Create a bucket in the Google Cloud Storage using Google official documentation — Create buckets.

g. Verify that the bucket is created by running the following command:

```
gcloud storage ls
```

The output should list the created bucket.

2. Run `neo4j-admin database backup` command to back up your database to your Google bucket:

```
bin/neo4j-admin database backup --to-path=gs://myBucket/myDirectory/ mydatabase
```

1. Ensure you have an Azure account, an Azure storage account, and a blob container.

   a. You can create a storage account using the Azure portal.
   For more information, see the Azure official documentation on Create a storage account.

   b. Create a blob container in the Azure portal.
   For more information, see the Azure official documentation on Quickstart: Upload, download, and list blobs with the Azure portal.

2. Install the Azure CLI by following the instructions in the Azure official documentation — Azure official documentation.

3. Authenticate the neo4j or neo4j-admin process against Azure using the default Azure credentials.
   See the Azure official documentation on default Azure credentials for more information.

```
az login
```

Then you should be ready to use Azure URLs in either neo4j or neo4j-admin.

4. To validate that you have access to the container with your login credentials, run the following commands:

```
# Upload a file:
az storage blob upload --file someLocalFile  --account-name accountName - --container
someContainer --name remoteFileName  --auth-mode login

# Download the file
az storage blob download  --account-name accountName --container someContainer --name
remoteFileName --file downloadedFile --auth-mode login

# List container files
az storage blob list  --account-name someContainer --container someContainer  --auth-mode
login
```

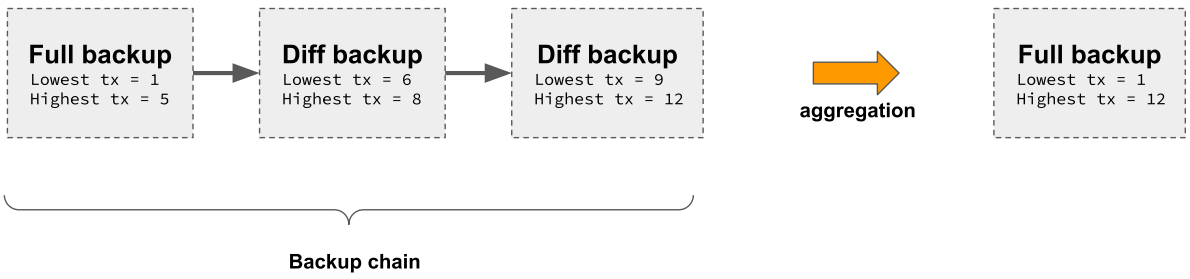
5. Run `neo4j-admin database backup` command to back up your database to your Azure container:

```
bin/neo4j-admin database backup --to-path=azb://myStorageAccount/myContainer/myDirectory/
mydatabase
```

Aggregate a database backup chain

Command

The aggregate command turns a a chain of backup artifacts into a single full backup artifact.



The benefits of aggregating a backup chain are notably:

- Reduces the size of backup artifacts in a given backup folder.
- Keeps the recovery time objective (RTO) low by generating a single backup artifact ready to be restored. As part of the aggregation, transactions contained in the differential backups are applied to the store contained in the full backup artifact. This operation is called *recovery* and can be costly.
- Reduces the risk of losing chain's links.

Syntax

```
neo4j-admin database aggregate-backup [-h] [--expand-commands]
                                      [--verbose] [--keep-old-backup[=true|false]]
                                      [--parallel-recovery[=true|false]]
                                      [--additional-config=<file>] --from-path=<path>
                                      [--temp-path=<path>] [<database>]
```

Description

Aggregates a chain of backup artifacts into a single artifact.

Parameters

Table 408. `neo4j-admin database aggregate-backup` *parameters*

Parameter	Description
`[<database>]`	Name of the database for which to aggregate the artifacts. Can contain * and ? for globbing.

Options

Table 409. `neo4j-admin database aggregate-backup` *options*

Option	Description	Default
`--additional-config=<file>`[2]	Configuration file with additional configuration.	
`--expand-commands`	Allow command expansion in config value evaluation.	
`--from-path=<path>`	Accepts either a path to a single artifact file or a folder containing backup artifacts. When a file is supplied, the <database> parameter should be omitted. It is possible to aggregate backup artifacts from AWS S3 buckets, Google Cloud storage buckets, and Azure buckets using the appropriate URI as the path.	
`-h, --help`	Show this help message and exit.	
`--keep-old-backup[=true|false]`	If set to true, the old backup chain is not removed.	`false`

Option	Description	Default
`--parallel-recovery[=true|false]`	Allow multiple threads to apply pulled transactions to a backup in parallel. For some databases and workloads, this may reduce aggregate times significantly. Note: this is an EXPERIMENTAL option. Consult Neo4j support before use.	`false`
`--temp-path=<path>`	Introduced in 5.24 Provide a path to a temporary empty directory for storing backup files until the command is completed. The files will be deleted once the command is finished.	
`--verbose`	Enable verbose output.	

Note:

The `--from-path=<path>` option can also load backup artifacts from AWS S3 buckets (from Neo4j 5.19), Google Cloud storage buckets (from Neo4j 5.21), and Azure buckets (from Neo4j 5.24). For more information, see Aggregating a backup chain located in a cloud storage.

Note:

Neo4j 5.24 introduces the `--temp-path` option to address potential issues related to disk space when performing backup-related commands, especially when cloud storage is involved.

If `--temp-path` is not set, a temporary directory is created inside the directory specified by the `--from-path` option.

If you don't provide the `--from-path` option or if your provided path points to a cloud storage bucket, a temporary folder is created inside the current working directory for Neo4j. This fallback option can cause issues because the local filesystem (or the partition where Neo4j is installed) may not have enough free disk to accommodate the intermediate computation.

Therefore, it is strongly recommended to provide a `--temp-path` option.

Examples

Aggregating a backup chain located in a given folder

The following is an example of how to perform aggregation of a set of backups located in a given folder for the `neo4j` database:

```
bin/neo4j-admin database aggregate-backup --from-path=/mnt/backups/ neo4j
```

The command first looks inside the `/mnt/backups/` directory for a backup chain for the database `neo4j`. If found, it is then aggregated into a single backup artifact.

Aggregating a backup chain identified using a given backup file

The following is an example of how to perform aggregation of a set of backups identified using a given backup file for the `neo4j` database:

```
bin/neo4j-admin database aggregate-backup --from-path=/mnt/backups/neo4j-2022-10-18T13-00-07.backup
```

The command checks the `/mnt/backups/` directory for a backup chain including the file *neo4j-2022-10-18T13-00-07.backup*, for the database `neo4j`. If found, it is then aggregated into a single backup artifact. This option is only available in Neo4j 5.2 and later.

Aggregating a backup chain located in a cloud storage

The following examples show how to perform aggregation of a set of backups located in a cloud storage.

Note:

Neo4j uses the AWS SDK v2 to call the APIs on AWS using AWS URLs. Alternatively, you can override the endpoints so that the AWS SDK can communicate with alternative storage systems, such as Ceph, Minio, or LocalStack, using the system variables `aws.endpointUrls3`, `aws.endpointUrlS3`, or `aws.endpointUrl`, or the environments variables `AWS_ENDPOINT_URL_S3` or `AWS_ENDPOINT_URL`.

1. Install the AWS CLI by following the instructions in the AWS official documentation — Install the AWS CLI version 2.
2. Create an S3 bucket and a directory to store the backup files using the AWS CLI:

   ```
   aws s3 mb --region=us-east-1 s3://myBucket
   aws s3api put-object --bucket myBucket --key myDirectory/
   ```

   For more information on how to create a bucket and use the AWS CLI, see the AWS official documentation — Use Amazon S3 with the AWS CLI and Use high-level (s3) commands with the AWS CLI.

3. Verify that the `~/.aws/config` file is correct by running the following command:

   ```
   cat ~/.aws/config
   ```

   The output should look like this:

   ```
   [default]
   ```

```
region=us-east-1
```

4. Configure the access to your AWS S3 bucket by setting the `aws_access_key_id` and `aws_secret_access_key` in the `~/.aws/credentials` file and, if needed, using a bucket policy. For example:

   a. Use `aws configure set aws_access_key_id aws_secret_access_key` command to set your IAM credentials from AWS and verify that the `~/.aws/credentials` is correct:

   ```
   cat ~/.aws/credentials
   ```

   The output should look like this:

   ```
   [default]
   aws_access_key_id=this.is.secret
   aws_secret_access_key=this.is.super.secret
   ```

   b. Additionally, you can use a resource-based policy to grant access permissions to your S3 bucket and the objects in it. Create a policy document with the following content and attach it to the bucket. Note that both resource entries are important to be able to download and upload files.

   ```
   {
       "Version": "2012-10-17",
       "Id": "Neo4jBackupAggregatePolicy",
       "Statement": [
           {
               "Sid": "Neo4jBackupAggregateStatement",
               "Effect": "Allow",
               "Action": [
                   "s3:ListBucket",
                   "s3:GetObject",
                   "s3:PutObject",
                   "s3:DeleteObject"
               ],
               "Resource": [
                   "arn:aws:s3:::myBucket/*",
                   "arn:aws:s3:::myBucket"
               ]
           }
       ]
   }
   ```

5. Then, use the following command to aggregate the backup chain located in a given folder in your AWS S3 bucket. The example assumes that you have a backup chain located in the `myBucket/myDirectory` folder identifiable by the file `myBackup.backup`:

   ```
   bin/neo4j-admin database aggregate-backup --from-path
   =s3://myBucket/myDirectory/myBackup.backup mydatabase
   ```

1. Ensure you have a Google account and a project created in the Google Cloud Platform (GCP).

   a. Install the `gcloud` CLI by following the instructions in the Google official documentation — Install the gcloud CLI.

b. Create a service account and a service account key using Google official documentation — Create service accounts and Creating and managing service account keys.

c. Download the JSON key file for the service account.

d. Set the `GOOGLE_APPLICATION_CREDENTIALS` and `GOOGLE_CLOUD_PROJECT` environment variables to the path of the JSON key file and the project ID, respectively:

```
export GOOGLE_APPLICATION_CREDENTIALS="/path/to/keyfile.json"
export GOOGLE_CLOUD_PROJECT=YOUR_PROJECT_ID
```

e. Authenticate the `gcloud` CLI with the e-mail address of the service account you have created, the path to the JSON key file, and the project ID:

```
gcloud auth activate-service-account service-account@example.com --key-file
=$GOOGLE_APPLICATION_CREDENTIALS --project=$GOOGLE_CLOUD_PROJECT
```

For more information, see the Google official documentation — gcloud auth activate-service-account.

f. Create a bucket in the Google Cloud Storage using Google official documentation — Create buckets.

g. Verify that the bucket is created by running the following command:

```
gcloud storage ls
```

The output should list the created bucket.

2. Then, use the following command to aggregate the backup chain located in a given folder in your Google storage bucket. The example assumes that you have a backup chain located in the `myBucket/myDirectory` folder identifiable by the file `myBackup.backup`:

```
bin/neo4j-admin database aggregate-backup --from-path
=gs://myBucket/myDirectory/myBackup.backup mydatabase
```

1. Ensure you have an Azure account, an Azure storage account, and a blob container.

   a. You can create a storage account using the Azure portal.
   For more information, see the Azure official documentation on Create a storage account.

   b. Create a blob container in the Azure portal.
   For more information, see the Azure official documentation on Quickstart: Upload, download, and list blobs with the Azure portal.

2. Install the Azure CLI by following the instructions in the Azure official documentation — Azure official documentation.

3. Authenticate the neo4j or neo4j-admin process against Azure using the default Azure credentials.

See the Azure official documentation on default Azure credentials for more information.

```
az login
```

Then you should be ready to use Azure URLs in either neo4j or neo4j-admin.

4. To validate that you have access to the container with your login credentials, run the following commands:

```
# Upload a file:
az storage blob upload --file someLocalFile  --account-name accountName - --container
someContainer --name remoteFileName  --auth-mode login

# Download the file
az storage blob download  --account-name accountName --container someContainer --name
remoteFileName --file downloadedFile --auth-mode login

# List container files
az storage blob list  --account-name someContainer --container someContainer  --auth-mode
login
```

5. Then, use the following command to aggregate the backup chain located in a given folder in your Azure blob storage container. The example assumes that you have a backup chain located with a `myStorageAccount/myContainer/myDirectory` folder identifiable by the file `myBackup.backup`:

```
bin/neo4j-admin database aggregate-backup --from-path
=azb://myStorageAccount/myContainer/myDirectory/myBackup.backup mydatabase
```

Inspect the metadata of a backup file

Enterprise Edition | Introduced in 5.25

You can inspect the metadata of a database backup file using the `neo4j-admin backup inspect` command.

Command

The inspect command lists the metadata stored in the header of backup files. This metadata primarily defines how backups are connected to form backup chains. A backup chain is a sequence of one or more backup(s) logically connected. The order of the sequence guarantees that when replayed (see restore or aggregate), the store and the transaction data are consumed in a consistent manner.

The metadata contains the following information:

- **Database**: database name of the database fragment that the backup includes.
- **Database ID**: a unique identifier that distinguishes databases (even with the same name).
- **Time**: time the backup was taken.
- **Full**: indicates whether it is a full backup (i.e. initial backup containing the store files) or a differential

backup (i.e. subsequent backup containing only the transactions to be applied to the store files).

- **Compressed**: indicates whether the backup data inside the backup file is compressed.
- **Lowest transaction ID**: when the backup is full, this value is always 1, and when it is a differential backup, the value corresponds to the first transaction ID the backup starts with.
- **Highest transaction ID**: similarly, this value indicates the last transaction ID stored in the backup file.

Syntax

```
neo4j-admin backup inspect [-h] [--empty] [--expand-commands] [--latest-backup]
                           [--latest-chain] [--show-metadata] [--verbose]
                           [--additional-config=<file>] [--database=<database>]
                           [--format=<value>] <backup-path>
```

Description

Command to read the backup metadata.

Parameters

Table 410. `neo4j-admin backup inspect` parameters

Parameter	Description
`<backup-path>`	Path denoting either a directory where backups are stored or a single backup to inspect.

Note:

The `<backup-path>` parameter can also inspect backups stored in AWS S3 buckets (from Neo4j 5.19), Google Cloud storage buckets (from Neo4j 5.21), and Azure buckets (from Neo4j 5.24).

Options

Table 411. `neo4j-admin backup inspect` options

Option	Description	Default
`--additional-config=<file>`[3]	Configuration file with additional configuration.	
`--expand-commands`	Allow command expansion in config value evaluation.	
`-h, --help`	Show this help message and exit.	
`--latest-backup`	Show only the latest backup.	`false`
`--latest-chain`	List the full backup chain ending with the latest downloaded backup.	`false`
`--show-metadata`	Show the backup metadata.	`false`

Option	Description	Default
`--database=<database>`	Name of the database to inspect.	
`--format=<value>`	Format of the output of the command. Possible values are: 'JSON, TABULAR'.	`TABULAR`
`--empty`	Include empty backups.	`false`
`--verbose`	Enable verbose output.	

Examples

Given the folder */backups* containing a set of database backups:

```
/backups
├── london-2024-10-07T16-03-51.backup
├── london-2024-10-07T16-04-05.backup
├── malmo-2024-10-07T16-00-07.backup
├── malmo-2024-10-07T16-00-19.backup
├── malmo-2024-10-07T16-00-34.backup
├── malmo-2024-10-07T16-00-44.backup
├── malmo-2024-10-07T16-00-50.backup
├── malmo-2024-10-07T16-01-08.backup
├── malmo-2024-10-07T16-01-24.backup
└── neo4j-2024-10-07T16-05-37.backup
```

Listing the metadata of the backup files

The following command lists the backup files' names along with their respective metadata:

```
bin/neo4j-admin backup inspect /backups --show-metadata --empty
```

The `--empty` option is used to include the empty backups. An empty backup is created when a database is backed up but no new data exists. Empty backups are used to record the backup history.

Example output

```
|                                          FILE | DATABASE |                           DATABASE ID |
TIME (UTC) |  FULL | COMPRESSED | LOWEST TX | HIGHEST TX |
|  file:///backups/neo4j-2024-10-07T16-05-37.backup |    neo4j | 7dcb1d0c-4374-4476-b8ae-d3c3f124683f |
2024-10-07T16:05:37 |  true |       true |         1 |          3 |
|  file:///backups/malmo-2024-10-07T16-01-24.backup |    malmo | 62d1820c-3ac6-4b15-a0b3-bf7e7becc8d0 |
2024-10-07T16:01:24 |  true |       true |         1 |          8 |
|  file:///backups/malmo-2024-10-07T16-01-08.backup |    malmo | 62d1820c-3ac6-4b15-a0b3-bf7e7becc8d0 |
2024-10-07T16:01:08 |  true |       true |         1 |          7 |
|  file:///backups/malmo-2024-10-07T16-00-50.backup |    malmo | 62d1820c-3ac6-4b15-a0b3-bf7e7becc8d0 |
2024-10-07T16:00:50 | false |       true |         0 |          0 |
|  file:///backups/malmo-2024-10-07T16-00-44.backup |    malmo | 62d1820c-3ac6-4b15-a0b3-bf7e7becc8d0 |
2024-10-07T16:00:44 | false |       true |         7 |          7 |
|  file:///backups/malmo-2024-10-07T16-00-34.backup |    malmo | 62d1820c-3ac6-4b15-a0b3-bf7e7becc8d0 |
2024-10-07T16:00:34 | false |       true |         6 |          6 |
|  file:///backups/malmo-2024-10-07T16-00-19.backup |    malmo | 62d1820c-3ac6-4b15-a0b3-bf7e7becc8d0 |
2024-10-07T16:00:19 | false |       true |         0 |          0 |
|  file:///backups/malmo-2024-10-07T16-00-07.backup |    malmo | 62d1820c-3ac6-4b15-a0b3-bf7e7becc8d0 |
2024-10-07T16:00:07 |  true |       true |         1 |          5 |
| file:///backups/london-2024-10-07T16-04-05.backup |   london | d4dae73c-dfef-4d28-88cd-fe6cc88ddca1 |
2024-10-07T16:04:05 | false |       true |         6 |          6 |
| file:///backups/london-2024-10-07T16-03-51.backup |   london | d4dae73c-dfef-4d28-88cd-fe6cc88ddca1 |
2024-10-07T16:03:51 |  true |       true |         1 |          5 |
```

Listing the latest backups

To list only the most recent backups performed for each database, use the `--latest-backup` option.

```
bin/neo4j-admin backup inspect /backups --show-metadata --latest-backup
```

Example output

```
|                                          FILE | DATABASE |                           DATABASE ID |
TIME (UTC) |  FULL | COMPRESSED | LOWEST TX | HIGHEST TX |
|  file:///backups/neo4j-2024-10-07T16-05-37.backup |    neo4j | 7dcb1d0c-4374-4476-b8ae-d3c3f124683f |
2024-10-07T16:05:37 |  true |       true |         1 |          3 |
|  file:///backups/malmo-2024-10-07T16-01-24.backup |    malmo | 62d1820c-3ac6-4b15-a0b3-bf7e7becc8d0 |
2024-10-07T16:01:24 |  true |       true |         1 |          8 |
| file:///backups/london-2024-10-07T16-04-05.backup |   london | d4dae73c-dfef-4d28-88cd-fe6cc88ddca1 |
2024-10-07T16:04:05 | false |       true |         6 |          6 |
```

Inspecting backup chains

A backup chain corresponds to a sequence of one or more backup(s) logically connected by their transaction IDs. To inspect the backup chains of a given database, use the `--latest-chain` option and the `--database` option with the database whose backup chain you want to inspect:

```
bin/neo4j-admin backup inspect /backups --show-metadata --latest-chain --database=london
```

Example output

```
|                                          FILE | DATABASE |                           DATABASE ID |
TIME (UTC) |  FULL | COMPRESSED | LOWEST TX | HIGHEST TX |
| file:///backups/london-2024-10-07T16-04-05.backup |   london | d4dae73c-dfef-4d28-88cd-fe6cc88ddca1 |
2024-10-07T16:04:05 | false |       true |         6 |          6 |
| file:///backups/london-2024-10-07T16-03-51.backup |   london | d4dae73c-dfef-4d28-88cd-fe6cc88ddca1 |
2024-10-07T16:03:51 |  true |       true |         1 |          5 |
```

The result returns a chain of size two:

- The first backup is a full backup containing the store files within the transaction range [1,5].
- The second backup is a differential backup containing only the subsequent modifications to the store files. Those modifications are materialised by a sequence of transactions to apply. Its range is [6,6].

Inspecting a backup chain ending with a specific backup

To inspect a backup chain ending with a specific backup, use the `--latest-chain` option as follows:

```
bin/neo4j-admin backup inspect /backups/london-2024-10-07T16-04-05.backup  --show-metadata --latest-chain
```

Example output

```
|                                          FILE | DATABASE |                           DATABASE ID |
TIME (UTC) |  FULL | COMPRESSED | LOWEST TX | HIGHEST TX |
| file:///backups/london-2024-10-07T16-04-05.backup |   london | d4dae73c-dfef-4d28-88cd-fe6cc88ddca1 |
2024-10-07T16:04:05 | false |       true |         6 |          6 |
| file:///backups/london-2024-10-07T16-03-51.backup |   london | d4dae73c-dfef-4d28-88cd-fe6cc88ddca1 |
2024-10-07T16:03:51 |  true |       true |         1 |          5 |
```

> **Note:**
>
> In this case, the `--database` option is unnecessary because the database identifier is part of the metadata stored in the header of the backup file *london-2024-10-07T16-04-05.backup*.

Restore a database backup

Command

A database backup artifact (full or differential) can be restored within the same or to a later Neo4j version using the `restore` command of `neo4j-admin`.

Starting with Neo4j 5.20, you can load a **full** database backup artifact using the `neo4j-admin database load` command. This functionality is available in the Community Edition.

> **Note:**
>
> Restoring a database backup to a previous Neo4j version is not supported.

You must create the database (using `CREATE DATABASE` against the `system` database) after the restore operation finishes, unless you are replacing an existing database. `neo4j-admin database restore` must be invoked as the `neo4j` user to ensure the appropriate file permissions. For more information, see Create databases.

> **Note:**
>
> If you are using CDC, make sure you create the new database with the same `txLogEnrichment` value and handle the potential loss or corruption of CDC data in your CDC application. For more information, see the Change Data Capture (CDC) documentation.

> **Note:**
>
> When restoring a backup chain, the transaction log contained in the differential backup artifacts must first be replayed. This recovery operation is resource-intensive and can be decoupled from the restore operation by using the aggregate command.

Syntax

```
neo4j-admin database restore [-h] [--expand-commands]
                             [--verbose] [--overwrite-destination[=true|false]]
                             [--additional-config=<file>]
                             --from-path=<path>[,<path>...]
                             [--restore-until=<recovery-criteria>] [--temp-path=<path>]
                             [--to-path-data=<path>] [--to-path-txn=<path>]
                              [<database>]
```

Parameters

Table 412. `neo4j-admin database restore` parameters

Parameter	Description
`[<database>]`	Name of the database after restore. Usage of this parameter is only allowed if the `--from-path` option points to a path to a single artifact.

Options

Table 413. `neo4j-admin database restore` options

Option	Description	Default
`--additional-config=<file>`[4]	Configuration file with additional configuration.	
`--expand-commands`	Allow command expansion in config value evaluation.	
`--from-path=<path>[,<path>…]`	A single path or a comma-separated list of paths pointing to a backup artifact file. An artifact file can be 1) a full backup, in which case it is restored directly or, 2) a differential backup, in which case the command tries first to find in the folder a backup chain ending at that specific differential backup and then restores that chain. It is possible to restore backups from AWS S3 buckets, Google Cloud storage buckets, and Azure buckets using the appropriate URI as the path.	
`-h, --help`	Show this help message and exit.	
`--overwrite-destination[=true|false]`	If an existing database should be replaced.  This option is not safe on a cluster since clusters have an additional state that would be inconsistent with the restored database. In a cluster, restore to a new database to avoid this problem.	`false`

Option	Description	Default
`--restore-until=<recovery-criteria>`	Differential backup artifacts contain transaction logs that can be replayed and applied to stores contained in full backup artifacts when restoring a backup chain. The database applies logs until the recovery predicate is satisfied. Currently supported predicates are: `<transactionID>` and `<timestamp>`. • to restore a database up to a transaction ID, the required transaction predicate should look like `--restore-until=123`, where 123 is the provided transaction ID. The restore recovers transaction logs up to, but not including, the transaction 123. • to restore a database up to a specific date, the required date predicate should look like `--restore-until=2021-09-11 10:15:30`, where 2021-09-11 10:15:30 is a UTC date and time. The restore recovers transactions that were committed before the provided timestamp.	
`--to-path-data=<path>`	Base directory for databases. Usage of this option is only allowed if the `--from-path` parameter points to exactly one directory.	
`--to-path-txn=<path>`	Base directory for transaction logs. Usage of this option is only allowed if the `--from-path` parameter points to exactly one directory.	
`--temp-path=<path>`	Introduced in 5.24 Provide a path to a temporary empty directory for storing backup files until the command is completed. The files will be deleted once the command is finished.	
`--verbose`	Enable verbose output.	

Note:

The `--from-path=<path>` option can also load backups from AWS S3 buckets (from Neo4j 5.19), Google Cloud storage buckets (from Neo4j 5.21), and Azure buckets (from Neo4j 5.24). For more information, see Restore a backup located in a cloud storage.

Note:

Neo4j 5.24 introduces the `--temp-path` option to address potential issues related to disk space when performing backup-related commands, especially when cloud storage is involved.

If `--temp-path` is not set, a temporary directory is created inside the directory specified by the `--from-path` option.

If you don't provide the `--from-path` option or if your provided path points to a cloud storage bucket, a temporary folder is created inside the current working directory for Neo4j. This fallback option can cause issues because the local filesystem (or the partition where Neo4j is installed) may not have enough free disk to accommodate the intermediate computation.

Therefore, it is strongly recommended to provide a `--temp-path` option.

Examples

The following examples show how to inspect your backup directory and restore a database backup, created in the section Back up an online database. It is assumed that the backup artifacts (full and differential) are located in the */path/to/mybackups* directory.

Inspect the backup artifacts

Use the following command to inspect the backup directory:

```
bin/neo4j-admin database backup --inspect-path=/path/to/mybackups
```

Example output

```
|                                                          FILE | DATABASE |
DATABASE ID |                TIME |  FULL | COMPRESSED | LOWEST TX | HIGHEST TX |
| file:///path/to/mybackups/neo4j-2023-06-29T14-46-27.backup |    neo4j | c8368b24-55e2-474d-bb41-
75657f5bfcde | 2023-06-29T13:46:27 |  true |       true |         1 |         11 |
| file:///path/to/mybackups/neo4j-2023-06-29T14-50-45.backup |    neo4j | c8368b24-55e2-474d-bb41-
75657f5bfcde | 2023-06-29T13:50:45 | false |       true |        12 |         14 |
| file:///path/to/mybackups/neo4j-2023-06-29T14-51-33.backup |    neo4j | c8368b24-55e2-474d-bb41-
75657f5bfcde | 2023-06-29T13:51:33 | false |       true |        15 |         18 |
```

The example output shows that the backup artifacts are part of a backup chain. The first artifact is a full backup, and the other two are differential backups. The `LOWEST TX` and `HIGHEST TX` columns show the transaction IDs of the first and the last transaction in the backup artifacts. That means, if you restore `neo4j-2023-06-29T14-50-45.backup`, your database will have `14` as the last transaction ID.

Restore a database backup

The following examples assume that you want to restore your data in a new database, called `mydatabase`. If you want to replace an existing database, you need to stop it first, and add the option `--overwrite-destination=true` to the restore command.

1. Restore a database backup by running the following command:

   ```
   bin/neo4j-admin database restore --from-path=/path/to/backups/neo4j-2023-06-29T14-51-33.backup
   mydatabase
   ```

   The `--from-path=` argument must contain the path to the last backup of a chain, in this case, `neo4j-2023-06-29T14-51-33.backup`.

> **Tip:**
>
> If you want to restore several databases at once, you must stop them first and then you can alter the command by specifying a comma-separated list of paths to backup artifacts, and remove the `<database>` parameter. You should also skip the `CREATE DATABASE` step afterward if you are replacing an existing database.

2. Create the new database using `CREATE DATABASE` against the `system` database.

```
CREATE DATABASE mydatabase
```

Restore data up to a specific date

To restore data up to a specific date, you need to pass the backup artifact that contains the data up to that date.
This example assumes that you want to restore your data in a new database, called `mydatabase`. If you want to replace an existing database, you need to stop it first, and add the option `--overwrite -destination=true` to the restore command.

1. Restore from the backup that contains the data up to the desired date.

```
bin/neo4j-admin database restore --from-path=/path/to/mybackups/neo4j-2023-06-29T14-50-45.backup
--restore-until="2023-06-29 13:50:45" mydatabase
```

The `--from-path=` argument must contain the path to either a full or a differential backup artifact. The `--restore-until=` argument must contain a UTC date and time. The restore recovers all transactions that were committed before the provided date and time.

> **Tip:**
>
> If you want to restore several databases at once, you must stop them first and then you can alter the command by specifying a comma-separated list of paths to backup artifacts, and remove the `<database>` parameter. You should also skip the `CREATE DATABASE` step afterward if you are replacing an existing database.

> **Note:**
>
> If you know the transaction ID of the last transaction that was committed before the date you want to restore to, you can use the `--restore-until=` argument with the transaction ID instead of the date. For example, `--restore-until=123`.

2. Create the new database using `CREATE DATABASE` against the `system` database:

```
CREATE DATABASE mydatabase;
```

Restore a backup located in a cloud storage

The following examples show how to restore a database located in a cloud storage bucket using the `--from-path` option.

> **Note:**
>
> Neo4j uses the AWS SDK v2 to call the APIs on AWS using AWS URLs. Alternatively, you can override the endpoints so that the AWS SDK can communicate with alternative storage systems, such as Ceph, Minio, or LocalStack, using the system variables `aws.endpointUrls3`, `aws.endpointUrlS3`, or `aws.endpointUrl`, or the environments variables `AWS_ENDPOINT_URL_S3` or `AWS_ENDPOINT_URL`.

1. Install the AWS CLI by following the instructions in the AWS official documentation — Install the AWS CLI version 2.
2. Create an S3 bucket and a directory to store the backup files using the AWS CLI:

   ```
   aws s3 mb --region=us-east-1 s3://myBucket
   aws s3api put-object --bucket myBucket --key myDirectory/
   ```

   For more information on how to create a bucket and use the AWS CLI, see the AWS official documentation — Use Amazon S3 with the AWS CLI and Use high-level (s3) commands with the AWS CLI.
3. Verify that the `~/.aws/config` file is correct by running the following command:

   ```
   cat ~/.aws/config
   ```

   The output should look like this:

   ```
   [default]
   region=us-east-1
   ```
4. Configure the access to your AWS S3 bucket by setting the `aws_access_key_id` and `aws_secret_access_key` in the `~/.aws/credentials` file and, if needed, using a bucket policy. For example:
   a. Use `aws configure set aws_access_key_id aws_secret_access_key` command to set your IAM credentials from AWS and verify that the `~/.aws/credentials` is correct:

      ```
      cat ~/.aws/credentials
      ```

      The output should look like this:

      ```
      [default]
      ```

```
aws_access_key_id=this.is.secret
aws_secret_access_key=this.is.super.secret
```

b. Additionally, you can use a resource-based policy to grant access permissions to your S3 bucket and the objects in it. Create a policy document with the following content and attach it to the bucket. Note that both resource entries are important to be able to download and upload files.

```
{
    "Version": "2012-10-17",
    "Id": "Neo4jBackupAggregatePolicy",
    "Statement": [
        {
            "Sid": "Neo4jBackupAggregateStatement",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetObject",
                "s3:PutObject",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::myBucket/*",
                "arn:aws:s3:::myBucket"
            ]
        }
    ]
}
```

5. Run the `neo4j-admin database restore` command to restore the database located in your AWS S3 storage. The example assumes that you have backup artifacts located in the `myBucket/myDirectory` folder in your bucket.

```
bin/neo4j-admin database restore --from-path=s3://myBucket/myDirectory/myBackup.backup
mydatabase
```

1. Ensure you have a Google account and a project created in the Google Cloud Platform (GCP).

   a. Install the `gcloud` CLI by following the instructions in the Google official documentation — Install the gcloud CLI.

   b. Create a service account and a service account key using Google official documentation — Create service accounts and Creating and managing service account keys.

   c. Download the JSON key file for the service account.

   d. Set the `GOOGLE_APPLICATION_CREDENTIALS` and `GOOGLE_CLOUD_PROJECT` environment variables to the path of the JSON key file and the project ID, respectively:

```
export GOOGLE_APPLICATION_CREDENTIALS="/path/to/keyfile.json"
export GOOGLE_CLOUD_PROJECT=YOUR_PROJECT_ID
```

   e. Authenticate the `gcloud` CLI with the e-mail address of the service account you have created, the path to the JSON key file, and the project ID:

```
gcloud auth activate-service-account service-account@example.com --key-file
=$GOOGLE_APPLICATION_CREDENTIALS --project=$GOOGLE_CLOUD_PROJECT
```

For more information, see the Google official documentation — gcloud auth activate-service-account.

f. Create a bucket in the Google Cloud Storage using Google official documentation — Create buckets.

g. Verify that the bucket is created by running the following command:

```
gcloud storage ls
```

The output should list the created bucket.

2. Run the `neo4j-admin database restore` command to restore the database located in a given folder in your Google storage bucket. The example assumes that you have backup artifacts located in the `myBucket/myDirectory` folder in your bucket.

```
bin/neo4j-admin database restore --from-path=gs://myBucket/myDirectory/myBackup.backup
mydatabase
```

1. Ensure you have an Azure account, an Azure storage account, and a blob container.

   a. You can create a storage account using the Azure portal.
   For more information, see the Azure official documentation on Create a storage account.

   b. Create a blob container in the Azure portal.
   For more information, see the Azure official documentation on Quickstart: Upload, download, and list blobs with the Azure portal.

2. Install the Azure CLI by following the instructions in the Azure official documentation — Azure official documentation.

3. Authenticate the neo4j or neo4j-admin process against Azure using the default Azure credentials.
   See the Azure official documentation on default Azure credentials for more information.

```
az login
```

Then you should be ready to use Azure URLs in either neo4j or neo4j-admin.

4. To validate that you have access to the container with your login credentials, run the following commands:

```
# Upload a file:
az storage blob upload --file someLocalFile  --account-name accountName - --container
someContainer --name remoteFileName  --auth-mode login

# Download the file
az storage blob download  --account-name accountName --container someContainer --name
remoteFileName --file downloadedFile --auth-mode login
```

```
# List container files
az storage blob list  --account-name someContainer --container someContainer  --auth-mode
login
```

5. Run the `neo4j-admin database restore` command to restore the database located in a given folder in your Azure blob storage container. The example assumes that you have backup artifacts located in the `myStorageAccount/myContainer/myDirectory` folder.

```
bin/neo4j-admin database restore --from-path
=azb://myStorageAccount/myContainer/myDirectory/myBackup.backup mydatabase
```

Restore a database backup in a cluster

To restore a database backup in a cluster, designate one of the servers to be used as a seeder, and restore the database backup on that server. Then, use that server to create the restored database on other servers in the cluster. For more information, see Designated seeder.

Restore users and roles metadata

If you have backed up a database with the option `--include-metadata`, you can manually restore the users and roles metadata.

From the *<NEO4J_HOME>* directory, you run the Cypher script */data/scripts/databasename/restore_metadata.cypher*, which the `neo4j-admin database restore` command outputs, using Cypher Shell:

Using `cat` (UNIX)

```
cat ../data/scripts/databasename/restore_metadata.cypher | bin/cypher-shell -u user -p password -a
ip_address:port -d system --param "database => 'databasename'"
```

Using `type` (Windows)

```
type ..\data\scripts\databasename\restore_metadata.cypher | bin\cypher-shell.bat -u user -p password -a
ip_address:port -d system --param "database => 'databasename'"
```

Back up an offline database

Caution:

Remember to plan your backup carefully and back up each of your databases, including the `system` database.

Command

The `neo4j-admin database dump` command can be used for performing a full backup of an **offline** database.

It dumps only the database contents into a single-file archive, called <*database*>.*dump*, and stores it in the <*NEO4J_HOME*>/data directory.

Check the target directory permissions before running the `neo4j-admin database dump` command. The target directory must never be world-readable or world-executable. Even if files inside are restricted, a world-executable directory can leak structure. For more information, see File permissions.

Note that the `neo4j-admin database dump` command **does not** support backing up users and roles metadata.

Alternatively, `neo4j-admin database dump` can stream dump to standard output, enabling the output to be piped to another program, for example to `neo4j-admin database load`.

If the database is hosted in a cluster, make sure that the database is stopped on the server you are connected to. The command can be run only locally from an online or an offline Neo4j DBMS on Enterprise Edition. On Community Edition, the command can be run only on an offline Neo4j DBMS. It does not support SSL/TLS.

Syntax

```
neo4j-admin database dump [-h] [--expand-commands]
                          [--verbose] [--overwrite-destination[=true|false]]
                          [--additional-config=<file>]
                          [--to-path=<path> | --to-stdout]
                          <database>
```

Description

Dump a database into a single-file archive. The archive can be used by the load command. `<to-path>` should be a directory (in which case a file called <*database*>.*dump* will be created), or `--to-stdout` can be supplied to use standard output. If neither `--to-path` or `--to-stdout` is supplied `server.directories.dumps.root` setting will be used as a destination. It is not possible to dump a database that is mounted in a running Neo4j server.

Parameters

Table 414. `neo4j-admin database dump` *parameters*

Parameter	Description
<database>	Name of the database to dump. Can contain * and ? for globbing. Note that * and ? have special meaning in some shells and might need to be escaped or used with quotes.

Options

The `neo4j-admin database dump` command has the following options:

Table 415. `neo4j-admin database dump` *options*

Option	Description	Default
`--additional-config=<file>`[5]	Configuration file with additional configuration.	
`--expand-commands`	Allow command expansion in config value evaluation.	
`-h, --help`	Show this help message and exit.	
`--overwrite-destination[=true|false]`	Overwrite any existing dump file in the destination folder.	`false`
`--to-path=<path>`	Destination folder of a database dump. It is possible to dump databases into AWS S3 buckets, Google Cloud storage buckets, and Azure buckets using the appropriate URI as the path.	
`--to-stdout`	Use standard output as the destination for the database dump.	
`--verbose`	Enable verbose output.	

Note:

The `--to-path=<path>` option can also dump databases into AWS S3 buckets (from Neo4j 5.19), Google Cloud storage buckets (from Neo4j 5.21), and Azure buckets (from Neo4j 5.24). For more information, see Dump a database to a folder located in a cloud storage.

Examples

The following examples show how to dump a database using the `neo4j-admin database dump` command. The command creates a file called *database.dump* where `database` is the database specified in the command.

Note:

`neo4j-admin database dump` cannot be applied to Composite databases. It must be run directly on the databases that are associated with that Composite database.

Dump the default database `neo4j` to a local directory

You can use the following command to create a dump of the default database `neo4j` in a local directory. The target directory must exist before running the command and the database must be offline.

```
bin/neo4j-admin database dump neo4j --to-path=/full/path/to/dumps
```

Dump a database to a folder located in a cloud storage

The following examples show how to dump a database to a cloud storage bucket using the `--to-path`

option.

> **Note:**
>
> Neo4j uses the AWS SDK v2 to call the APIs on AWS using AWS URLs. Alternatively, you can override the endpoints so that the AWS SDK can communicate with alternative storage systems, such as Ceph, Minio, or LocalStack, using the system variables `aws.endpointUrls3`, `aws.endpointUrlS3`, or `aws.endpointUrl`, or the environments variables `AWS_ENDPOINT_URL_S3` or `AWS_ENDPOINT_URL`.

1. Install the AWS CLI by following the instructions in the AWS official documentation — Install the AWS CLI version 2.
2. Create an S3 bucket and a directory to store the backup files using the AWS CLI:

   ```
   aws s3 mb --region=us-east-1 s3://myBucket
   aws s3api put-object --bucket myBucket --key myDirectory/
   ```

   For more information on how to create a bucket and use the AWS CLI, see the AWS official documentation — Use Amazon S3 with the AWS CLI and Use high-level (s3) commands with the AWS CLI.

3. Verify that the `~/.aws/config` file is correct by running the following command:

   ```
   cat ~/.aws/config
   ```

   The output should look like this:

   ```
   [default]
   region=us-east-1
   ```

4. Configure the access to your AWS S3 bucket by setting the `aws_access_key_id` and `aws_secret_access_key` in the `~/.aws/credentials` file and, if needed, using a bucket policy. For example:

   a. Use `aws configure set aws_access_key_id aws_secret_access_key` command to set your IAM credentials from AWS and verify that the `~/.aws/credentials` is correct:

      ```
      cat ~/.aws/credentials
      ```

      The output should look like this:

      ```
      [default]
      aws_access_key_id=this.is.secret
      aws_secret_access_key=this.is.super.secret
      ```

   b. Additionally, you can use a resource-based policy to grant access permissions to your S3

bucket and the objects in it. Create a policy document with the following content and attach it to the bucket. Note that both resource entries are important to be able to download and upload files.

```
{
    "Version": "2012-10-17",
    "Id": "Neo4jBackupAggregatePolicy",
    "Statement": [
        {
            "Sid": "Neo4jBackupAggregateStatement",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetObject",
                "s3:PutObject",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::myBucket/*",
                "arn:aws:s3:::myBucket"
            ]
        }
    ]
}
```

5. Run `neo4j-admin database dump` command to dump your database into your AWS S3 bucket:

```
bin/neo4j-admin database dump mydatabase --to-path=s3://myBucket/myDirectory/
```

1. Ensure you have a Google account and a project created in the Google Cloud Platform (GCP).
   a. Install the `gcloud` CLI by following the instructions in the Google official documentation — Install the gcloud CLI.
   b. Create a service account and a service account key using Google official documentation — Create service accounts and Creating and managing service account keys.
   c. Download the JSON key file for the service account.
   d. Set the `GOOGLE_APPLICATION_CREDENTIALS` and `GOOGLE_CLOUD_PROJECT` environment variables to the path of the JSON key file and the project ID, respectively:

```
export GOOGLE_APPLICATION_CREDENTIALS="/path/to/keyfile.json"
export GOOGLE_CLOUD_PROJECT=YOUR_PROJECT_ID
```

   e. Authenticate the `gcloud` CLI with the e-mail address of the service account you have created, the path to the JSON key file, and the project ID:

```
gcloud auth activate-service-account service-account@example.com --key-file
=$GOOGLE_APPLICATION_CREDENTIALS --project=$GOOGLE_CLOUD_PROJECT
```

   For more information, see the Google official documentation — gcloud auth activate-

service-account.

f. Create a bucket in the Google Cloud Storage using Google official documentation — Create buckets.

g. Verify that the bucket is created by running the following command:

```
gcloud storage ls
```

The output should list the created bucket.

2. Run `neo4j-admin database dump` command to dump your database into your Google bucket:

```
bin/neo4j-admin database dump mydatabase --to-path=gs://myBucket/myDirectory/
```

1. Ensure you have an Azure account, an Azure storage account, and a blob container.

a. You can create a storage account using the Azure portal.
For more information, see the Azure official documentation on Create a storage account.

b. Create a blob container in the Azure portal.
For more information, see the Azure official documentation on Quickstart: Upload, download, and list blobs with the Azure portal.

2. Install the Azure CLI by following the instructions in the Azure official documentation — Azure official documentation.

3. Authenticate the neo4j or neo4j-admin process against Azure using the default Azure credentials.
See the Azure official documentation on default Azure credentials for more information.

```
az login
```

Then you should be ready to use Azure URLs in either neo4j or neo4j-admin.

4. To validate that you have access to the container with your login credentials, run the following commands:

```
# Upload a file:
az storage blob upload --file someLocalFile  --account-name accountName - --container
someContainer --name remoteFileName  --auth-mode login

# Download the file
az storage blob download  --account-name accountName --container someContainer --name
remoteFileName --file downloadedFile --auth-mode login

# List container files
az storage blob list  --account-name someContainer --container someContainer  --auth-mode
login
```

5. Run `neo4j-admin database dump` command to dump your database into your Azure container:

```
bin/neo4j-admin database dump mydatabase --to-path
=azb://myStorageAccount/myContainer/myDirectory/
```

Restore a database dump

The `neo4j-admin database load` command can be used to load a database from an archive created with the `neo4j-admin database dump` command.
Starting from Neo4j 5.20, the `neo4j-admin database load` command also supports loading a full backup artifact created by the `neo4j-admin database backup` command from Neo4j Enterprise.

If you are replacing an existing database, you have to shut it down before running the command and use the `--overwrite-destination` option.
Enterprise Edition If you are not replacing an existing database, you must create the database (using `CREATE DATABASE` against the `system` database) after the load operation finishes.

The command can be run from either an online or offline Neo4j DBMS on Enterprise edition. For Community edition, the command can be run only on an offline Neo4j DBMS. The command must be executed as the `neo4j` user to ensure the appropriate file permissions.

Note:

Change Data Capture does **not** capture any data changes resulting from the use of `neo4j-admin database load`. See Change Data Capture → Key considerations for more information.

Syntax

```
neo4j-admin database load [-h] [--expand-commands] [--info] [--verbose] [--overwrite-
destination[=true|false]]
                          [--additional-config=<file>] [--from-path=<path> | --from-stdin] <database>
```

Description

Load a database from an archive. *<archive-path>* must be a directory containing an archive(s). Archive can be a database dump created with the dump command, or can be a full backup artifact created by the backup command from Neo4j Enterprise. If neither `--from-path` or `--from-stdin` is supplied `server.directories.dumps.root` setting will be searched for the archive. Existing databases can be replaced by specifying `--overwrite-destination`. It is not possible to replace a database that is mounted in a running Neo4j server. If `--info` is specified, then the database is not loaded, but information (i.e. file count, byte count, and format of load file) about the archive is printed instead.

Parameters

Table 416. `neo4j-admin database load` *parameters*

Parameter	Description
`<database>`	Name of the database to load. Can contain * and ? for globbing. Note that * and ? have special meaning in some shells and might need to be escaped or used with quotes.

Options

Table 417. `neo4j-admin database load` options

Option	Description	Default
`--additional-config=<file>`[6]	Configuration file with additional configuration.	
`--expand-commands`	Allow command expansion in config value evaluation.	
`--from-path=<path>`	Path to directory containing archive(s). It is possible to load databases from AWS S3 buckets, Google Cloud storage buckets, and Azure bucket using the appropriate URI as the path.	
`--from-stdin`	Read archive from standard input.	
`-h, --help`	Show this help message and exit.	
`--info`	Print meta-data information about the archive file, instead of loading the contained database.	
`--overwrite -destination[=true|false]`	If an existing database should be replaced.	`false`
`--verbose`	Enable verbose output.	

Note:

The `--from-path=<path>` option can also load databases from AWS S3 buckets (from Neo4j 5.19), Google Cloud storage buckets (from Neo4j 5.21), and Azure buckets (from Neo4j 5.24). For more information, see Load a dump from a cloud storage.

Examples

The following are examples of how to load a dump of a database (*database.dump*) created in the section Back up an offline database, using the `neo4j-admin database load` command. When replacing an existing database, you have to shut it down before running the command. The `--overwrite-destination` option is required because you are replacing an existing database.

If you are not replacing an existing database, you must create the database (using `CREATE DATABASE` against the `system` database) after the load operation finishes.

Tip:

The command looks for a file called <database>.dump where `<database>` is the database specified in the command.

As of Neo4j 5.20, the command also looks for a full backup artifact.

Note:

When using the `load` command to seed a cluster, and a previous version of the database exists, you must delete it (using `DROP DATABASE`) first. Alternatively, you can stop the Neo4j instance and unbind it from the cluster using `neo4j-admin server unbind` to remove its cluster state data. If you fail to DROP or unbind before loading the dump, that database's store files will be out of sync with its cluster state, potentially leading to logical corruptions. For more information, see Seed a cluster.

Note:

`neo4j-admin database load` cannot be applied to Composite databases. It must be run directly on the databases that are associated with that Composite database.

Load a dump from a local directory

You can load a dump from a local directory using the following command:

```
bin/neo4j-admin database load --from-path=/full-path/data/dumps neo4j --overwrite-destination=true
```

Starting from Neo4j 5.20, you can use the same command to load the database from its full backup artifact:

```
bin/neo4j-admin database load --from-path=/full-path/to/backups neo4j --overwrite-destination=true
```

The following example shows how to designate a specific archive for the `load` command.

```
cat foo.dump | neo4j-admin database load --from-stdin mydatabase
```

Load a dump from a cloud storage

The following examples show how to load a database dump located in a cloud storage bucket using the `--from-path` option.

Note:

Neo4j uses the AWS SDK v2 to call the APIs on AWS using AWS URLs. Alternatively, you can override the endpoints so that the AWS SDK can communicate with alternative storage systems, such as Ceph, Minio, or LocalStack, using the system variables `aws.endpointUrls3`, `aws.endpointUrlS3`, or `aws.endpointUrl`, or the environments variables `AWS_ENDPOINT_URL_S3` or

`AWS_ENDPOINT_URL`.

1. Install the AWS CLI by following the instructions in the AWS official documentation — Install the AWS CLI version 2.
2. Create an S3 bucket and a directory to store the backup files using the AWS CLI:

   ```
   aws s3 mb --region=us-east-1 s3://myBucket
   aws s3api put-object --bucket myBucket --key myDirectory/
   ```

   For more information on how to create a bucket and use the AWS CLI, see the AWS official documentation — Use Amazon S3 with the AWS CLI and Use high-level (s3) commands with the AWS CLI.

3. Verify that the `~/.aws/config` file is correct by running the following command:

   ```
   cat ~/.aws/config
   ```

   The output should look like this:

   ```
   [default]
   region=us-east-1
   ```

4. Configure the access to your AWS S3 bucket by setting the `aws_access_key_id` and `aws_secret_access_key` in the `~/.aws/credentials` file and, if needed, using a bucket policy. For example:

   a. Use `aws configure set aws_access_key_id aws_secret_access_key` command to set your IAM credentials from AWS and verify that the `~/.aws/credentials` is correct:

      ```
      cat ~/.aws/credentials
      ```

      The output should look like this:

      ```
      [default]
      aws_access_key_id=this.is.secret
      aws_secret_access_key=this.is.super.secret
      ```

   b. Additionally, you can use a resource-based policy to grant access permissions to your S3 bucket and the objects in it. Create a policy document with the following content and attach it to the bucket. Note that both resource entries are important to be able to download and upload files.

      ```
      {
          "Version": "2012-10-17",
          "Id": "Neo4jBackupAggregatePolicy",
          "Statement": [
              {
                  "Sid": "Neo4jBackupAggregateStatement",
                  "Effect": "Allow",
                  "Action": [
                      "s3:ListBucket",
                      "s3:GetObject",
      ```

```
                "s3:PutObject",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::myBucket/*",
                "arn:aws:s3:::myBucket"
            ]
        }
    ]
}
```

5. Run the `neo4j-admin database load` command to load a dump from your AWS S3 storage. The example assumes that you have dump artifacts located in the `myBucket/myDirectory` folder in your bucket.

```
bin/neo4j-admin database load mydatabase --from-path=s3://myBucket/myDirectory/ --overwrite
-destination=true
```

1. Ensure you have a Google account and a project created in the Google Cloud Platform (GCP).
   a. Install the `gcloud` CLI by following the instructions in the Google official documentation — Install the gcloud CLI.
   b. Create a service account and a service account key using Google official documentation — Create service accounts and Creating and managing service account keys.
   c. Download the JSON key file for the service account.
   d. Set the `GOOGLE_APPLICATION_CREDENTIALS` and `GOOGLE_CLOUD_PROJECT` environment variables to the path of the JSON key file and the project ID, respectively:

```
export GOOGLE_APPLICATION_CREDENTIALS="/path/to/keyfile.json"
export GOOGLE_CLOUD_PROJECT=YOUR_PROJECT_ID
```

   e. Authenticate the `gcloud` CLI with the e-mail address of the service account you have created, the path to the JSON key file, and the project ID:

```
gcloud auth activate-service-account service-account@example.com --key-file
=$GOOGLE_APPLICATION_CREDENTIALS --project=$GOOGLE_CLOUD_PROJECT
```

   For more information, see the Google official documentation — gcloud auth activate-service-account.

   f. Create a bucket in the Google Cloud Storage using Google official documentation — Create buckets.
   g. Verify that the bucket is created by running the following command:

```
gcloud storage ls
```

   The output should list the created bucket.

2. Run the `neo4j-admin database load` command to load a dump from your Google storage bucket. The example assumes that you have dump artifacts located in the `myBucket/myDirectory` folder in your bucket.

```
bin/neo4j-admin database load mydatabase --from-path=gs://myBucket/myDirectory/ --overwrite
-destination=true
```

1. Ensure you have an Azure account, an Azure storage account, and a blob container.
   a. You can create a storage account using the Azure portal.
      For more information, see the Azure official documentation on Create a storage account.
   b. Create a blob container in the Azure portal.
      For more information, see the Azure official documentation on Quickstart: Upload, download, and list blobs with the Azure portal.
2. Install the Azure CLI by following the instructions in the Azure official documentation — Azure official documentation.
3. Authenticate the neo4j or neo4j-admin process against Azure using the default Azure credentials.
   See the Azure official documentation on default Azure credentials for more information.

```
az login
```

   Then you should be ready to use Azure URLs in either neo4j or neo4j-admin.

4. To validate that you have access to the container with your login credentials, run the following commands:

```
# Upload a file:
az storage blob upload --file someLocalFile  --account-name accountName - --container
someContainer --name remoteFileName  --auth-mode login

# Download the file
az storage blob download  --account-name accountName --container someContainer --name
remoteFileName --file downloadedFile --auth-mode login

# List container files
az storage blob list  --account-name someContainer --container someContainer  --auth-mode
login
```

5. Run the `neo4j-admin database load` command to load a dump from your Azure blob storage container. The example assumes that you have dump artifacts located in the `myStorageAccount/myContainer/myDirectory` folder in your Azure account.

```
bin/neo4j-admin database load mydatabase --from-path
=azb://myStorageAccount/myContainer/myDirectory --overwrite-destination=true
```

Copy a database store

You can use the `neo4j-admin database copy` command to copy a database, create a compacted/defragmented copy of a database, clean up database inconsistencies, or do a direct migration from Neo4j 4.4 to any 5.x version. `neo4j-admin database copy` reclaims the unused space, creates a defragmented copy of the data store, and creates the node label and relationship type lookup indexes.

Command limitations

Note:

- `neo4j-admin database copy` preserves the node IDs (unless `--compact-node-store` is used), but the relationships get new IDs.
- `neo4j-admin database copy` is not supported for use on the `system` database.
- `neo4j-admin database copy` is not supported for use on Composite databases. It must be run directly on the databases that are associated with that Composite database.
- `neo4j-admin database copy` is an IOPS-intensive process. For more information, see Estimating the processing time.

Command

`neo4j-admin database copy` copies the data store of an existing **offline** database to a new database.

Syntax

```
neo4j-admin database copy [-h] [--copy-schema] [--expand-commands] [--force] [--verbose] [--compact-node-
store[=true|false]]
                          [--additional-config=<file>] [--from-pagecache=<size>] [--temp-path=<path>]
[--to-format=<format>]
                          [--to-path-schema=<path>] [--copy-only-node-
properties=<label.property>[,<label.property>...]]...
                          [--copy-only-nodes-with-labels=<label>[,<label>...]]... [--copy-only-
relationship-properties=<relationship.
                          property>[,<relationship.property>...]]... [--copy-only-relationships-with-
types=<type>[,<type>...]]...
                          [--ignore-nodes-with-labels=<label>[,<label>...]]... [--ignore-relationships-
with-types=<type>[,<type>...]]...
                          [--skip-labels=<label>[,<label>...]]... [--skip-node-
properties=<label.property>[,<label.property>...]]...
                          [--skip-properties=<property>[,<property>...]]... [--skip-relationship-
properties=<relationship.property>[,
                          <relationship.property>...]]... [--from-path-data=<path> --from-path
-txn=<path>] [--to-path-data=<path>
                          --to-path-txn=<path>] <fromDatabase> <toDatabase>
```

Description

This command will create a copy of a database. If your labels, properties, or relationships contain dots or commas, you can use backticks to quote them, e.g. `My,label`, `My.property`. A file named *<database-name>-schema.cypher*, containing the schema commands needed to recreate indexes/constraints on the copy, will be created.

From Neo4j 5.20 onwards, you can use the `--copy-schema` option to automatically copy the schema. Indexes will be built the first time the database is started. This option can copy the schema from any 4.4 and 5.x version to 5.20 and later versions.

Parameters

Table 418. `neo4j-admin database copy` parameters

Parameter	Description
`<fromDatabase>`	Name of the source database.
`<toDatabase>`	Name of the target database. If the same as `<fromDatabase>`, it is copied to a temporary location, by default the current working directory or the path as defined by `--temp-path`, before being moved to replace the original.

From Neo4j 5.5, you can use the same values for `<fromDatabase>` and `<toDatabase>` if you do not need an actual copy of the database. The command will replace the original database with the newly created copy.

Options

The `neo4j-admin database copy` command has the following options:

Table 419. `neo4j-admin database copy` options

Option	Description	Default
`--additional-config=<file>`[7]	Configuration file with additional configuration.	
`--compact-node-store[=true|false]`	By default node store is not compacted on copy since that changes node ids. Please use this option to enforce node store compaction.	`false`
`--copy-only-node-properties=<label.property>[,<label.property>…]`	A comma-separated list of property keys to include in the copy for nodes with the specified label. Any labels not explicitly mentioned will have all their properties included in the copy. Cannot be combined with `--skip-properties` or `--skip-node-properties`.	
`--copy-only-nodes-with-labels=<label>[,<label>…]`	A comma-separated list of labels. All nodes that have ANY of the specified labels will be included in the copy. Cannot be combined with `--ignore-nodes-with-labels`.	
`--copy-only-relationship-properties=<relationship.property>[,<relationship.property>…]`	A comma-separated list of property keys to include in the copy for relationships with the specified type. Any relationship types not explicitly mentioned will have all their properties included in the copy. Cannot be combined with `--skip-properties` or `--skip-relationship-properties`.	
`--copy-only-relationships-with-types=<type>[,<type>…]`	A comma-separated list of relationship types. All relationships with any of the specified types will be included in the copy. Cannot be combined with `--ignore-relationships-with-types`.	

Option	Description	Default
`--copy-schema`	Introduced in 5.20 Copy the schema instead of generating schema statements, meaning index and constraint definitions. The indexes will be built the first time the database is started.	
`--expand-commands`	Allow command expansion in config value evaluation.	
`--force`	Force the command to run even if the integrity of the database cannot be verified.	
`--from-pagecache=<size>`	The size of the page cache to use for reading.  You can use the `--from-pagecache` option to speed up the copy operation by specifying how much cache to allocate when reading the source. The `--from-pagecache` should be assigned whatever memory you can spare since Neo4j does random reads from the source.	`8m`
`--from-path-data=<path>`	Path to the databases directory, containing the database directory to source from. It can be used to target databases outside of the installation.	`server.directories.data/databases`
`--from-path-txn=<path>`	Path to the transactions directory, containing the transaction directory for the database to source from.	`server.directories.transaction.logs.root`
`-h, --help`	Show this help message and exit.	
`--ignore-nodes-with-labels=<label>[,<label>…]`	A comma-separated list of labels. Nodes that have ANY of the specified labels will not be included in the copy. Cannot be combined with `--copy-only-nodes-with-labels`.	
`--ignore-relationships-with-types=<type>[,<type>…]`	A comma-separated list of relationship types. Relationships with any of the specified relationship types will not be included in the copy. Cannot be combined with `--copy-only-relationships-with-types`.	
`--skip-labels=<label>[,<label>…]`	A comma-separated list of labels to ignore.	
`--skip-node-properties=<label.property>[,<label.property>…]`	A comma-separated list of property keys to ignore for nodes with the specified label. Cannot be combined with `--skip-properties` or `--copy-only-node-properties`.	

Option	Description	Default
`--skip-properties=<property>[,<property>…]`	A comma-separated list of property keys to ignore. Cannot be combined with `--skip-node-properties`, `--copy-only-node-properties`, `--skip-relationship-properties` or `--copy-only-relationship-properties`.	
`--skip-relationship-properties=<relationship.property>[,<relationship.property>…]`	A comma-separated list of property keys to ignore for relationships with the specified type. Cannot be combined with `--skip-properties` or `--copy-only-relationship-properties`.	
`--temp-path=<path>`	Introduced in 5.24 Path to a directory to be used as a staging area when the source and target databases are the same. Default is the current directory.	
`--to-format=<format>`	Set the format for the new database. Must be one of `same`, `standard`, `high_limit`, `aligned`, `block`. `same` will use the same format as the source. If you go from `high_limit` to `standard` or `aligned`, there is no validation that the data will actually fit.	`same`
`--to-path-data=<path>`	Path to the databases directory, containing the database directory to target from.	`server.directories.data/databases`
`--to-path-schema=<path>`	Path to directory to create the schema commands file in. Default is the current directory.	
`--to-path-txn=<path>`	Path to the transactions directory containing the transaction directory for the database to target from.	`server.directories.transaction.logs.root`
`--verbose`	Enable verbose output.	

Note:

The block format is introduced in Neo4j 5.14 and from Neo4j 5.22, is the default format for all newly-created databases as long as they do not have the `db.format` setting specified. For more information on the block format, see Store formats.

Examples

Copying the data store of a database

You can use `neo4j-admin database copy` to copy the data store of a database, for example, `neo4j`.

1. Stop the database named `neo4j`:

```
STOP DATABASE neo4j
```

2. Copy the data store from `neo4j` to a new database called `database-copy`.

> **Tip:**
>
> If you do not need an actual copy of the database, you can use the same values for `<fromDatabase>` and `<toDatabase>`. The command replaces the original database with the newly created copy.
>
> From Neo4j 5.20 onwards, you can use the `--copy-schema` option to automatically copy the schema. Indexes will be built the first time the database is started. This option copies the schema from any 4.4 and 5.x version to 5.20 and later versions.
>
> For previous versions, you need to manually recreate the schema using the Cyher statements saved in the file *<database-name>-schema.cypher*.

```
bin/neo4j-admin database copy neo4j database-copy
```

3. Verify that the database has been successfully copied:

```
ls -al ../data/databases
```

> **Note:**
>
> Copying a database does not automatically create it. Therefore, it will not be visible if you do `SHOW DATABASES` at this point.

4. Create the copied database.

```
CREATE DATABASE database-copy
```

5. Verify that the new database is online.

```
SHOW DATABASES
```

6. (For versions before Neo4j 5.20) If your original database has a schema defined, change your active database to the copied database and recreate the schema using the schema commands saved in the file *<database-name>-schema.cypher*.

Filtering data while copying a database

You can use `neo4j-admin database copy` to filter out any unwanted data while copying a database, for example, by removing nodes, labels, properties, and relationships.

```
bin/neo4j-admin database copy neo4j copy --ignore-nodes-with-labels="Cat,Dog"
```

The command creates a copy of the database `neo4j` but without the nodes with the labels `:Cat` and `:Dog`.

> **Note:**
>
> Labels are processed independently, i.e., the filter ignores any node with a label `:Cat`, `:Dog`, or both.

> **Tip:**
>
> For a detailed example of how to use `neo4j-admin database copy` to filter out data for sharding a database, see Sharding data with the `copy` command.

Further compacting an existing database

You can use the command `neo4j-admin database copy` with the argument `-compact-node-store` to further compact the store of an existing database.
This example uses the same values for `<toDatabase>` and `<fromDatabase>`, which means that the command will compact the database in place by creating a new version of the database. After running the command, you need to recreate the indexes using the generated script. If the database belongs to a cluster, you also need to reseed the cluster from that server. For more information, see Designated seeder.

> **Note:**
>
> Note that even though there is only one database copy in the end, you still need double the space during the operation.

1. Stop the database named `neo4j`:

   ```
   STOP DATABASE neo4j
   ```

2. Compact the `neo4j` database using the command:

   ```
   bin/neo4j-admin database copy neo4j neo4j --compact-node-store --temp-path=<my-prefered-staging-area>
   ```

   > **Tip:**
   >
   > `--temp-path`, introduced in Neo4j 5.24, can be used to specify a different directory to use as a temporary staging area. If omitted, the current working directory will be used.
   >
   > From Neo4j 5.20 onwards, you can use the `--copy-schema` option to automatically copy the schema. Indexes will be built the first time the database is started. This option can copy the schema from any 4.4 and 5.x to 5.20 and later versions.

For previous versions, you need to manually recreate the schema using the Cyher statements saved in the file *<database-name>-schema.cypher*.

3. Start the `neo4j` database. This is the newly created version of the database.

```
START DATABASE neo4j
```

4. (For versions before Neo4j 5.20) If your original database has a schema defined, recreate the schema using the schema commands saved in the file *<database-name>-schema.cypher*.

Tip:

For a detailed example of how to reclaim unused space, see Reclaim unused space.

Estimating the processing time

Estimations for how long the `neo4j-admin database copy` command takes can be made based on the following:

- Neo4j, like many other databases, does IO in 8K pages.
- Your disc manufacturer will have a value for the maximum IOPS it can process.

For example, if your disc manufacturer has provided a maximum of 5000 IOPS, you can reasonably expect up to 5000 such page operations a second. Therefore, the maximal theoretical throughput you can expect is 40MB/s (or 144 GB/hour) on that disc. You may then assume that the best-case scenario for running `neo4j-admin database copy` on that 5000 IOPS disc is that it takes at least 1 hour to process a 144 GB database. [8]

However, it is important to remember that the process must read 144 GB from the source database, and must also write to the target store (assuming the target store is of comparable size). Additionally, there are internal processes during the copy that reads/modifies/writes the store multiple times. Therefore, with an additional 144 GB of both read and write, the best-case scenario for running `neo4j-admin database copy` on a 5000 IOPS disc is that it takes **at least 3 hours to process a 144 GB database**.

Finally, it is also important to consider that in almost all Cloud environments, the published IOPS value may not be the same as the actual value, or be able to continuously maintain the maximum possible IOPS. The real processing time for this example *could* be well above that estimation of 3 hours.

[1] See Tools → Configuration for details.
[2] See Tools → Configuration for details.
[3] See Tools → Configuration for details.
[4] See Tools → Configuration for details.
[5] See Tools → Configuration for details.
[6] See Tools → Configuration for details.
[7] See Tools → Configuration for details.
[8] The calculations are based on `MB/s = (IOPS * B) ÷ 10^6`, where `B` is the block size in bytes; in the case of Neo4j, this is `8000`. GB/hour can then be calculated from `(MB/s * 3600) ÷ 1000`.

Authentication and authorization

This page provides an overview of authentication and authorization in Neo4j.

Authentication

Authentication is the process of verifying the identity of a user. Neo4j has the following authentication (auth) providers that can perform user and role authentication:

Native auth provider

Neo4j provides a native auth provider that stores user and role information in the `system` database. The following parameters control this provider:

- `dbms.security.auth_enabled` (Default: `true`) — Enable auth requirement to access Neo4j.

  If you need to disable authentication, make sure you block all network connections during the recovery phase so users can connect to Neo4j only via *localhost*. This is necessary if, for example, you need to recover an `admin` user password or assign a user to the `admin` role. For more information, see Password and user recovery.

- `dbms.security.auth_lock_time` (Default: `5s`) — The amount of time a user account is locked after a configured number of unsuccessful authentication attempts.
- `dbms.security.auth_max_failed_attempts` (Default: `3`) — The maximum number of unsuccessful authentication attempts before imposing a user lock for a configured amount of time.
  When triggered, Neo4j logs an error containing a timestamp and the message `failed to log in: too many failed attempts` in the *security.log*.

For the relevant Cypher commands, see Manage users syntax, Manage roles syntax, and Manage privileges syntax. Various scenarios that illustrate the use of the native auth provider are available in Fine-grained access control.

User auth providers

User auth providers allow you to link externally-defined users (e.g., in a third-party ID provider like OIDC or LDAP) to the Neo4j internal user model. For more information, see User auth providers.

LDAP auth provider

Controls authentication and authorization through external security software such as Active Directory or OpenLDAP, which is accessed via the built-in LDAP connector. A description of the LDAP plugin using Active Directory is available in Integration with LDAP directory services.

Single sign-on provider

Integration with a single sign-on service, such as Okta, Auth0, or Microsoft Entra ID to provide centralized authentication and authorization for all your systems. Neo4j supports the popular OpenID

Connect mechanism for integrating with identity providers. The configuration steps are described in Single sign-on integration.

Custom-built plugin auth providers

A plugin option for building custom integrations. It is recommended that this option is used as part of a custom delivery as negotiated with Neo4j Professional Services. For more information, see Java Reference → Authentication and authorization plugins.

Kerberos authentication and single sign-on

In addition to LDAP, native, and custom providers, Neo4j supports Kerberos for authentication and single sign-on. Kerberos support is provided via the Neo4j Kerberos Add-On.

Mixed-mode authentication

Neo4j also supports mixed-mode authentication that allows you to use multiple authentication providers in your database setup. For more information and examples, see Set Neo4j to use LDAP and Configure Neo4j to use OpenID Connect.

Authorization

Authorization is the process of determining whether a user is allowed to perform a specific action. Authorization is managed using role-based access control (*RBAC*). RBAC is a method of restricting access to authorized users. It is a way of assigning privileges to roles that are then assigned to users. This simplifies user management, as permissions are assigned to roles rather than to individual users. The roles are defined in terms of their underlying *privileges*, and they can be modified by adding or removing these access rights using the Cypher commands described in this chapter.

Neo4j provides a set of built-in roles and also allows you to create custom roles with specific privileges. You can also use the *sub-graph* access control, through which read access to the graph can be limited to specific combinations of labels, relationship types, and properties.

Note:

The functionality described in these pages applies to Enterprise Edition. A limited set of user management functions are also available in Community Edition. Built-in roles capabilities gives a quick overview of these.

The Neo4j security model is stored in the system graph, which is maintained in the `system` database. All administrative commands need to be executed against it. When connected to the DBMS over Configure network connectors, administrative commands are automatically routed to the `system` database.

Terminology

The following terms are relevant to role-based access control within Neo4j:

active user

A user who is active within the system and can perform actions prescribed by any assigned roles on the data. This is in contrast to a suspended user.

administrator

This is a user who has been assigned the admin role.

auth provider

Properties attached to a user which define which authentication and authorization config to use for that user.

authentication

The process of verifying the identity of a user, typically using credentials like a username and password or a cryptographic token like a JWT.

authorization

The process of determining a user's access rights and privileges within Neo4j, based on their verified identity.

current user

This is the currently logged-in user invoking the commands.

password policy

The password policy is a set of rules about what makes up a valid password. For Neo4j, the following rules apply:

- The password cannot be an empty string.
- When changing passwords, the new password cannot be the same as the previous password.
- The password must be at least 8 characters long.

role

A collection of privileges that enables users to perform specific actions on the data. A user can have multiple roles.

suspended user

A user who has been suspended is not able to access the database in any capacity, regardless of any assigned roles.

user

- A user is composed of a username and credentials, where the latter is a unit of information, such as a password, verifying the identity of a user.
- A user may represent a human, an application, etc.

Manage users

Users can be created and managed using a set of Cypher administration commands executed against the `system` database. When connected to the DBMS over `bolt`, administration commands are automatically routed to the `system` database.

User states

There are two types of user states in the `system` database:

`ACTIVE` state

(default for new users) Users can log into Neo4j and perform queries according to their privileges.

Enterprise Edition

`SUSPENDED` state Enterprise Edition

- Native users who authenticate and authorize against the system graph cannot log into Neo4j. If suspended while using Neo4j, they lose all assigned roles with their privileges, including the `PUBLIC` role, until reactivated.
- Users who authenticate and authorize against an external ID provider (e.g., LDAP) can still log in. If suspended while using Neo4j, they retain the roles and the privileges assigned by the external provider, including the `PUBLIC` role. To prevent any of these, you need to use the mechanisms of their identity provider.

User management command syntax

Note:

For more details about the syntax descriptions, see Database management command syntax.

Command	`SHOW CURRENT USER`

Syntax	`SHOW CURRENT USER`
	`  [YIELD { * | field[, ...] } [ORDER BY field[, ...]] [SKIP n] [LIMIT n]]`
	`  [WHERE expression]`
	`  [RETURN field[, ...] [ORDER BY field[, ...]] [SKIP n] [LIMIT n]]`

Description	Lists the current user. When using the `RETURN` clause, the `YIELD` clause is mandatory and must not be omitted. For more information, see Listing current user.
Required privilege	None

Command	`SHOW USERS`

Syntax	`SHOW USER[S]`
	`  [WITH AUTH]`
	`  [YIELD { * | field[, ...] } [ORDER BY field[, ...]] [SKIP n] [LIMIT n]]`
	`  [WHERE expression]`
	`  [RETURN field[, ...] [ORDER BY field[, ...]] [SKIP n] [LIMIT n]]`

Description	Lists all users. When using the `RETURN` clause, the `YIELD` clause is mandatory and must not be omitted. For more information, see Listing users.
Required privilege	`GRANT SHOW USER` For more information, see DBMS USER MANAGEMENT privileges.

Command	`SHOW USER PRIVILEGES`

Syntax	`SHOW USER[S] [name[, ...]] PRIVILEGE[S] [AS [REVOKE] COMMAND[S]]`
	`  [YIELD { * | field[, ...] } [ORDER BY field[, ...]] [SKIP n] [LIMIT n]]`
	`  [WHERE expression]`
	`  [RETURN field[, ...] [ORDER BY field[, ...]] [SKIP n] [LIMIT n]]`

Description	Lists the privileges granted to the specified users or the current user if no user is specified. When using the `RETURN` clause, the `YIELD` clause is mandatory and must not be omitted. The `SHOW USER name PRIVILEGES` command is described in Listing privileges.

Required privilege	`GRANT SHOW PRIVILEGE`
	
	For more information, see DBMS PRIVILEGE MANAGEMENT privileges.
	
	`GRANT SHOW USER`
	
	For more information, see DBMS USER MANAGEMENT privileges.

Command	`CREATE USER`

Syntax	`CREATE USER name [IF NOT EXISTS]`
	`  [SET [PLAINTEXT | ENCRYPTED] PASSWORD 'password']`
	`  [[SET PASSWORD] CHANGE [NOT] REQUIRED]`
	`  [SET STATUS {ACTIVE | SUSPENDED}]`
	`  [SET HOME DATABASE name]`
	`  [SET AUTH [PROVIDER] 'provider' "{"{SET <key> <value>}..."}"]... # Introduced in Neo4j 5.24`
	
	`<key><value>` pairs for the `SET AUTH` clause could include:
	
	`SET AUTH [PROVIDER] 'provider' "{"`
	`  {`
	`  SET ID 'id' # a unique identifier of the user in an external system`
	`  | SET [PLAINTEXT | ENCRYPTED] PASSWORD 'password' # only applicable to the 'native' provider`
	`  | SET PASSWORD CHANGE [NOT] REQUIRED # only applicable to the 'native' provider`
	`  }`
	`"}"`

Description	Creates a new user. Note: It is mandatory to specify a `SET PASSWORD` and/or at least one `SET AUTH` clause because users must have at least one auth provider. For more information, see Creating users.
Required privilege	`GRANT CREATE USER` For more information, see Grant privilege to create users.

Command	`CREATE OR REPLACE USER`

Syntax	`CREATE OR REPLACE USER name [SET [PLAINTEXT | ENCRYPTED] PASSWORD 'password'] [[SET PASSWORD] CHANGE [NOT] REQUIRED] [SET STATUS {ACTIVE | SUSPENDED}] [SET HOME DATABASE name] [SET AUTH [PROVIDER] 'provider' "{"{SET <key> <value>}..."}"]... # Introduced in Neo4j 5.24`
	
	`<key><value>` pairs for the `SET AUTH` clause could include:
	
	`SET AUTH [PROVIDER] 'provider' "{" { SET ID 'id' # a unique identifier of the user in an external system | SET [PLAINTEXT | ENCRYPTED] PASSWORD 'password' # only applicable to the 'native' provider | SET PASSWORD CHANGE [NOT] REQUIRED # only applicable to the 'native' provider } "}"`

Description	Creates a new user, or if a user with the same name exists, replace it. Note: It is mandatory to specify a `SET PASSWORD` and/or at least one `SET AUTH` clause because users must have at least one auth provider. For more information, see Creating users.

Required privilege	`GRANT CREATE USER`
	
	For more information, see Grant privilege to create users.
	
	`GRANT DROP USER`
	
	For more information, see Grant privilege to delete users.

Command	`RENAME USER`
Syntax	`RENAME USER name [IF EXISTS] TO otherName`

Description	Changes the name of a user. For more information, see Renaming users.
Required privilege	`GRANT RENAME USER` For more information, see DBMS USER MANAGEMENT privileges.

Command	`ALTER USER`

Syntax	`ALTER USER name [IF EXISTS]`
	`  [REMOVE HOME DATABASE]`
	`  [REMOVE { AUTH [PROVIDER[S]] provider[, ...] | ALL AUTH [PROVIDER[S]] }]...`
	`  [SET [PLAINTEXT | ENCRYPTED] PASSWORD 'password']`
	`  [[SET PASSWORD] CHANGE [NOT] REQUIRED]`
	`  [SET STATUS {ACTIVE | SUSPENDED} ]`
	`  [SET HOME DATABASE name]`
	`  [SET AUTH [PROVIDER] 'provider' "{"{SET <key> <value>}..."}"]... # Introduced in Neo4j 5.24`
	
	`<key><value>` pairs for the `SET AUTH` clause could include:
	
	`SET AUTH [PROVIDER] 'provider' "{"`
	`  {`
	`  SET ID 'id' # a unique identifier of the user in an external system`
	`  | SET [PLAINTEXT | ENCRYPTED] PASSWORD 'password' # PASSWORD clauses are only applicable to the 'native' provider`
	`  | SET PASSWORD CHANGE [NOT] REQUIRED # PASSWORD clauses are only applicable to the 'native' provider`
	`  }`
	`"}"`

Description	Modifies the settings for an existing user. • At least one `SET` or `REMOVE` clause is required. • Any `REMOVE` clause(s) must appear before the first `SET` clause. For more information, see Modifying users.

Required privilege	`GRANT SET PASSWORD`
	
	`GRANT SET USER STATUS`
	
	`GRANT SET USER HOME DATABASE`
	
	`GRANT SET AUTH`
	
	For more information, see DBMS USER MANAGEMENT privileges.

Command	`ALTER CURRENT USER SET PASSWORD`
Syntax	`ALTER CURRENT USER SET PASSWORD FROM 'oldPassword' TO 'newPassword'`
Description	Changes the current user's password. For more information, see Changing the current user's password.

Required privilege	None

Command	`DROP USER`
Syntax	`DROP USER name [IF EXISTS]`
Description	Removes an existing user. For more information, see Delete users.
Required privilege	`GRANT DROP USER` For more information, see Grant privilege to delete users.

Note:

The `SHOW USER[S] PRIVILEGES` command is described in Listing privileges.

Listing current user

You can view the currently logged-in user using the Cypher command `SHOW CURRENT USER`. It produces a table with the following columns:

Column	Description	Type	Community Edition	Enterprise Edition
user	User name	`STRING`	✔	✔
roles	Roles granted to the user. It returns `null` in Community edition.	`LIST OF STRING`	✖	✔
passwordChangeRequired	If `true`, the user must change their password at the next login. This is `null` if the user has `native` auth disabled.	`BOOLEAN`	✔	✔
suspended	If `true`, the user is currently suspended. It returns `null` in Community edition.	`BOOLEAN`	✖	✔
home	The home database configured by the user, or `null` if no home database has been configured. If this database is unavailable and the user does not specify a database to use, they will not be able to log in. It returns `null` in Community edition.	`STRING`	✖	✔

```
SHOW CURRENT USER
```

Table 420. Result

user	roles	passwordChangeRequired	suspended	home
"jake"	["PUBLIC"]	false	false	<null>
Rows: 1				

Note:

This command is only supported for a logged-in user and returns an empty result if authorization has been disabled.

Listing users

You can list all available users using the Cypher command `SHOW USERS`. It produces a table containing a single row per user with the following columns:

Column	Description	Type	Community Edition	Enterprise Edition
user	User name	STRING	✔	✔
roles	Native roles granted to the user using the `GRANT ROLE` command. The set of roles a user receives in practice may differ from those in this column. It depends on DMBS configuration and the user's auth providers. For example, if they use external (e.g. LDAP or OIDC) auth, or if `native` is not listed in the `dbms.security.authorization_providers` configuration setting. It returns `null` in Community edition.	LIST OF STRING	✖	✔
passwordChangeRequired	If `true`, the user must change their password at the next login. This is `null` if the user has `native` auth disabled.	BOOLEAN	✔	✔
suspended	If `true`, the user is currently suspended. It returns `null` in Community edition.	BOOLEAN	✖	✔

Column	Description	Type	Community Edition	Enterprise Edition
home	The home database configured for the user, otherwise `null`. A home database is resolved if it is pointing to a database or a database alias. If the configured home database is unavailable and the user does not specify another database, the login will fail. It returns `null` in Community edition.	`STRING`	✖	✔

Note:

When first starting a Neo4j DBMS, there is always a single default user `neo4j` with administrative privileges. It is possible to set the initial password using `neo4j-admin dbms set-initial-password <password>`, otherwise you must change the password after the first login.

Example 83. Show users

```
SHOW USERS
```

Table 421. Result

user	roles	passwordChangeRequired	suspended	home
"neo4j"	["admin","PUBLIC"]	false	false	<null>
"jake"	["PUBLIC"]	false	false	<null>
Rows: 2				

Example 84. Show user with column reorder and filtering

This example shows how to:

- Reorder the columns using a `YIELD` clause.
- Filter the results using a `WHERE` clause.

```
SHOW USER YIELD user, suspended, passwordChangeRequired, roles, home
WHERE user = 'jake'
```

Table 422. Result

user	suspended	passwordChangeRequired	roles	home
"jake"	false	false	["PUBLIC"]	<null>

user	suspended	passwordChangeRequired	roles	home
Rows: 1				

Example 85. Show user with `RETURN` clause

It is possible to add a `RETURN` clause to further manipulate the results after filtering. In this example, the `RETURN` clause is used to filter out the `roles` column and rename the `user` column to `adminUser`.

```
SHOW USERS YIELD roles, user
WHERE 'admin' IN roles
RETURN user AS adminUser
```

Table 423. Result

adminUser
"neo4j"
Rows: 1

Listing user auth providers

Introduced in 5.24

To inspect available user auth providers, use `SHOW USERS WITH AUTH`. The command produces a row per user per auth provider and yields the following two columns in addition to those output by `SHOW USERS`:

Column	Description	Type	Community Edition	Enterprise Edition
provider	The name of the auth provider.	`STRING`	✔	✔
auth	A map containing configuration for the user. For example, dn of the user for an `ldap` auth provider, the unique external identifier for an `oidc` auth provider, or password status for a `native` auth provider.	`MAP`	✔	✔

Example 86. Show users with auth

```
SHOW USERS WITH AUTH
```

Table 424. Result

user	roles	passwordChangeRequired	suspended	home	provider	auth
"neo4j"	["admin","PUBLIC"]	false	false	<null>	"native"	{ "password": "***", "changeRequired": false }
"jack"	["PUBLIC"]	false	false	<null>	"native"	{ "password": "***", "changeRequired": false }
"jack"	["PUBLIC"]	false	false	<null>	"oidc1"	{ "id": "jacksIdForOidc1" }
Rows: 3						

Example 87. Show user with auth using filtering

Show all users with the `oidc` auth provider.

```
SHOW USERS WITH AUTH
WHERE provider = 'oidc1'
```

Table 425. Result

user	roles	passwordChangeRequired	suspended	home	provider	auth
"jack"	["PUBLIC"]	false	false	<null>	"oidc1"	{ "id": "jacksIdForOidc1" }
Rows: 1						

For more information about auth providers, see User auth providers.

Creating users

You can create users using one of the following Cypher commands, depending on whether you want to create a new user or replace an existing one. In both cases, you can specify the user's password, whether they must change it at the next login, their status, home database, and auth provider settings. The `SET` clauses can be applied in any order. It is mandatory to specify a `SET PASSWORD` and/or at least one `SET AUTH` clause because users must have at least one auth provider. `SET AUTH` is available from Neo4j 5.24 onwards.

`CREATE USER` syntax

```
CREATE USER name [IF NOT EXISTS]
  [SET [PLAINTEXT \| ENCRYPTED] PASSWORD 'password']
  [[SET PASSWORD] CHANGE [NOT] REQUIRED]
  [SET STATUS {ACTIVE \| SUSPENDED}]
  [SET HOME DATABASE name]
  [SET AUTH [PROVIDER] 'provider' "{"{SET <key> <value>}..."}"]...
```

`CREATE OR REPLACE USER` syntax

```
CREATE OR REPLACE USER name
  [SET [PLAINTEXT \| ENCRYPTED] PASSWORD 'password']
  [[SET PASSWORD] CHANGE [NOT] REQUIRED]
  [SET STATUS {ACTIVE \| SUSPENDED}]
  [SET HOME DATABASE name]
  [SET AUTH [PROVIDER] 'provider' "{"{SET <key> <value>}..."}"]...
```

Where:

Specifies the command to create a user.

Specifies the password for the user. The `'password'` can either be a string value or a string parameter with default value length of at least 8 characters.
The `PLAINTEXT` and `ENCRYPTED` keywords are optional and can be used to specify the format of the password, i.e. whether Neo4j needs to hash it or it has already been hashed. By default, all passwords are encrypted (hashed) when stored in the Neo4j `system` database.

- The optional `PLAINTEXT` in `SET PLAINTEXT PASSWORD` has the same behavior as `SET PASSWORD`.
- The optional `ENCRYPTED` is used to recreate an existing user when the plaintext password is unknown, but the encrypted password is available in the */data/scripts/databasename/restore_metadata.cypher* file of restored database backup. See Restore users and roles metadata.
  With `ENCRYPTED`, the password string is expected to be in the format of `<encryption-version>,<hash>,<salt>`, where, for example:
  - `0` is the first version and refers to the `SHA-256` cryptographic hash function with iterations `1`.
  - `1` is the second version and refers to the `SHA-256` cryptographic hash function with iterations `1024`.

Specifies whether the user must change their password at the next login. If the optional `SET PASSWORD CHANGE [NOT] REQUIRED` is omitted but a password is given, the default is `CHANGE REQUIRED`. The `SET PASSWORD` prefix of the `CHANGE [NOT] REQUIRED` clause is only optional if it directly follows the `SET PASSWORD 'password'` clause and is not part of a `SET AUTH` clause.

Specifies the user's status. If not set, the default is `ACTIVE`.

Specifies a home database for a user. A home database is resolved if it is pointing to a database or a database alias. If no home database is set, the DBMS default database is used as the home database for that user.

Introduced in 5.24 One or more `SET AUTH` clause can be used to configure external auth providers, such as LDAP or OIDC, which define authentication/authorization providers for that user. `SET AUTH` can also be used as an alternative way to set the native (password-based) auth settings like `SET PASSWORD` and `SET PASSWORD CHANGE REQUIRED`. For further informations, see the examples in this section, as well as Configure SSO at the user level using auth providers for OIDC, and Configure authentication/authorization at the user level using auth providers for LDAP.

```
SET AUTH [PROVIDER] 'provider' "{"
    {
        SET ID 'id' # a unique identifier of the user in an external system.
        \| SET [PLAINTEXT \| ENCRYPTED] PASSWORD 'password' # only applicable to the 'native' provider.
        \| SET PASSWORD CHANGE [NOT] REQUIRED # only applicable to the 'native' provider.
    }
"}"
```

Note:

Usernames are case sensitive. The created user will appear on the list provided by `SHOW USERS`.

- In Neo4j Community Edition there are no roles, but all users have implied administrator privileges.
- In Neo4j Enterprise Edition all users are automatically assigned the `PUBLIC` role, giving them a base set of privileges.

Example 88. Create user

For example, you can create the user `jake` in a suspended state, with the home database `anotherDb`, and the requirement to change the password by using the command:

```
CREATE USER jake
SET PASSWORD 'abcd1234' CHANGE REQUIRED
SET STATUS SUSPENDED
SET HOME DATABASE anotherDb
```

Introduced in 5.24

The equivalent command using the auth providers syntax would be:

```
CREATE USER jake
SET STATUS SUSPENDED
SET HOME DATABASE anotherDb
SET AUTH 'native' {SET PASSWORD 'abcd1234' SET PASSWORD CHANGE REQUIRED}
```

Example 89. Create user with an encrypted password

Or you can create the user `Jake` in an active state, with an encrypted password (taken from the */data/scripts/databasename/restore_metadata.cypher* of a restored database backup), and the requirement to not change the password by running:

```
CREATE USER Jake
SET ENCRYPTED PASSWORD
'1,6d57a5e0b3317055454e455f96c98c750c77fb371f3f0634a1b8ff2a55c5b825,190ae47c661e0668a0c8be8a21ff78a4a
34cdf918cae3c407e907b73932bd16c' CHANGE NOT REQUIRED
SET STATUS ACTIVE
```

Introduced in 5.24

The equivalent command using the auth providers syntax would be: .

```
CREATE USER jake
SET STATUS ACTIVE
SET AUTH 'native' {
  SET ENCRYPTED PASSWORD
'1,6d57a5e0b3317055454e455f96c98c750c77fb371f3f0634a1b8ff2a55c5b825,190ae47c661e0668a0c8be8a21ff78a4a
34cdf918cae3c407e907b73932bd16c'
```

```
  SET PASSWORD CHANGE NOT REQUIRED
}
```

Note:

The `SET STATUS {ACTIVE | SUSPENDED}`, `SET HOME DATABASE` parts of the commands are only available in Neo4j Enterprise Edition. The `SET AUTH` clause for external providers is only available in Neo4j Enterprise Edition. However, `SET AUTH 'native'` can be used in Neo4j Community Edition.

The `CREATE USER` command is optionally idempotent, with the default behavior to throw an exception if the user already exists. Appending `IF NOT EXISTS` to the `CREATE USER` command will ensure that no exception is thrown and nothing happens should the user already exist.

Example 90. Create user if not exists

```
CREATE USER jake IF NOT EXISTS
SET PLAINTEXT PASSWORD 'abcd1234'
```

Introduced in 5.24

The equivalent command using the auth providers syntax would be:

```
CREATE USER jake IF NOT EXISTS
SET AUTH 'native' {SET PLAINTEXT PASSWORD 'abcd1234'}
```

The `CREATE OR REPLACE USER` command will result in any existing user being deleted and a new one created.

Example 91. Create or replace user

```
CREATE OR REPLACE USER jake
SET PLAINTEXT PASSWORD 'abcd1234'
```

This is equivalent to running `DROP USER jake IF EXISTS` followed by `CREATE USER jake SET PASSWORD 'abcd1234'`.

Introduced in 5.24

The equivalent command using the auth providers syntax would be:

```
CREATE OR REPLACE USER jake
SET AUTH 'native' {SET PLAINTEXT PASSWORD 'abcd1234'}
```

Note:

The `CREATE OR REPLACE USER` command does not allow the use of `IF NOT EXISTS`.

Renaming users

Users can be renamed with the `RENAME USER` command.

```
RENAME USER jake TO bob
```

To verify the change, you can use the `SHOW USERS` command:

```
SHOW USERS
```

Table 426. Result

user	roles	passwordChangeRequired	suspended	home
"bob"	["PUBLIC"]	true	false	<null>
"neo4j"	["admin","PUBLIC"]	true	false	<null>
Rows: 2				

Note:

The `RENAME USER` command is only available when using native authentication and authorization.

Modifying users

You can modify users with the `ALTER USER` command. The command allows you to change the user's password, status, home database, and auth provider settings. The `SET` and `REMOVE` clauses can be applied in any order. However, all `REMOVE` clauses must come before the first `SET` clause and at least one `SET` or `REMOVE` clause is required for the command. If any of the `SET` or `REMOVE` clauses are omitted, the corresponding settings will not be changed.

```
ALTER USER name [IF EXISTS]
  [REMOVE HOME DATABASE]
  [REMOVE { AUTH [PROVIDER[S]] provider[, ...] \| ALL AUTH [PROVIDER[S]] }]...
  [SET [PLAINTEXT | ENCRYPTED] PASSWORD 'password']
  [[SET PASSWORD] CHANGE [NOT] REQUIRED]
  [SET STATUS {ACTIVE | SUSPENDED}]
  [SET HOME DATABASE name]
  [SET AUTH [PROVIDER] 'provider' "{"{SET <key> <value>}..."}"]...
```

Where:

Specifies the command to alter a user.

Removes the home database for the user. As a result, the DBMS default database will be used as the home database for that user.

Introduced in 5.24 Removes one, several, or all existing auth provider(s) from a user. However, a user must always have at least one auth provider. Therefore, `REMOVE ALL AUTH` must be used in conjunction with at least one `SET AUTH` clause in order to meet this requirement.

Specifies the password for the user. The `'password'` can either be a string value or a string parameter with default value length of at least 8 characters.
The `PLAINTEXT` and `ENCRYPTED` keywords are optional and can be used to specify the format of the password, i.e. whether Neo4j needs to hash it or it has already been hashed. By default, all passwords are encrypted (hashed) when stored in the Neo4j `system` database.

- The optional `PLAINTEXT` in `SET PLAINTEXT PASSWORD` has the same behavior as `SET PASSWORD`.
- The optional `ENCRYPTED` is used to recreate an existing user when the plaintext password is unknown, but the encrypted password is available in the */data/scripts/databasename/restore_metadata.cypher* file when you restore a database backup. See Restore users and roles metadata.
  With `ENCRYPTED`, the password string is expected to be in the format of `<encryption-version>,<hash>,<salt>`, where, for example:
  - `0` is the first version and refers to the `SHA-256` cryptographic hash function with iterations `1`.
  - `1` is the second version and refers to the `SHA-256` cryptographic hash function with iterations `1024`.

Specifies whether the user must change their password at the next login. If the optional `SET PASSWORD CHANGE [NOT] REQUIRED` is omitted when adding native auth to a user (either by first removing pre-existing native auth or if the user does not have native auth to start with), the default is `CHANGE REQUIRED`. The `SET PASSWORD` prefix of the `CHANGE [NOT] REQUIRED` clause is only optional if it directly follows the `SET PASSWORD 'password'` clause and is not part of a `SET AUTH` clause.

Specifies the user's status.

Specifies a home database for a user. A home database is resolved if it is pointing to a database or a database alias. If no home database is set, the DBMS default database is used as the home database for that user.

Introduced in 5.24 One or more `SET AUTH` clauses can be used to set auth providers, which define authentication / authorization providers for that user. This might be used to configure external auth providers like LDAP or OIDC, but can also be used as an alternative way to set the native (password-based) auth settings like `SET PASSWORD` and `SET PASSWORD CHANGE REQUIRED`. For further informations, see the examples in this section, as well as Configure SSO at the user level using auth providers, and Configure authentication/authorization at the user level using auth providers.

```
SET AUTH [PROVIDER] 'provider' "{"
    {
        SET ID 'id' # a unique identifier of the user in an external system
        \| SET [PLAINTEXT \| ENCRYPTED] PASSWORD 'password' # only applicable to the 'native' provider
        \| SET PASSWORD CHANGE [NOT] REQUIRED # only applicable to the 'native' provider
    }
"}"
```

Example 92. Modify a user's password and status

For example, you can modify the user `bob` by setting a new password and active status, and removing the requirement to change his password by running:

```
ALTER USER bob
SET PASSWORD 'abcd5678' CHANGE NOT REQUIRED
SET STATUS ACTIVE
```

Introduced in 5.24

The equivalent command using the auth providers syntax would be:

```
ALTER USER bob
SET AUTH 'native' {SET PASSWORD 'abcd5678' SET PASSWORD CHANGE NOT REQUIRED}
SET STATUS ACTIVE
```

Example 93. Modify a user to expire their current password

For example, you can modify the user `bob` to expire his current password so that he must change it the next time he logs in:

```
ALTER USER bob
SET PASSWORD CHANGE REQUIRED
```

Introduced in 5.24

The equivalent command using the auth providers syntax would be:

```
ALTER USER bob
SET AUTH 'native' {SET PASSWORD CHANGE REQUIRED}
```

Example 94. Modify a user to use an external OIDC auth provider

For example, you can modify the user `bob` by removing his native auth provider and adding an external OIDC auth provider:

```
ALTER USER bob
REMOVE AUTH 'native'
SET AUTH 'oidc-mysso1' {SET ID 'bobsUniqueMySso1Id'}
```

Example 95. Modify a user to use multiple external OIDC auth providers

For example, you can modify the user `bob` by removing all of his existing auth providers and adding two external OIDC auth providers:

```
ALTER USER bob
REMOVE ALL AUTH
SET AUTH 'oidc-mysso1' {SET ID 'bobsUniqueMySso1Id'}
SET AUTH 'oidc-mysso2' {SET ID 'bobsUniqueMySso2Id'}
```

Example 96. Assign a user a different home database

For example, you can modify the user `bob` by assigning him a different home database:

```
ALTER USER bob
SET HOME DATABASE anotherDbOrAlias
```

Example 97. Remove the home database from a user and set their status to suspended

For example, you can modify the user `bob` by removing his home database and setting his status to suspended:

```
ALTER USER bob
REMOVE HOME DATABASE
SET STATUS SUSPENDED
```

Note:

When altering a user, it is only necessary to specify the changes required. For example, leaving out the `CHANGE [NOT] REQUIRED` part of the query leaves that unchanged.

Note:

The `SET STATUS {ACTIVE | SUSPENDED}`, `SET HOME DATABASE`, `REMOVE HOME DATABASE`, and `REMOVE AUTH` parts of the command are only available in Neo4j Enterprise Edition. The `SET AUTH` clause for external providers is only available in Neo4j Enterprise Edition. However, `SET AUTH 'native'` can be used in Neo4j Community Edition.

The changes to the user will appear on the list provided by `SHOW USERS`:

```
SHOW USERS
```

Table 427. Result

user	roles	passwordChangeRequired	suspended	home
"bob"	["PUBLIC"]	false	false	<null>
"neo4j"	["admin","PUBLIC"]	true	false	<null>
Rows: 2				

The default behavior of this command is to throw an exception if the user does not exist. Adding an optional parameter `IF EXISTS` to the command makes it idempotent and ensures that no exception is thrown. Nothing happens should the user not exist.

```
ALTER USER nonExistingUser IF EXISTS SET PASSWORD 'abcd1234'
```

Changing the current user's password

Users can change their password using `ALTER CURRENT USER SET PASSWORD`. The old password is required in addition to the new one, and either or both can be a string value or a string parameter. When a user executes this command it will change their password as well as set the `CHANGE NOT REQUIRED` flag.

```
ALTER CURRENT USER
SET PASSWORD FROM 'password1' TO 'password2'
```

Note:

This command works only for a logged-in user and cannot be run with auth disabled.

Delete users

Users can be deleted with `DROP USER`.

```
DROP USER bob
```

Deleting a user does not automatically terminate associated connections, sessions, transactions, or queries.

However, when a user is deleted, it no longer appears on the list provided by `SHOW USERS`:

```
SHOW USERS
```

Table 428. Result

user	roles	passwordChangeRequired	suspended	home
"neo4j"	["admin","PUBLIC"]	true	false	<null>
Rows: 1				

Manage roles

Roles can be created and managed using a set of Cypher administration commands executed against the `system` database.

When connected to the DBMS over `bolt`, administration commands are automatically routed to the `system` database.

Role management command syntax

Note:

For more details about the syntax descriptions, see Database management command

syntax.

Command	`SHOW ROLES`

Syntax	`SHOW [ALL|POPULATED] ROLE[S]`
	`  [YIELD { * | field[, ...] } [ORDER BY field[, ...]] [SKIP n] [LIMIT n]]`
	`  [WHERE expression]`
	`  [RETURN field[, ...] [ORDER BY field[, ...]] [SKIP n] [LIMIT n]]`

Description	Lists roles. When using the `RETURN` clause, the `YIELD` clause is mandatory and must not be omitted. For more information, see Listing roles.
Required privilege	`GRANT SHOW ROLE` See DBMS ROLE MANAGEMENT privileges.

Command	`SHOW ROLES WITH USERS`

Syntax	`SHOW [ALL|POPULATED] ROLE[S] WITH USER[S]`
	`  [YIELD { * | field[, ...] } [ORDER BY field[, ...]] [SKIP n] [LIMIT n]]`
	`  [WHERE expression]`
	`  [RETURN field[, ...] [ORDER BY field[, ...]] [SKIP n] [LIMIT n]]`

Description	Lists roles and users assigned to them. When using the `RETURN` clause, the `YIELD` clause is mandatory and must not be omitted. For more information, see Listing roles.

Required privilege	`GRANT SHOW ROLE`
	
	For more information, see DBMS ROLE MANAGEMENT privileges.
	
	`GRANT SHOW USER`
	
	For more information, see DBMS USER MANAGEMENT privileges.

Command	`SHOW ROLE PRIVILEGES`

Syntax	`SHOW ROLE[S] name[, ...] PRIVILEGE[S] [AS [REVOKE] COMMAND[S]]`
	`  [YIELD { * | field[, ...] } [ORDER BY field[, ...]] [SKIP n] [LIMIT n]]`
	`  [WHERE expression]`
	`  [RETURN field[, ...] [ORDER BY field[, ...]] [SKIP n] [LIMIT n]]`

Description	Lists the privileges granted to the specified roles. When using the `RETURN` clause, the `YIELD` clause is mandatory and must not be omitted. The `SHOW ROLE name PRIVILEGES` command is described in Listing privileges.

Required privilege	`GRANT SHOW PRIVILEGE` See DBMS PRIVILEGE MANAGEMENT privileges.

Command	`CREATE [IMMUTABLE] ROLE`
Syntax	`CREATE ROLE name [IF NOT EXISTS] [AS COPY OF otherName]`
Description	Creates a new role. For more information, see Creating roles.
Required privilege	`GRANT CREATE ROLE` For more information, see DBMS ROLE MANAGEMENT privileges.

Command	`CREATE OR REPLACE ROLE`
Syntax	`CREATE OR REPLACE ROLE name [AS COPY OF otherName]`
Description	Creates a new role, or if a role with the same name exists, replace it. For more information, see Creating roles.

Required privilege	`GRANT CREATE ROLE`
	
	`GRANT DROP ROLE`
	
	For more information, see DBMS ROLE MANAGEMENT privileges.

Command	`RENAME ROLE`
Syntax	`RENAME ROLE name [IF EXISTS] TO otherName`
Description	Changes the name of a role. For more information, see Renaming roles.
Required privilege	`GRANT RENAME ROLE` For more information, see DBMS ROLE MANAGEMENT privileges.

Command	`DROP ROLE`
Syntax	`DROP ROLE name [IF EXISTS]`

Description	Removes a role. For more information, see Deleting roles.
Required privilege	`GRANT DROP ROLE` For more information, see DBMS ROLE MANAGEMENT privileges.

Command	`GRANT ROLE TO`
Syntax	`GRANT ROLE[S] name[, ...] TO user[, ...]`
Description	Assigns roles to users. For more information, see Assigning roles to users.
Required privilege	`GRANT ASSIGN ROLE` For more information, see DBMS ROLE MANAGEMENT privileges.

Command	`REVOKE ROLE`
Syntax	`REVOKE ROLE[S] name[, ...] FROM user[, ...]`
Description	Removes roles from users. For more information, see Revoking roles from users.
Required privilege	`GRANT REMOVE ROLE` For more information, see DBMS ROLE MANAGEMENT privileges.

Listing roles

You can view all available roles using the Cypher command `SHOW ROLES`, which returns a single column by default. Starting from 5.26, you can optionally use `SHOW ROLES YIELD *` to see if the role is immutable. See Immutable roles for more information.

Table 429. `SHOW ROLES` output

Column	Description	Type
role	Role name	`STRING`
immutable	`true` if the role is immutable, otherwise `false`.	`BOOLEAN`

Example 98. List all roles

```
SHOW ROLES
```

This is the same command as `SHOW ALL ROLES`.

Table 430. Result

role
"PUBLIC"
"admin"
"architect"
"editor"
"publisher"
"reader"
Rows: 6

When first starting a Neo4j DBMS, there are a number of built-in roles:

- `PUBLIC` - a role that all users have granted. By default it gives access to the home database and to execute privileges for procedures and functions.
- `reader` - can perform traverse and read operations in all databases except `system`.
- `editor` - can perform traverse, read, and write operations in all databases except `system`, but cannot create new labels or relationship types.
- `publisher` - can do the same as `editor`, but also create new labels and relationship types.
- `architect` - can do the same as `publisher` as well as create and manage indexes and constraints.
- `admin` - can do the same as all the above, as well as manage databases, aliases, users, roles, and privileges.

More information about the built-in roles and their privileges can be found in Built-in roles and privileges.

There are multiple versions of this command, the default being `SHOW ALL ROLES`. To only show roles that are assigned to users, the command is `SHOW POPULATED ROLES`. To see which users are assigned to which roles, `WITH USERS` can be added to the command. The command produces a row per role per user and yields the following column in addition to the one output by `SHOW ROLES`:

Table 431. `SHOW POPULATED ROLES WITH USERS` output

Column	Description	Type
member	User name	`STRING`

Since this gives a result with one row for each user, it shows up twice if a role is assigned to two users.

Example 99. Show roles with users

```
SHOW POPULATED ROLES WITH USERS
```

The table of results will show information about the role and what database it belongs to:

Table 432. Result

role	member
"PUBLIC"	"neo4j"
"PUBLIC"	"bob"
"PUBLIC"	"user1"
"PUBLIC"	"user2"
"PUBLIC"	"user3"
"admin"	"neo4j"
Rows: 6	

It is also possible to filter and sort the results by using `YIELD`, `ORDER BY` and `WHERE`.

Example 100. Show roles with ordering and filtering

```
SHOW ROLES YIELD role
ORDER BY role
WHERE role ENDS WITH 'r'
```

In this example:

- The results have been filtered to only return the roles ending in 'r'.
- The results are ordered by the `action` column using `ORDER BY`.

It is also possible to use `SKIP` and `LIMIT` to paginate the results.

Table 433. Result

role
"editor"
"publisher"
"reader"
Rows: 3

Creating roles

Roles can be created using `CREATE [IMMUTABLE] ROLE`:

```
CREATE [IMMUTABLE] ROLE name [IF NOT EXISTS] [AS COPY OF otherName]
```

Roles can be created or replaced by using `CREATE OR REPLACE [IMMUTABLE] ROLE`:

```
CREATE OR REPLACE [IMMUTABLE] ROLE name [AS COPY OF otherName]
```

Note:

The following naming rules apply:

- The first character must be an ASCII alphabetic character.
- Subsequent characters can be ASCII alphabetic, numeric characters, and underscore.
- Role names are case sensitive.

A role can be copied, keeping its privileges, using `CREATE [IMMUTABLE] ROLE name AS COPY OF otherName`.

Example 101. Copy a role

```
CREATE ROLE mysecondrole AS COPY OF myrole
```

Created roles will appear on the list provided by `SHOW ROLES`.

Example 102. List roles

```
SHOW ROLES
```

Table 434. Result

role
"PUBLIC"
"admin"
"architect"
"editor"
"myrole"
"mysecondrole"
"publisher"
"reader"
Rows: 8

The `CREATE ROLE` command is optionally idempotent, with the default behavior to throw an exception if the role already exists. Adding `IF NOT EXISTS` to the `CREATE ROLE` command will ensure that no exception is thrown and nothing happens should the role already exist.

Example 103. Create role if not exists

```
CREATE ROLE myrole IF NOT EXISTS
```

The `CREATE OR REPLACE ROLE` command will result in any existing role being deleted and a new one created.

Example 104. Create or replace role

```
CREATE OR REPLACE ROLE myrole
```

This is equivalent to running `DROP ROLE myrole IF EXISTS` followed by `CREATE ROLE myrole`.

Note:

The `CREATE OR REPLACE ROLE` command does not allow you to use the `IF NOT EXISTS`.

Immutable roles

Immutable roles are those that cannot be modified in the usual way. This means they cannot be created, renamed, dropped, or have privileges granted to or revoked from them under normal operating conditions. See Immutable roles and privileges for details of when and how the `IMMUTABLE` keyword may be used.

They are useful in cases where you need a permanent built-in system role that cannot be modified even by users who have `ROLE MANAGEMENT` privileges but yet can be granted to and revoked from users in the same way as an ordinary role.

Renaming roles

Roles can be renamed using `RENAME ROLE` command:

```
RENAME ROLE mysecondrole TO mythirdrole
```

```
SHOW ROLES
```

Table 435. Result

role
"PUBLIC"
"admin"
"architect"
"editor"

role
"myrole"
"mythirdrole"
"publisher"
"reader"
Rows: 8

Note:

The `RENAME ROLE` command is only available when using native authentication and authorization.

Assigning roles to users

Users can be given access rights by assigning them roles using `GRANT ROLE`:

```
GRANT ROLE myrole TO bob
```

The roles assigned to each user can be seen on the list provided by `SHOW USERS`:

```
SHOW USERS
```

Table 436. Result

user	roles	passwordChangeRequired	suspended	home
"bob"	["myrole","PUBLIC"]	false	false	<null>
"neo4j"	["admin","PUBLIC"]	true	false	<null>
"user1"	["PUBLIC"]	true	false	<null>
"user2"	["PUBLIC"]	true	false	<null>
"user3"	["PUBLIC"]	true	false	<null>
Rows: 5				

It is possible to assign multiple roles to multiple users in one command:

```
GRANT ROLES role1, role2 TO user1, user2, user3
```

```
SHOW USERS
```

Table 437. Result

user	roles	passwordChangeRequired	suspended	home
"bob"	["myrole","PUBLIC"]	false	false	<null>
"neo4j"	["admin","PUBLIC"]	true	false	<null>
"user1"	["role1","role2","PUBLIC"]	true	false	<null>
"user2"	["role1","role2","PUBLIC"]	true	false	<null>
"user3"	["role1","role2","PUBLIC"]	true	false	<null>
Rows: 5				

Common errors, such as attempts to grant roles to users who have already been granted those roles, will lead to notifications. Some of these notifications may be replaced with errors in a future major version of Neo4j. See Status Codes for Errors & Notifications → List of notification codes for details on notifications.

Revoking roles from users

Users can lose access rights by revoking their role using `REVOKE ROLE`:

```
REVOKE ROLE myrole FROM bob
```

The roles revoked from users can no longer be seen on the list provided by `SHOW USERS`:

```
SHOW USERS
```

Table 438. Result

user	roles	passwordChangeRequired	suspended	home
"bob"	["PUBLIC"]	false	false	<null>
"neo4j"	["admin","PUBLIC"]	true	false	<null>
"user1"	["role1","role2","PUBLIC"]	true	false	<null>
"user2"	["role1","role2","PUBLIC"]	true	false	<null>
"user3"	["role1","role2","PUBLIC"]	true	false	<null>
Rows: 5				

It is possible to revoke multiple roles from multiple users in one command:

```
REVOKE ROLES role1, role2 FROM user1, user2, user3
```

Common errors, such as misspellings or attempts to revoke roles from users who have not been granted those roles, will lead to notifications. Some of these notifications may be replaced with errors in a future major version of Neo4j. See Status Codes → Notification codes for details on notifications.

Deleting roles

Roles can be deleted using `DROP ROLE` command:

```
DROP ROLE mythirdrole
```

When a role has been deleted, it will no longer appear on the list provided by `SHOW ROLES`:

```
SHOW ROLES
```

Table 439. Result

role
"PUBLIC"
"admin"
"architect"
"editor"
"myrole"
"publisher"
"reader"
Rows: 8

This command is optionally idempotent, with the default behavior to throw an exception if the role does not exist. Adding `IF EXISTS` to the command will ensure that no exception is thrown and nothing happens should the role not exist:

```
DROP ROLE mythirdrole IF EXISTS
```

Recover admin user and password

This page describes how to reset a password to recover a user's access when their password is lost. It specifically focuses on how to recover an admin user if all the admin users have been unassigned the admin role, and how to recreate the built-in admin role if it has been dropped.

Disable authentication

1. Stop Neo4j:

   ```
   bin/neo4j stop
   ```

2. Open the *neo4j.conf* file and set `dbms.security.auth_enabled` parameter to `false` to disable the authentication:

```
dbms.security.auth_enabled=false
```

3. Block network connections during the recovery phase, so users can connect to Neo4j only via `localhost`. This can be achieved by temporarily commenting out the `server.default_listen_address` parameter and providing the specific localhost value:

```
server.default_listen_address=127.0.0.1
```

> **Note:**
>
> Ensure, you have blocked all network connections for any individual services configured to listen to `listen_addresses`.

4. Start Neo4j:

```
bin/neo4j start
```

1. Stop all members of the cluster:

```
bin/neo4j stop
```

2. On each member, open the *neo4j.conf* file and modify the following settings:
   a. Set `dbms.security.auth_enabled` parameter to `false` to disable the authentication:

```
dbms.security.auth_enabled=false
```

   b. Disable the HTTP and HTTPS network connections and restrict the `bolt` connector to use only `localhost`. Set `server.http.enabled` to `false`. This ensures that no one from outside can access the cluster during the recovery period.

```
server.http.enabled=false
#server.https.enabled=true
server.bolt.listen_address:127.0.0.1
```

3. Start all members of the cluster:

```
bin/neo4j start
```

Recover a lost password

You can use a client such as Cypher Shell or the Neo4j Browser to connect to the `system` database and set a new password for the admin user.

> **Note:**
>
> In a cluster deployment, you should complete the steps only on one of the cluster members.

1. Complete the steps in Disable authentication as per your deployment.
2. Connect to the `system` database using Cypher shell. Alternatively, log into Neo4j Browser.

```
bin/cypher-shell -d system
```

> **Note:**
>
> If you have specified a non-default port for your `bolt` connector, add `-a neo4j://<your-cluster-member>:<non-default-bolt-port>` to the `cypher-shell` command to be able to connect to your cluster member.

3. Set a new password for the admin user. In this example, the admin user is named `neo4j`.

```
ALTER USER neo4j SET PASSWORD 'mynewpassword'
```

4. Exit the `cypher-shell` console:

```
:exit;
```

5. Proceed with the post-recovery steps as per your deployment.

Recover an unassigned admin role

You can use a client such as Cypher Shell or the Neo4j Browser to connect to the `system` database and grant the admin user role to an existing user.

> **Note:**
>
> In a cluster deployment, you should complete the steps only on one of the cluster members.

1. Complete the steps in Disable authentication as per your deployment.
2. Connect to the `system` database using Cypher shell. Alternatively, log into Neo4j Browser.

```
bin/cypher-shell -d system
```

> **Note:**
>
> If you have specified a non-default port for your `bolt` connector, add `-a neo4j://<your-cluster-member>:<non-default-bolt-port>` to the `cypher-shell` command to be able to connect to your cluster member.

3. Grant the admin user role to an existing user. In this example, the user is named `neo4j`.

```
GRANT ROLE admin TO neo4j
```

4. Exit the `cypher-shell` console:

```
:exit;
```

5. Proceed with the post-recovery steps as per your deployment.

Recover the admin role

If you have removed the admin role from your system entirely, you can use a client such as Cypher Shell or the Neo4j Browser to connect to the `system` database and recreate the role with its original capabilities.

> **Note:**
>
> In a cluster deployment, you should complete the steps only on one of the cluster members.

1. Complete the steps in Disable authentication as per your deployment.
2. Connect to the `system` database using Cypher shell. Alternatively, log into Neo4j Browser.

```
bin/cypher-shell -d system
```

> **Note:**
>
> If you have specified a non-default port for your `bolt` connector, add `-a neo4j://<your-cluster-member>:<non-default-bolt-port>` to the `cypher-shell` command to be able to connect to your cluster member.

3. Recreate the admin role with its original capabilities.

```
CREATE ROLE admin;
GRANT ALL DBMS PRIVILEGES ON DBMS TO admin;
GRANT TRANSACTION MANAGEMENT ON DATABASE * TO admin;
GRANT START ON DATABASE * TO admin;
GRANT STOP ON DATABASE * TO admin;
GRANT MATCH {*} ON GRAPH * TO admin;
GRANT WRITE ON GRAPH * TO admin;
GRANT ALL ON DATABASE * TO admin;
```

4. Grant the admin user role to an existing user.

> **Note:**
>
> Before running the `:exit` command, grant the newly created role to a user. Although this is optional, without this step you will have only collected all admin privileges in a role that no one is assigned to.

To grant the role to a user (assuming your existing user is named `neo4j`), you can run `GRANT ROLE admin TO neo4j;`

5. Exit the `cypher-shell` console:

```
:exit;
```

6. Proceed with the post-recovery steps as per your deployment.

Post-recovery steps

1. Stop Neo4j:

```
bin/neo4j stop
```

2. Enable the authentication and restore your Neo4j to its original configuration (See Disable authentication).
3. Start Neo4j:

```
bin/neo4j start
```

1. Stop the cluster members.

```
bin/neo4j stop
```

2. Enable the authentication and restore each cluster member to its original configuration (See Disable authentication).
3. Start the cluster (all cluster members):

```
bin/neo4j start
```

Manage privileges

Role-based access control

Role-based access control (*RBAC*) is a secure method for granting users access by defining roles with specific privileges and then mapping users to those roles. These privileges define what users can and cannot do within the database, such as which data they can read or write, and which administrative actions they can perform. Therefore, it is very important to apply the principle of least privilege when defining user roles and privileges, so that users are not granted more access than necessary. Or in other words, users should be granted only the minimum level of access necessary to perform their tasks, and no more.

Privileges control the access rights to graph elements using a combined allowlist/denylist mechanism. It is possible to grant or deny access, or use a combination of the two. You are able to access a resource if you have a `GRANT` (allowlist) and do not have a `DENY` (denylist) relevant to that resource. All other combinations of `GRANT` and `DENY` result in the matching path being inaccessible. What this means in practice depends on whether you have a read privilege or a write privilege:

- If an entity is not accessible due to read privileges, the data is invisible. It appears to you as if you had a smaller database (smaller graph).
- If an entity is not accessible due to write privileges, an error occurs on any attempt to write that data.

Note:

This page often uses the terms *'allows'* and *'enables'* in seemingly identical ways. However, there is a subtle difference. *'enables'* refers to the consequences of read privileges where a restriction will not cause an error, only a reduction in the apparent graph size. *'allows'* refers to the consequence of write privileges where a restriction can result in an error.

Note:

If a user was not also provided with the database `ACCESS` privilege, then access to the entire database will be denied. Information about the database access privilege can be found in The ACCESS privilege.

Note:

For more details about the syntax descriptions, see Database management command syntax.

Graph privilege commands

Administrators can use the Cypher commands `GRANT`, `DENY`, and `REVOKE` to manage Neo4j graph administrative rights. These commands can be used to grant or deny privileges to roles, and to revoke previously granted or denied privileges. The privileges can be granted or denied on the entire graph or

specific elements within the graph. The privileges can also be made immutable, which means they cannot be granted, denied, or revoked unless auth is disabled.

Components of the graph privilege commands

The components of the graph privilege commands are:

- *the command*:
  - `GRANT` – gives privileges to roles.
  - `DENY` – denies privileges to roles.
  - `REVOKE` – removes granted or denied privileges from roles.
- *mutability*:
  - `IMMUTABLE` can optionally be specified when performing a `GRANT` or `DENY` to indicate that the privilege cannot be subsequently removed unless auth is disabled. Auth must also be disabled in order to `GRANT` or `DENY` an immutable privilege. Contrastingly, when `IMMUTABLE` is specified in conjunction with a `REVOKE` command, it will act as a filter and only remove matching *immutable* privileges. Starting from Neo4j 5.26, immutable privileges can also be used together with immutable roles. See Immutable roles and privileges for more information.
- *graph-privilege*:
  - Can be either a read privilege or write privilege.
- *name*:
  - The graph or graphs to associate the privilege with. Because in Neo4j 5 you can have only one graph per database, this command uses the database name or alias to refer to that graph. When using an alias, the command will be executed on the resolved graph.

    > **Note:**
    >
    > If you delete a database and create a new one with the same name, the new one will *NOT* have the privileges previously assigned to the deleted graph.

  - It can be `*`, which means all graphs. Graphs created after this command execution will also be associated with these privileges.
  - `HOME GRAPH` refers to the graph associated with the home database for that user. The default database will be used as home database if a user does not have one configured. If the user's home database changes for any reason after privileges have been created, then these privileges will be associated with the graph attached to the new database. This can be quite powerful as it allows permissions to be switched from one graph to another simply by changing a user's home database.
- *entity*
  - The graph elements this privilege applies to:
    - `NODES` label (nodes with the specified label(s)).
    - `RELATIONSHIPS` type (relationships of the specific type(s)).
    - `ELEMENTS` label (both nodes and relationships).

        - `FOR` pattern (nodes that match the pattern). See Property-based access control for details
    - The label or type can be referred with `*`, which means all labels or types.
    - Multiple labels or types can be specified, comma-separated.
    - Defaults to `ELEMENTS *` if omitted.
    - Some of the commands for write privileges do not allow an *entity* part. See Write privileges for details.
    - The `FOR` pattern *entity* is not supported for write privileges.
- *role[, ...]*
    - The role or roles to associate the privilege with, comma-separated.

General syntax for graph privilege commands

Table 440. General grant ON GRAPH privilege syntax

Command	`GRANT ... ON ... TO ...`
Syntax	`GRANT [IMMUTABLE] graph-privilege ON { HOME GRAPH | GRAPH[S] { * | name[, ...] } } [entity] TO role[, ...]`
Description	Grants a privilege to one or multiple roles.

Table 441. General deny ON GRAPH privilege syntax

Command	`DENY ... ON ... TO ...`
Syntax	`DENY [IMMUTABLE] graph-privilege ON { HOME GRAPH | GRAPH[S] { * | name[, ...] } } [entity] TO role[, ...]`
Description	Denies a privilege to one or multiple roles.

Table 442. General revoke ON GRAPH privilege syntax

Command	`REVOKE GRANT ... ON ... FROM ...`
Syntax	`REVOKE [IMMUTABLE] GRANT graph-privilege ON { HOME GRAPH | GRAPH[S] { * | name[, ...] } } [entity] FROM role[, ...]`
Description	Revokes a granted privilege from one or multiple roles.

Table 443. General revoke ON GRAPH privilege syntax

Command	`REVOKE DENY ... ON ... FROM ...`
Syntax	`REVOKE [IMMUTABLE] DENY graph-privilege ON { HOME GRAPH | GRAPH[S] {* | name[, ...] } } [entity] FROM role[, ...]`
Description	Revokes a denied privilege from one or multiple roles.

Table 444. General revoke ON GRAPH privilege syntax

Command	`REVOKE ... ON ... FROM ...`
Syntax	`REVOKE [IMMUTABLE] graph-privilege ON { HOME GRAPH | GRAPH[S] { * | name[, ...] } } [entity] FROM role[, ...]`
Description	Revokes a granted or denied privilege from one or multiple roles.

> **Note:**
>
> `DENY` does NOT erase a granted privilege; they both exist. Use `REVOKE` if you want to remove a privilege.

Common errors, such as misspellings or attempts to revoke privileges that have not been granted or denied, will result in notifications. Some of these notifications may be replaced with errors in a future major version of Neo4j. See Status Codes → Notification codes for details on notifications.

The general `GRANT` and `DENY` syntaxes are illustrated in the following image:

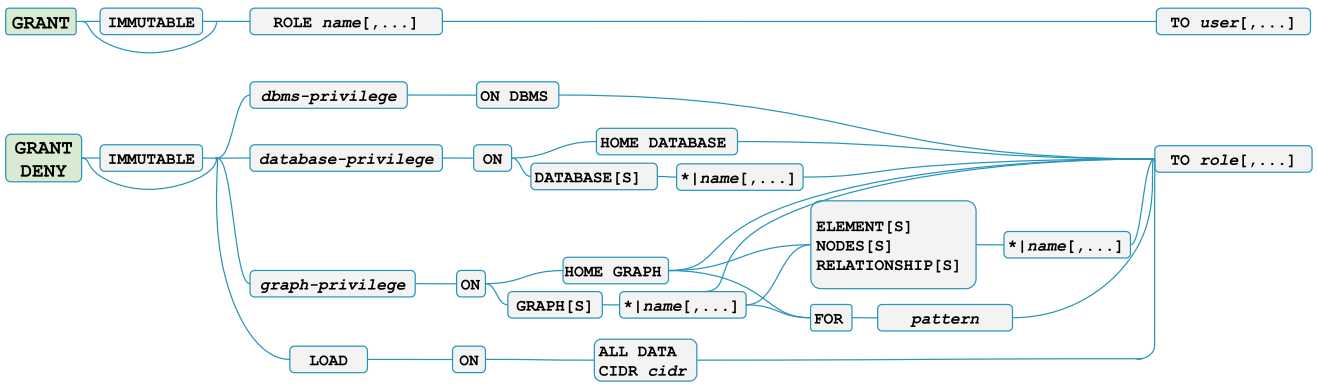


Figure 28. GRANT and DENY Syntax

A more detailed syntax illustration for graph privileges would be the following:

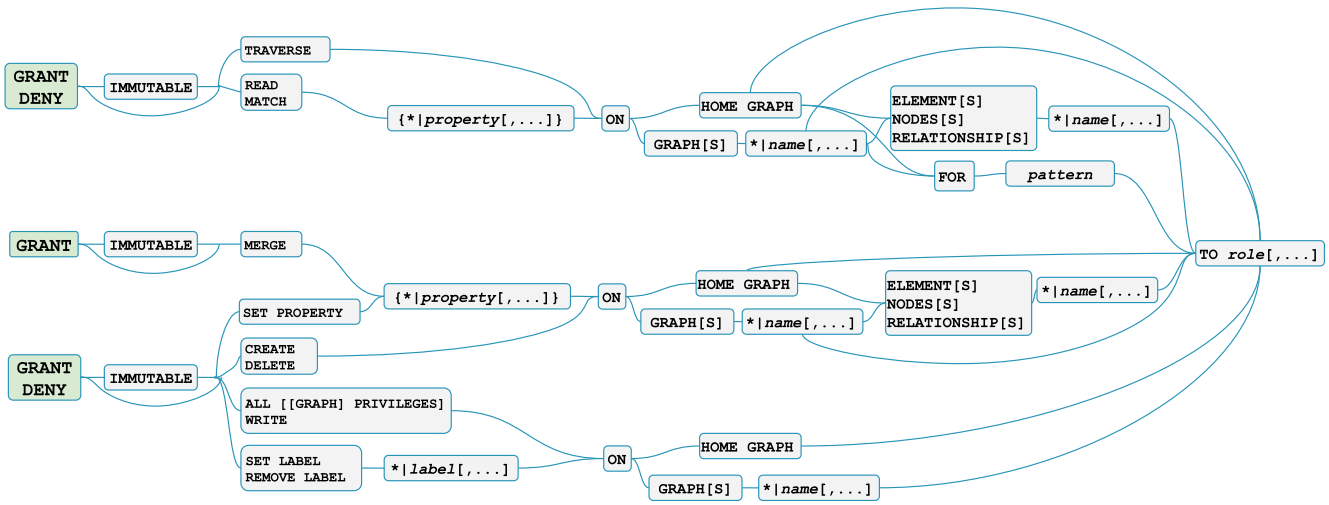


Figure 29. Syntax of GRANT and DENY Graph Privileges. The `{` and `}` are part of the syntax and not used for grouping.

The following image shows the hierarchy between different graph privileges:

Figure 30. Graph privileges hierarchy

Listing supported privileges

Introduced in 5.9

Supported privileges can be displayed using the `SHOW SUPPORTED PRIVILEGES` command. This lists the privileges that are possible to grant or deny on a server, together with the structure of the privilege.

Table 445. Show supported privileges command syntax

Command	`SHOW SUPPORTED PRIVILEGES`

Syntax	`SHOW SUPPORTED PRIVILEGE[S]`
	`  [YIELD { * | field[, ...] } [ORDER BY field[, ...]] [SKIP n] [LIMIT n]]`
	`  [WHERE expression]`
	`  [RETURN field[, ...] [ORDER BY field[, ...]] [SKIP n] [LIMIT n]]`

Description	List all privileges supported by the server.

When using the `RETURN` clause, the `YIELD` clause is mandatory and must not be omitted.

Results will include multiple columns describing the privileges:

Column	Description	Type
`action`	The privilege action.	`STRING`
`qualifier`	Qualifier to further limit the target of the privilege (`function`, `label`, `procedure`, `property`, `setting`, `username`) or null if not applicable.	`STRING`
`target`	Target of the privilege: `dbms`, `database`, `graph`, `cidr`, or `all data`.	`STRING`
`scope`	List of possible scopes for the privilege (`elements`, `nodes`, `pattern`, `relationships`) or null if not applicable.	`LIST OF STRING`

Column	Description	Type
`description`	A short description of the privilege.	`STRING`

If a privilege lists a qualifier, it has to be used in the command by either an identifier or `*` if it should affect all identifiers. The below table demonstrates how qualifiers are used:

qualifier	example
`function`	`... EXECUTE FUNCTION abc* ON …`
`label`	`... SET LABEL A ON …`
`procedure`	`... EXECUTE BOOSTED PROCEDURE apoc.* ON …`
`property`	`... READ {property} ON …`
`setting`	`... SHOW SETTINGS dbms.* ON …`
`username`	`... IMPERSONATE (username) ON …`

It is optional to specify the scope of a privilege. If it is not specified, the default scope will be `ELEMENT *`. Note that not all privileges have a scope.

Examples for listing supported privileges

```
SHOW SUPPORTED PRIVILEGES YIELD * ORDER BY action DESC LIMIT 10 RETURN action, qualifier, target, scope,
description
```

Lists 10 supported privileges:

Table 446. Result

action	qualifier	target	scope	description
"write"	NULL	"graph"	NULL	"allows all WRITE operations on an entire graph"
"user management"	NULL	"dbms"	NULL	"enables the specified roles to create, delete, modify, and list users"
"traverse"	NULL	"graph"	["elements", "nodes", "pattern", "relationships"]	"enables the specified entities to be found"
"transaction management"	"username"	"database"	NULL	"allows listing and ending transactions and queries for the specified users on the specified database"

action	qualifier	target	scope	description
"terminate transactions"	"username"	"database"	NULL	"allows ending transactions and queries for the specified users on the specified database"
"stop"	NULL	"database"	NULL	"allows the specified database to be stopped"
"start"	NULL	"database"	NULL	"allows the specified database to be started"
"show user"	NULL	"dbms"	NULL	"enables the specified roles to list users"
"show transactions"	"username"	"database"	NULL	"allows listing transactions and queries for the specified users on the specified database"
"show settings"	"setting"	"dbms"	NULL	"enables the specified roles to query given configuration settings"
Rows: 10				

Listing assigned privileges

Privileges that have been granted or denied to roles can be displayed using the following `SHOW PRIVILEGE[S]` commands.

Table 447. Show privileges command syntax

Command	`SHOW PRIVILEGE`

Syntax	`SHOW [ALL] PRIVILEGE[S] [AS [REVOKE] COMMAND[S]]`
	`  [YIELD { * | field[, ...] } [ORDER BY field[, ...]] [SKIP n] [LIMIT n]]`
	`  [WHERE expression]`
	`  [RETURN field[, ...] [ORDER BY field[, ...]] [SKIP n] [LIMIT n]]`

Description	List all granted or denied privileges.

Table 448. Show role privileges syntax

Command	`SHOW ROLE ... PRIVILEGE`

Syntax	`SHOW ROLE[S] name[, ...] PRIVILEGE[S] [AS [REVOKE] COMMAND[S]]`
	`  [YIELD { * | field[, ...] } [ORDER BY field[, ...]] [SKIP n] [LIMIT n]]`
	`  [WHERE expression]`
	`  [RETURN field[, ...] [ORDER BY field[, ...]] [SKIP n] [LIMIT n]]`

Description	List privileges granted or denied to a specific role.

Table 449. Show user privileges syntax

Command	`SHOW USER ... PRIVILEGE`

Syntax	`SHOW USER[S] [name[, ...]] PRIVILEGE[S] [AS [REVOKE] COMMAND[S]]`
	`  [YIELD { * | field[, ...] } [ORDER BY field[, ...]] [SKIP n] [LIMIT n]]`
	`  [WHERE expression]`
	`  [RETURN field[, ...] [ORDER BY field[, ...]] [SKIP n] [LIMIT n]]`

Description	List privileges for a specific user, or the current user. [NOTE] ==== **Note:** Please note that it is only possible for a user to show their own privileges. Therefore, if a non-native auth provider like LDAP is in use, `SHOW USER PRIVILEGES` will only work in a limited capacity. Other users' privileges cannot be listed when using a non-native auth provider. ====

When using the `RETURN` clause, the `YIELD` clause is mandatory and must not be omitted.

For an easy overview of the existing privileges, it is recommended to use the `AS COMMANDS` version of the `SHOW` command, which returns two columns.

Table 450. `SHOW PRIVILEGES AS COMMANDS` *output*

Column	Description	Type
command	The privilege as the command that is granted or denied. Or in the `AS REVOKE COMMANDS` case, the command to revoke the privilege. Default Output	`STRING`
immutable	Whether or not the privilege is immutable.	`BOOLEAN`

Alternatively, you can omit the `AS COMMANDS` clause and get the full details of the privileges returned in multiple columns. They are all returned by default without requiring a `YIELD`.

Table 451. `SHOW PRIVILEGES` *output*

Column	Description	Type
`access`	Whether the privilege is granted or denied.	`STRING`
`action`	The type of the privilege. E.g., traverse, read, index management, or role management.	`STRING`
`resource`	The scope of the privilege. E.g., the entire DBMS, a specific database, a graph, or sub-graph access.	`STRING`

Column	Description	Type
graph	The specific database or graph the privilege applies to.	STRING
segment	The labels, relationship types, pattern, procedures, functions, transactions or settings the privilege applies to (if applicable).	STRING
role	The role the privilege is granted to.	STRING
immutable	Whether or not the privilege is immutable.	BOOLEAN
user	The user the privilege belongs to. Note that this is only returned for `SHOW USER [username] PRIVILEGES`.	STRING

Examples for listing all privileges

Assigned privileges can be displayed using the different `SHOW PRIVILEGE[S]` commands.

```
SHOW [ALL] PRIVILEGE[S] [AS [REVOKE] COMMAND[S]]
  [WHERE expression]

SHOW [ALL] PRIVILEGE[S] [AS [REVOKE] COMMAND[S]]
  YIELD { * | field[, ...] } [ORDER BY field[, ...]] [SKIP n] [LIMIT n]
  [WHERE expression]
  [RETURN field[, ...] [ORDER BY field[, ...]] [SKIP n] [LIMIT n]]
```

```
SHOW PRIVILEGES
```

Lists all privileges for all roles:

Table 452. Result

access	action	resource	graph	segment	role	immutable
"GRANTED"	"execute"	"database"	"*"	"FUNCTION(*)"	"PUBLIC"	false
"GRANTED"	"execute"	"database"	"*"	"PROCEDURE(*)"	"PUBLIC"	false
"GRANTED"	"access"	"database"	"DEFAULT"	"database"	"PUBLIC"	false
"GRANTED"	"match"	"all_properties"	"*"	"NODE(*)"	"admin"	false
"GRANTED"	"write"	"graph"	"*"	"NODE(*)"	"admin"	false
"GRANTED"	"match"	"all_properties"	"*"	"RELATIONSHIP(*)"	"admin"	false
"GRANTED"	"write"	"graph"	"*"	"RELATIONSHIP(*)"	"admin"	false
"GRANTED"	"transaction_management"	"database"	"*"	"USER(*)"	"admin"	false
"GRANTED"	"access"	"database"	"*"	"database"	"admin"	false
"GRANTED"	"constraint"	"database"	"*"	"database"	"admin"	false

access	action	resource	graph	segment	role	immutable
"GRANTED"	"dbms_actions"	"database"	"*"	"database"	"admin"	false
"GRANTED"	"index"	"database"	"*"	"database"	"admin"	false
"GRANTED"	"start_database"	"database"	"*"	"database"	"admin"	false
"GRANTED"	"stop_database"	"database"	"*"	"database"	"admin"	false
"GRANTED"	"token"	"database"	"*"	"database"	"admin"	false
"GRANTED"	"match"	"all_properties"	"*"	"NODE(*)"	"architect"	false
"GRANTED"	"write"	"graph"	"*"	"NODE(*)"	"architect"	false
"GRANTED"	"match"	"all_properties"	"*"	"RELATIONSHIP(*)"	"architect"	false
"GRANTED"	"write"	"graph"	"*"	"RELATIONSHIP(*)"	"architect"	false
"GRANTED"	"access"	"database"	"*"	"database"	"architect"	false
"GRANTED"	"constraint"	"database"	"*"	"database"	"architect"	false
"GRANTED"	"index"	"database"	"*"	"database"	"architect"	false
"GRANTED"	"token"	"database"	"*"	"database"	"architect"	false
"GRANTED"	"match"	"all_properties"	"*"	"NODE(*)"	"editor"	false
"GRANTED"	"write"	"graph"	"*"	"NODE(*)"	"editor"	false
"GRANTED"	"match"	"all_properties"	"*"	"RELATIONSHIP(*)"	"editor"	false
"GRANTED"	"write"	"graph"	"*"	"RELATIONSHIP(*)"	"editor"	false
"GRANTED"	"access"	"database"	"*"	"database"	"editor"	false
"DENIED"	"access"	"database"	"neo4j"	"database"	"noAccessUsers"	false
"GRANTED"	"match"	"all_properties"	"*"	"NODE(*)"	"publisher"	false
"GRANTED"	"write"	"graph"	"*"	"NODE(*)"	"publisher"	false
"GRANTED"	"match"	"all_properties"	"*"	"RELATIONSHIP(*)"	"publisher"	false
"GRANTED"	"write"	"graph"	"*"	"RELATIONSHIP(*)"	"publisher"	false
"GRANTED"	"access"	"database"	"*"	"database"	"publisher"	false

access	action	resource	graph	segment	role	immutable
"GRANTED"	"token"	"database"	"*"	"database"	"publisher"	false
"GRANTED"	"match"	"all_properties"	"*"	"NODE(*)"	"reader"	false
"GRANTED"	"match"	"all_properties"	"*"	"RELATIONSHIP(*)"	"reader"	false
"GRANTED"	"access"	"database"	"*"	"database"	"reader"	false
"GRANTED"	"access"	"database"	"neo4j"	"database"	"regularUsers"	false
Rows: 39						

Note:

The `token` action corresponds to the `NAME MANAGEMENT` privilege.

It is also possible to filter and sort the results by using `YIELD`, `ORDER BY` and `WHERE`:

```
SHOW PRIVILEGES YIELD role, access, action, segment
ORDER BY action
WHERE role = 'admin'
```

In this example:

- The number of columns returned has been reduced with the `YIELD` clause.
- The order of the returned columns has been changed.
- The results have been filtered to only return the `admin` role using a `WHERE` clause.
- The results are ordered by the `action` column using `ORDER BY`.

`SKIP` and `LIMIT` can also be used to paginate the results.

Table 453. Result

role	access	action	segment
"admin"	"GRANTED"	"access"	"database"
"admin"	"GRANTED"	"constraint"	"database"
"admin"	"GRANTED"	"dbms_actions"	"database"
"admin"	"GRANTED"	"index"	"database"
"admin"	"GRANTED"	"match"	"NODE(*)"
"admin"	"GRANTED"	"match"	"RELATIONSHIP(*)"
"admin"	"GRANTED"	"start_database"	"database"
"admin"	"GRANTED"	"stop_database"	"database"

role	access	action	segment
"admin"	"GRANTED"	"token"	"database"
"admin"	"GRANTED"	"transaction_management"	"USER(*)"
"admin"	"GRANTED"	"write"	"NODE(*)"
"admin"	"GRANTED"	"write"	"RELATIONSHIP(*)"

Rows: 12

Note:

The `token` action corresponds to the `NAME MANAGEMENT` privilege.

`WHERE` can also be used without `YIELD`:

```
SHOW PRIVILEGES
WHERE graph <> '*'
```

In this example, the `WHERE` clause is used to filter privileges down to those that target specific graphs only.

Table 454. Result

access	action	graph	resource	role	segment
"GRANTED"	"access"	"DEFAULT"	"database"	"PUBLIC"	"database"
"DENIED"	"access"	"neo4j"	"database"	"noAccessUsers"	"database"
"GRANTED"	"access"	"neo4j"	"database"	"regularUsers"	"database"

Rows: 3

Aggregations in the `RETURN` clause can be used to group privileges. In this case, by user and `GRANTED` or `DENIED`:

```
SHOW PRIVILEGES YIELD * RETURN role, access, collect([graph, resource, segment, action]) AS privileges
```

Table 455. Result

role	access	privileges
"PUBLIC"	"GRANTED"	[["*","database","FUNCTION(*)","execute"],["*","database","PROCEDURE(*)","execute"],["DEFAULT","database","database","access"]]

role	access	privileges
"admin"	"GRANTED"	[["*","all_properties","NODE(*)","match"],["*","graph","NODE(*)","write"],["*","all_properties","RELATIONSHIP(*)","match"],["*","graph","RELATIONSHIP(*)","write"],["*","database","USER(*)","transaction_management"],["*","database","database","access"],["*","database","database","constraint"],["*","database","database","dbms_actions"],["*","database","database","index"],["*","database","database","start_database"],["*","database","database","stop_database"],["*","database","database","token"]]
"architect"	"GRANTED"	[["*","all_properties","NODE(*)","match"],["*","graph","NODE(*)","write"],["*","all_properties","RELATIONSHIP(*)","match"],["*","graph","RELATIONSHIP(*)","write"],["*","database","database","access"],["*","database","database","constraint"],["*","database","database","index"],["*","database","database","token"]]
"editor"	"GRANTED"	[["*","all_properties","NODE(*)","match"],["*","graph","NODE(*)","write"],["*","all_properties","RELATIONSHIP(*)","match"],["*","graph","RELATIONSHIP(*)","write"],["*","database","database","access"]]
"noAccessUsers"	"DENIED"	[["neo4j","database","database","access"]]
"publisher"	"GRANTED"	[["*","all_properties","NODE(*)","match"],["*","graph","NODE(*)","write"],["*","all_properties","RELATIONSHIP(*)","match"],["*","graph","RELATIONSHIP(*)","write"],["*","database","database","access"],["*","database","database","token"]]
"reader"	"GRANTED"	[["*","all_properties","NODE(*)","match"],["*","all_properties","RELATIONSHIP(*)","match"],["*","database","database","access"]]
"regularUsers"	"GRANTED"	[["neo4j","database","database","access"]]
Rows: 8		

Note:

The `token` action corresponds to the `NAME MANAGEMENT` privilege.

The `RETURN` clause can also be used to order and paginate the results, which is useful when combined with `YIELD` and `WHERE`. In this example the query returns privileges for display five-per-page, and skips the first five to display the second page.

```
SHOW PRIVILEGES YIELD * RETURN * ORDER BY role SKIP 5 LIMIT 5
```

Table 456. Result

access	action	graph	resource	role	segment	immutable
"GRANTED"	"match"	"*"	"all_properties"	"admin"	"RELATIONSHIP(*)"	false
"GRANTED"	"write"	"*"	"graph"	"admin"	"RELATIONSHIP(*)"	false

access	action	graph	resource	role	segment	immutable
"GRANTED"	"transaction_management"	"*"	"database"	"admin"	"USER(*)"	false
"GRANTED"	"access"	"*"	"database"	"admin"	"database"	false
"GRANTED"	"constraint"	"*"	"database"	"admin"	"database"	false
Rows: 5						

Available privileges can also be displayed as Cypher commands by adding `AS COMMAND[S]`:

```
SHOW PRIVILEGES AS COMMANDS
```

Table 457. Result

command
"DENY ACCESS ON DATABASE `neo4j` TO `noAccessUsers`"
"GRANT ACCESS ON DATABASE * TO `admin`"
"GRANT ACCESS ON DATABASE * TO `architect`"
"GRANT ACCESS ON DATABASE * TO `editor`"
"GRANT ACCESS ON DATABASE * TO `publisher`"
"GRANT ACCESS ON DATABASE * TO `reader`"
"GRANT ACCESS ON DATABASE `neo4j` TO `regularUsers`"
"GRANT ACCESS ON HOME DATABASE TO `PUBLIC`"
"GRANT ALL DBMS PRIVILEGES ON DBMS TO `admin`"
"GRANT CONSTRAINT MANAGEMENT ON DATABASE * TO `admin`"
"GRANT CONSTRAINT MANAGEMENT ON DATABASE * TO `architect`"
"GRANT EXECUTE FUNCTION * ON DBMS TO `PUBLIC`"
"GRANT EXECUTE PROCEDURE * ON DBMS TO `PUBLIC`"
"GRANT INDEX MANAGEMENT ON DATABASE * TO `admin`"
"GRANT INDEX MANAGEMENT ON DATABASE * TO `architect`"
"GRANT MATCH {*} ON GRAPH * NODE * TO `admin`"
"GRANT MATCH {*} ON GRAPH * NODE * TO `architect`"
"GRANT MATCH {*} ON GRAPH * NODE * TO `editor`"
"GRANT MATCH {*} ON GRAPH * NODE * TO `publisher`"
"GRANT MATCH {*} ON GRAPH * NODE * TO `reader`"
"GRANT MATCH {*} ON GRAPH * RELATIONSHIP * TO `admin`"

command
"GRANT MATCH {*} ON GRAPH * RELATIONSHIP * TO `architect`"
"GRANT MATCH {*} ON GRAPH * RELATIONSHIP * TO `editor`"
"GRANT MATCH {*} ON GRAPH * RELATIONSHIP * TO `publisher`"
"GRANT MATCH {*} ON GRAPH * RELATIONSHIP * TO `reader`"
"GRANT NAME MANAGEMENT ON DATABASE * TO `admin`"
"GRANT NAME MANAGEMENT ON DATABASE * TO `architect`"
"GRANT NAME MANAGEMENT ON DATABASE * TO `publisher`"
"GRANT START ON DATABASE * TO `admin`"
"GRANT STOP ON DATABASE * TO `admin`"
"GRANT TRANSACTION MANAGEMENT (*) ON DATABASE * TO `admin`"
"GRANT WRITE ON GRAPH * TO `admin`"
"GRANT WRITE ON GRAPH * TO `architect`"
"GRANT WRITE ON GRAPH * TO `editor`"
"GRANT WRITE ON GRAPH * TO `publisher`"
Rows: 35

Like other `SHOW` commands, the output can also be processed using `YIELD` / `WHERE` / `RETURN`:

```
SHOW PRIVILEGES AS COMMANDS
WHERE command CONTAINS 'MANAGEMENT'
```

Table 458. Result

command
"GRANT CONSTRAINT MANAGEMENT ON DATABASE * TO `admin`"
"GRANT CONSTRAINT MANAGEMENT ON DATABASE * TO `architect`"
"GRANT INDEX MANAGEMENT ON DATABASE * TO `admin`"
"GRANT INDEX MANAGEMENT ON DATABASE * TO `architect`"
"GRANT NAME MANAGEMENT ON DATABASE * TO `admin`"
"GRANT NAME MANAGEMENT ON DATABASE * TO `architect`"
"GRANT NAME MANAGEMENT ON DATABASE * TO `publisher`"
"GRANT TRANSACTION MANAGEMENT (*) ON DATABASE * TO `admin`"
Rows: 8

It is also possible to get the privileges listed as revoking commands instead of granting or denying:

```
SHOW PRIVILEGES AS REVOKE COMMANDS
```

Table 459. Result

command
"REVOKE DENY ACCESS ON DATABASE `neo4j` FROM `noAccessUsers`"
"REVOKE GRANT ACCESS ON DATABASE * FROM `admin`"
"REVOKE GRANT ACCESS ON DATABASE * FROM `architect`"
"REVOKE GRANT ACCESS ON DATABASE * FROM `editor`"
"REVOKE GRANT ACCESS ON DATABASE * FROM `publisher`"
"REVOKE GRANT ACCESS ON DATABASE * FROM `reader`"
"REVOKE GRANT ACCESS ON DATABASE `neo4j` FROM `regularUsers`"
"REVOKE GRANT ACCESS ON HOME DATABASE FROM `PUBLIC`"
"REVOKE GRANT ALL DBMS PRIVILEGES ON DBMS FROM `admin`"
"REVOKE GRANT CONSTRAINT MANAGEMENT ON DATABASE * FROM `admin`"
"REVOKE GRANT CONSTRAINT MANAGEMENT ON DATABASE * FROM `architect`"
"REVOKE GRANT EXECUTE FUNCTION * ON DBMS FROM `PUBLIC`"
"REVOKE GRANT EXECUTE PROCEDURE * ON DBMS FROM `PUBLIC`"
"REVOKE GRANT INDEX MANAGEMENT ON DATABASE * FROM `admin`"
"REVOKE GRANT INDEX MANAGEMENT ON DATABASE * FROM `architect`"
"REVOKE GRANT MATCH {*} ON GRAPH * NODE * FROM `admin`"
"REVOKE GRANT MATCH {*} ON GRAPH * NODE * FROM `architect`"
"REVOKE GRANT MATCH {*} ON GRAPH * NODE * FROM `editor`"
"REVOKE GRANT MATCH {*} ON GRAPH * NODE * FROM `publisher`"
"REVOKE GRANT MATCH {*} ON GRAPH * NODE * FROM `reader`"
"REVOKE GRANT MATCH {*} ON GRAPH * RELATIONSHIP * FROM `admin`"
"REVOKE GRANT MATCH {*} ON GRAPH * RELATIONSHIP * FROM `architect`"
"REVOKE GRANT MATCH {*} ON GRAPH * RELATIONSHIP * FROM `editor`"
"REVOKE GRANT MATCH {*} ON GRAPH * RELATIONSHIP * FROM `publisher`"
"REVOKE GRANT MATCH {*} ON GRAPH * RELATIONSHIP * FROM `reader`"
"REVOKE GRANT NAME MANAGEMENT ON DATABASE * FROM `admin`"
"REVOKE GRANT NAME MANAGEMENT ON DATABASE * FROM `architect`"
"REVOKE GRANT NAME MANAGEMENT ON DATABASE * FROM `publisher`"
"REVOKE GRANT START ON DATABASE * FROM `admin`"

command
"REVOKE GRANT STOP ON DATABASE * FROM `admin`"
"REVOKE GRANT TRANSACTION MANAGEMENT (*) ON DATABASE * FROM `admin`"
"REVOKE GRANT WRITE ON GRAPH * FROM `admin`"
"REVOKE GRANT WRITE ON GRAPH * FROM `architect`"
"REVOKE GRANT WRITE ON GRAPH * FROM `editor`"
"REVOKE GRANT WRITE ON GRAPH * FROM `publisher`"
Rows: 35

For more info about revoking privileges, please see The REVOKE command.

Examples for listing privileges for specific roles

Available privileges for specific roles can be displayed using `SHOW ROLE name PRIVILEGE[S]`:

```
SHOW ROLE[S] name[, ...] PRIVILEGE[S] [AS [REVOKE] COMMAND[S]]
  [WHERE expression]

SHOW ROLE[S] name[, ...] PRIVILEGE[S] [AS [REVOKE] COMMAND[S]]
  YIELD { * | field[, ...] } [ORDER BY field[, ...]] [SKIP n] [LIMIT n]
  [WHERE expression]
  [RETURN field[, ...] [ORDER BY field[, ...]] [SKIP n] [LIMIT n]]
```

```
SHOW ROLE regularUsers PRIVILEGES
```

Lists all privileges for role `regularUsers`.

Table 460. Result

access	action	graph	resource	role	segment	immutable
"GRANTED"	"access"	"database"	"neo4j"	"database"	"regularUsers"	false
Rows: 1						

```
SHOW ROLES regularUsers, noAccessUsers PRIVILEGES
```

Lists all privileges for roles `regularUsers` and `noAccessUsers`.

Table 461. Result

access	action	graph	resource	role	segment	immutable
"DENIED"	"access"	"database"	"neo4j"	"database"	"noAccessUsers"	false
"GRANTED"	"access"	"database"	"neo4j"	"database"	"regularUsers"	false
Rows: 2						

Similar to the other `SHOW PRIVILEGES` commands, the available privileges for roles can also be listed as Cypher commands with the optional `AS COMMAND[S]`.

```
SHOW ROLES regularUsers, noAccessUsers PRIVILEGES AS COMMANDS
```

Table 462. Result

command
"GRANT ACCESS ON DATABASE * TO `admin`"
"GRANT ALL DBMS PRIVILEGES ON DBMS TO `admin`"
"GRANT CONSTRAINT MANAGEMENT ON DATABASE * TO `admin`"
"GRANT INDEX MANAGEMENT ON DATABASE * TO `admin`"
"GRANT MATCH {*} ON GRAPH * NODE * TO `admin`"
"GRANT MATCH {*} ON GRAPH * RELATIONSHIP * TO `admin`"
"GRANT NAME MANAGEMENT ON DATABASE * TO `admin`"
"GRANT START ON DATABASE * TO `admin`"
"GRANT STOP ON DATABASE * TO `admin`"
"GRANT TRANSACTION MANAGEMENT (*) ON DATABASE * TO `admin`"
"GRANT WRITE ON GRAPH * TO `admin`"
Rows: 11

The output can be processed using `YIELD` / `WHERE` / `RETURN` here as well:

```
SHOW ROLE architect PRIVILEGES AS COMMANDS WHERE command CONTAINS 'MATCH'
```

Table 463. Result

command
"GRANT MATCH {*} ON GRAPH * NODE * TO `architect`"
"GRANT MATCH {*} ON GRAPH * RELATIONSHIP * TO `architect`"
Rows: 2

Again, it is possible to get the privileges listed as revoking commands instead of granting or denying. For more info about revoking privileges, please see The REVOKE command.

```
SHOW ROLE reader PRIVILEGES AS REVOKE COMMANDS
```

Table 464. Result

command
"REVOKE GRANT ACCESS ON DATABASE * FROM `reader`"
"REVOKE GRANT MATCH {*} ON GRAPH * NODE * FROM `reader`"
"REVOKE GRANT MATCH {*} ON GRAPH * RELATIONSHIP * FROM `reader`"
Rows: 3

Examples for listing privileges for specific users

Available privileges for specific users can be displayed using `SHOW USER name PRIVILEGES`.

> **Note:**
>
> Note that if a non-native auth provider like LDAP is in use, `SHOW USER PRIVILEGES` will only work with a limited capacity as it is only possible for a user to show their own privileges. Other users' privileges cannot be listed when using a non-native auth provider.

```
SHOW USER[S] [name[, ...]] PRIVILEGE[S] [AS [REVOKE] COMMAND[S]]
  [WHERE expression]

SHOW USER[S] [name[, ...]] PRIVILEGE[S] [AS [REVOKE] COMMAND[S]]
  YIELD { * | field[, ...] } [ORDER BY field[, ...]] [SKIP n] [LIMIT n]
  [WHERE expression]
  [RETURN field[, ...] [ORDER BY field[, ...]] [SKIP n] [LIMIT n]]
```

```
SHOW USER jake PRIVILEGES
```

Lists all privileges for user `jake`.

Table 465. Result

access	action	resource	graph	resource	role	segment	immutable
"GRANTED"	"execute"	"database"	"*"	"FUNCTION(*)"	"PUBLIC"	"jake"	false
"GRANTED"	"execute"	"database"	"*"	"PROCEDURE(*)"	"PUBLIC"	"jake"	false
"GRANTED"	"access"	"database"	"DEFAULT"	"database"	"PUBLIC"	"jake"	false
"GRANTED"	"access"	"database"	"neo4j"	"database"	"regularUsers"	"jake"	false
Rows: 4							

```
SHOW USERS jake, joe PRIVILEGES
```

Lists all privileges for users `jake` and `joe`.

Table 466. Result

access	action	resource	graph	resource	role	segment	immutable
"GRANTED"	"execute"	"database"	"*"	"FUNCTION(*)"	"PUBLIC"	"jake"	false
"GRANTED"	"execute"	"database"	"*"	"PROCEDURE(*)"	"PUBLIC"	"jake"	false
"GRANTED"	"access"	"database"	"DEFAULT"	"database"	"PUBLIC"	"jake"	false
"GRANTED"	"access"	"database"	"neo4j"	"database"	"regularUsers"	"jake"	false
"GRANTED"	"execute"	"database"	"*"	"FUNCTION(*)"	"PUBLIC"	"joe"	false
"GRANTED"	"execute"	"database"	"*"	"PROCEDURE(*)"	"PUBLIC"	"joe"	false
"GRANTED"	"access"	"database"	"DEFAULT"	"database"	"PUBLIC"	"joe"	false
"DENIED"	"access"	"database"	"neo4j"	"database"	"noAccessUsers"	"joe"	false

Rows: 8

The same command can be used at all times to review available privileges for the current user. For this purpose, there is a shorter form of the command: `SHOW USER PRIVILEGES`:

```
SHOW USER PRIVILEGES
```

As for the other privilege commands, available privileges for users can also be listed as Cypher commands with the optional `AS COMMAND[S]`.

> **Note:**
>
> When showing user privileges as commands, the roles in the Cypher commands are replaced with a parameter. This can be used to quickly create new roles based on the privileges of specific users.

```
SHOW USER jake PRIVILEGES AS COMMANDS
```

Table 467. Result

command
"GRANT ACCESS ON DATABASE `neo4j` TO $role"
"GRANT ACCESS ON HOME DATABASE TO $role"
"GRANT EXECUTE FUNCTION * ON DBMS TO $role"
"GRANT EXECUTE PROCEDURE * ON DBMS TO $role"

Rows: 4

Like other `SHOW` commands, the output can also be processed using `YIELD` / `WHERE` / `RETURN`. Additionally, similar to the other show privilege commands, it is also possible to show the commands for revoking the privileges.

```
SHOW USER jake PRIVILEGES AS REVOKE COMMANDS
WHERE command CONTAINS 'EXECUTE'
```

Table 468. Result

command
"REVOKE GRANT EXECUTE FUNCTION * ON DBMS FROM $role"
"REVOKE GRANT EXECUTE PROCEDURE * ON DBMS FROM $role"
Rows: 2

Revoking privileges

Privileges that were granted or denied earlier can be revoked using the `REVOKE` command:

```
REVOKE
  [ IMMUTABLE ]
  [ GRANT | DENY ] graph-privilege
  FROM role[, ...]
```

An example usage of the `REVOKE` command is given here:

```
REVOKE GRANT TRAVERSE ON HOME GRAPH NODES Post FROM regularUsers
```

While it can be explicitly specified that `REVOKE` should remove a `GRANT` or `DENY`, it is also possible to `REVOKE` both by not specifying them at all, as the next example demonstrates. Because of this, if there happens to be a `GRANT` and a `DENY` for the same privilege, it would remove both.

```
REVOKE TRAVERSE ON HOME GRAPH NODES Payments FROM regularUsers
```

Adding `IMMUTABLE` explicitly specifies that only immutable privileges should be removed. Omitting it specifies that both immutable and regular privileges should be removed.

Read privileges

There are three separate read privileges:

- `TRAVERSE` - enables the specified entities to be found.
- `READ` - enables the specified properties of the found entities to be read.
- `MATCH` - combines both `TRAVERSE` and `READ`, enabling an entity to be found and its properties read.

For more details about how to read the administration commands syntax, see Reading the administration commands syntax and Components of the graph privilege commands.

The `TRAVERSE` privilege

Users can be granted the right to find nodes and relationships using the `GRANT TRAVERSE` privilege.

```
GRANT [IMMUTABLE] TRAVERSE
  ON { HOME GRAPH | GRAPH[S] { * | name[, ...] } }
    [
      ELEMENT[S] { * | label-or-rel-type[, ...] }
      | NODE[S] { * | label[, ...] }
      | RELATIONSHIP[S] { * | rel-type[, ...] }
      | FOR pattern
    ]
  TO role[, ...]
```

> **Note:**
>
> For more details about the `pattern` syntax used to express attributes based access control rules, see Property-based access control.

For example, you can enable users with the role `regularUsers` to find all nodes with the label `Post` in the database `neo4j`:

```
GRANT TRAVERSE ON GRAPH neo4j NODES Post TO regularUsers
```

The `TRAVERSE` privilege can also be denied.

```
DENY [IMMUTABLE] TRAVERSE
  ON { HOME GRAPH | GRAPH[S] { * | name[, ...] } }
    [
      ELEMENT[S] { * | label-or-rel-type[, ...] }
      | NODE[S] { * | label[, ...] }
      | RELATIONSHIP[S] { * | rel-type[, ...] }
      | FOR pattern
    ]
  TO role[, ...]
```

For example, you can disable users with the role `regularUsers` from finding all nodes with the label `Payments`:

```
DENY TRAVERSE ON HOME GRAPH NODES Payments TO regularUsers
```

Although you just granted the role `regularUsers` the right to read all properties on nodes with label `Post`, you may want to make this more fine-grained using Property-based access control to hide the posts with `secret` property set to `true`. For example:

```
DENY TRAVERSE ON HOME GRAPH FOR (:Post {secret: true}) TO regularUsers
```

> **Note:**
>
> If a label or a relationship type does not exist in the database, the user cannot use the corresponding privilege until it is created. See Privileges for nonexistent labels, relationship

types, and property names for more information.

The `READ` privilege

Users can be granted the right to do property reads on nodes and relationships using the `GRANT READ` privilege. It is very important to note that users can only read properties on entities that they are enabled to find in the first place.

```
GRANT [IMMUTABLE] READ "{" { * | property[, ...] } "}"
  ON { HOME GRAPH | GRAPH[S] { * | name[, ...] } }
    [
      ELEMENT[S] { * | label-or-rel-type[, ...] }
      | NODE[S] { * | label[, ...] }
      | RELATIONSHIP[S] { * | rel-type[, ...] }
      | FOR pattern
    ]
  TO role[, ...]
```

Note:

For more details about the `pattern` syntax used to express attributes based access control rules, see Property-based access control.

For example, you can enable users with the role `regularUsers` to read all properties on nodes with the label `Post` in the database `neo4j`. The `*` implies that the ability to read all properties also extends to properties that might be added in the future.

```
GRANT READ { * } ON GRAPH neo4j NODES Post TO regularUsers
```

To further fine-grained the read access, you can enable users with the role `regularUsers` to read all properties on nodes with the label `Post` that have property `secret` not set to `true` in the database `neo4j`. For example:

```
GRANT READ { * } ON GRAPH neo4j FOR (n:Post) WHERE n.secret <> true TO regularUsers
```

Note:

Granting property `READ` access does not imply that the entities with that property can be found. For example, if there is also a `DENY TRAVERSE` present on the same entity as a `GRANT READ`, the entity will not be found by a Cypher `MATCH` statement.

The `READ` privilege can also be denied.

```
DENY [IMMUTABLE] READ "{" { * | property[, ...] } "}"
  ON { HOME GRAPH | GRAPH[S] { * | name[, ...] } }
    [
      ELEMENT[S] { * | label-or-rel-type[, ...] }
      | NODE[S] { * | label[, ...] }
      | RELATIONSHIP[S] { * | rel-type[, ...] }
      | FOR pattern
    ]
```

```
  TO role[, ...]
```

Although you just granted the role `regularUsers` the right to read all properties, you may want to hide the `secret` property. The following example shows how to do that:

```
DENY READ { secret } ON GRAPH neo4j NODES Post TO regularUsers
```

> **Note:**
>
> If a label, a relationship type, or a property name does not exist in the database, the user cannot use the corresponding privilege until it is created. See Privileges for nonexistent labels, relationship types, and property names for more information.

The `MATCH` privilege

Users can be granted the right to find and do property reads on nodes and relationships using the `GRANT MATCH` privilege. This is semantically the same as having both `TRAVERSE` and `READ` privileges.

```
GRANT [IMMUTABLE] MATCH "{" { * | property[, ...] } "}"
  ON { HOME GRAPH | GRAPH[S] { * | name[, ...] } }
    [
      ELEMENT[S] { * | label-or-rel-type[, ...] }
      | NODE[S] { * | label[, ...] }
      | RELATIONSHIP[S] { * | rel-type[, ...] }
      | FOR pattern
    ]
  TO role[, ...]
```

> **Note:**
>
> For more details about the `pattern` syntax used to express attributes based access control rules, see Property-based access control.

For example if you want to grant the ability to read the properties `language` and `length` for nodes with the label `Message`, as well as the ability to find these nodes to the role `regularUsers`, you can use the following `GRANT MATCH` query:

```
GRANT MATCH { language, length } ON GRAPH neo4j NODES Message TO regularUsers
```

The following query grants the `regularUsers` role the ability to find `Post` and `Likes` nodes where the `secret` property is set to `false`, as well as reading all their properties.

```
GRANT MATCH { * } ON GRAPH neo4j FOR (n:Post|Likes) WHERE n.secret = false TO regularUsers
```

Like all other privileges, the `MATCH` privilege can also be denied.

```
DENY [IMMUTABLE] MATCH "{" { * | property[, ...] } "}"
  ON { HOME GRAPH | GRAPH[S] { * | name[, ...] } }
    [
      ELEMENT[S] { * | label-or-rel-type[, ...] }
```

```
    | NODE[S] { * | label[, ...] }
    | RELATIONSHIP[S] { * | rel-type[, ...] }
    | FOR pattern
  ]
TO role[, ...]
```

Please note that the effect of denying a `MATCH` privilege depends on whether concrete property keys are specified or are `*`. If you specify concrete property keys, then `DENY MATCH` will only deny reading those properties. Finding the elements to traverse would still be enabled. If you specify `*` instead, then both traversal of the element and all property reads will be disabled. The following queries will show examples for this.

Denying to read the property `content` on nodes with the label `Message` for the role `regularUsers` would look like the following query. Although not being able to read this specific property, nodes with that label can still be traversed (and, depending on other grants, other properties on it could still be read).

```
DENY MATCH { content } ON GRAPH neo4j NODES Message TO regularUsers
```

The following query exemplifies how it would look if you wanted to deny both reading all properties and traversing nodes labeled with `Account` in the database `neo4j`:

```
DENY MATCH { * } ON GRAPH neo4j NODES Account TO regularUsers
```

> **Note:**
>
> If a label, a relationship type, or a property name does not exist in the database, the user cannot use the corresponding privilege until it is created. See Privileges for nonexistent labels, relationship types, and property names for more information.

Property-based access control

Property-based access control grants permissions to users to read node properties based on property/value conditions. Each property-based privilege can only be restricted by a single property. For information about read privileges and their syntax, see Read privileges.

> **Important:**
>
> When using property-based access control, ensure the property used for the rule cannot be modified. Users who can change this property can affect the granted property-based privileges.

Syntax

To specify the property/value conditions of the read privilege, you can use the following syntax:

```
{GRANT | DENY | REVOKE [GRANT | DENY]}
[IMMUTABLE]
{MATCH | READ | TRAVERSE}
  ON { HOME GRAPH | GRAPH[S] { * | name[, ...] } }
```

```
    [
      ELEMENT[S] { * | label-or-rel-type[, ...] }
      | NODE[S] { * | label[, ...] }
      | RELATIONSHIP[S] { * | rel-type[, ...] }
      | FOR {

([var][:label["|" ...]] "{" property: value "}")
  | (var[:label["|" ...]])
      WHERE [NOT] var.property { { = | <> | > | >= | < | <= } value | IS NULL | IS NOT NULL | IN {
"["[value[, ...]]"]" | listParam } }
  | (var[:label["|" ...]]
WHERE [NOT] var.property { { = | <> | > | >= | < | <= } value | IS NULL | IS NOT NULL | IN { "["[value[,
...]]"]" | listParam } } )
}

 {TO | FROM} role[, ...]
```

Performance considerations

Adding property-based access control may lead to a significant performance overhead in certain scenarios. See Limitations for more detailed information.

When having property rules, the following factors can worsen the impact on performance:

- The number of properties on the nodes concerned (more properties = greater performance impact).
- The number of property-based privileges (more property-based privileges = greater performance impact).
- The type of the privilege: `TRAVERSE` property-based privileges have greater performance impact than `READ` property-based privileges.
- The type of storage medium in operation. The impact of the property-based privileges on performance is considerably amplified by accessing disc storage.

To reduce the performance impact, it is recommended to use the `block` storage format as it is better optimized for the kind of read required for the resolution of property-based privileges.

For performance-critical scenarios, it is recommended to design privileges based on labels.

Examples

You can use the following syntax for defining a property-based privilege:

```
GRANT privilege-name ON GRAPH graph-name FOR pattern TO role-name
```

Note:

The user role does not need to have `READ` privilege for the property used by the property-based privilege.

Grant a property-based privilege on a specific property using its value

The following example shows how to grant permission to `READ` the `address` property on `Email` or `Website` nodes with domain `exampledomain.com` to role `regularUsers`:

```
GRANT READ { address } ON GRAPH * FOR (n:Email|Website) WHERE n.domain = 'exampledomain.com' TO
regularUsers
```

Alternatively, you can use the following syntax:

```
GRANT READ { address } ON GRAPH * FOR (:Email|Website {domain: 'exampledomain.com'}) TO regularUsers
```

Grant a property-based privilege using `NULL`

The following example shows how to grant permission to `TRAVERSE` nodes with the label `Email` where property `classification` is `NULL` to role `regularUsers`:

```
GRANT TRAVERSE ON GRAPH * FOR (n:Email) WHERE n.classification IS NULL TO regularUsers
```

Deny a property-based privilege using a comparison operator

The following example shows how to deny permission to `READ` and `TRAVERSE` nodes where the property `classification` is different from `UNCLASSIFIED` to role `regularUsers`:

```
DENY MATCH {*} ON GRAPH * FOR (n) WHERE n.classification <> 'UNCLASSIFIED' TO regularUsers
```

Grant a property-based privilege on all properties using a property value

The following example shows how to grant permission to `READ` all properties on nodes where the property `securityLevel` is higher than `3` to role `regularUsers`:

```
GRANT READ {*} ON GRAPH * FOR (n) WHERE n.securityLevel > 3 TO regularUsers
```

Note:

The role `regularUsers` does not need to have `READ` privilege for the property `securityLevel` used by the property-based privilege.

Deny a property-based privilege using a list of values

The following example shows how to deny permission to `READ` all properties on nodes where the property `classification` is not included in the list of `[UNCLASSIFIED, PUBLIC]`:

```
DENY READ {*} ON GRAPH * FOR (n) WHERE NOT n.classification IN ['UNCLASSIFIED', 'PUBLIC'] TO regularUsers
```

Grant a property-based privilege using temporal value

Introduced in 5.26

The following example shows how to grant permission to `READ` all properties on nodes where the property

`createdAt` is later than the current date:

```
GRANT READ {*} ON GRAPH * FOR (n) WHERE n.createdAt > date() TO regularUsers
```

> **Note:**
>
> The `date()` function is evaluated, and the value used to evaluate the privilege is the date when the property-based privilege is created. Keep this in mind when designing your property rules, and use the `SHOW PRIVILEGES AS COMMANDS` command to check the stored value. This is essential when revoking property-based privileges containing evaluated function values like `date()`.

> **Note:**
>
> Not all temporal values are comparable, see Cypher Manual → Equality, ordering, and comparison of value types.

You can show the privilege created by the command in the previous example as a revoke command by running:

```
SHOW ROLE regularUsers PRIVILEGES AS REVOKE COMMANDS
```

Table 469. Result

command
`REVOKE GRANT READ {*} ON GRAPH * FOR (n) WHERE n.createdAt › date('2024-10-25') FROM \`regularUsers\``
Rows: 1

Write privileges

Write privileges are defined for different parts of the graph:

- `CREATE` - allows creating nodes and relationships.
- `DELETE` - allows deleting nodes and relationships.
- `SET LABEL` - allows setting the specified node labels using the `SET` clause.
- `REMOVE LABEL` - allows removing the specified node labels using the `REMOVE` clause.
- `SET PROPERTY` - allows setting properties on nodes and relationships.

There are also compound privileges that combine the above specific privileges:

- `MERGE` - allows `MATCH`, `CREATE`, and `SET PROPERTY` to apply the `MERGE` command.
- `WRITE` - allows all `WRITE` operations on an entire graph.
- `ALL GRAPH PRIVILEGES` - allows all `READ` and `WRITE` operations on an entire graph.

For more details about how to read the administration commands syntax, see Reading the administration commands syntax and Components of the graph privilege commands.

The `CREATE` privilege

The `CREATE` privilege allows a user to create new node and relationship elements on a graph. For more details, see the Cypher Manual → `CREATE` clause.

```
GRANT [IMMUTABLE] CREATE
  ON { HOME GRAPH | GRAPH[S] { * | name[, ...] } }
    [
      ELEMENT[S] { * | label-or-rel-type[, ...] }
      | NODE[S] { * | label[, ...] }
      | RELATIONSHIP[S] { * | rel-type[, ...] }
    ]
  TO role[, ...]
```

For example, to grant the role `regularUsers` the ability to `CREATE` elements on the graph `neo4j`, use:

```
GRANT CREATE ON GRAPH neo4j ELEMENTS * TO regularUsers
```

The `CREATE` privilege can also be denied:

```
DENY [IMMUTABLE] CREATE
  ON { HOME GRAPH | GRAPH[S] { * | name[, ...] } }
    [
      ELEMENT[S] { * | label-or-rel-type[, ...] }
      | NODE[S] { * | label[, ...] }
      | RELATIONSHIP[S] { * | rel-type[, ...] }
    ]
  TO role[, ...]
```

For example, to deny the role `regularUsers` the ability to `CREATE` nodes with the label `foo` on all graphs, use:

```
DENY CREATE ON GRAPH * NODES foo TO regularUsers
```

> **Note:**
>
> If the user attempts to create nodes with a label that does not already exist on the database, then the user must also possess the `CREATE NEW LABEL` privilege. The same applies to new relationships: the `CREATE NEW RELATIONSHIP TYPE` privilege is required.

> **Note:**
>
> If a label or a relationship type does not exist in the database, the user cannot use the corresponding privilege until it is created. See Privileges for nonexistent labels, relationship types, and property names for more information.

The `DELETE` privilege

The `DELETE` privilege allows a user to delete node and relationship elements on a graph. For more details, see the Cypher Manual → `DELETE` clause.

```
GRAM [IMMUTABLE] DELETE
  ON { HOME GRAPH | GRAPH[S] { * | name[, ...] } }
    [
      ELEMENT[S] { * | label-or-rel-type[, ...] }
      | NODE[S] { * | label[, ...] }
      | RELATIONSHIP[S] { * | rel-type[, ...] }
    ]
  TO role[, ...]
```

For example, to grant the role `regularUsers` the ability to `DELETE` elements on the graph `neo4j`, use:

```
GRANT DELETE ON GRAPH neo4j ELEMENTS * TO regularUsers
```

The `DELETE` privilege can also be denied:

```
DENY [IMMUTABLE] DELETE
  ON { HOME GRAPH | GRAPH[S] { * | name[, ...] } }
    [
      ELEMENT[S] { * | label-or-rel-type[, ...] }
      | NODE[S] { * | label[, ...] }
      | RELATIONSHIP[S] { * | rel-type[, ...] }
    ]
  TO role[, ...]
```

For example, to deny the role `regularUsers` the ability to `DELETE` relationships with the relationship type `bar` on all graphs, use:

```
DENY DELETE ON GRAPH * RELATIONSHIPS bar TO regularUsers
```

> **Note:**
>
> Users with `DELETE` privilege, but restricted `TRAVERSE` privileges, will not be able to do `DETACH DELETE` in all cases. See delete restricted user for more info.

> **Note:**
>
> If a label or a relationship type does not exist in the database, the user cannot use the corresponding privilege until it is created. See Privileges for nonexistent labels, relationship types, and property names for more information.

The `SET LABEL` privilege

The `SET LABEL` privilege allows you to set labels on a node using the Cypher `SET` clause:

```
GRANT [IMMUTABLE] SET LABEL { * | label[, ...] }
  ON { HOME GRAPH | GRAPH[S] { * | name[, ...] } }
  TO role[, ...]
```

For example, to grant the role `regularUsers` the ability to `SET` any label on nodes of the graph `neo4j`, use:

```
GRANT SET LABEL * ON GRAPH neo4j TO regularUsers
```

> **Note:**
>
> Unlike many of the other `READ` and `WRITE` privileges, it is not possible to restrict the `SET LABEL` privilege to specific ELEMENTS, NODES or RELATIONSHIPS.

The `SET LABEL` privilege can also be denied:

```
DENY [IMMUTABLE] SET LABEL { * | label[, ...] }
  ON { HOME GRAPH | GRAPH[S] { * | name[, ...] } }
  TO role[, ...]
```

For example, to deny the role `regularUsers` the ability to `SET` the label `foo` on nodes of all graphs, use:

```
DENY SET LABEL foo ON GRAPH * TO regularUsers
```

> **Note:**
>
> If no instances of this label exist on the database, then the `CREATE NEW LABEL` privilege is also required.

> **Note:**
>
> If a label does not exist in the database, the user cannot use the corresponding privilege until it is created. See Privileges for nonexistent labels, relationship types, and property names for more information.

The `REMOVE LABEL` privilege

The `REMOVE LABEL` privilege allows you to remove labels from a node by using the Cypher `REMOVE` clause:

```
GRANT [IMMUTABLE] REMOVE LABEL { * | label[, ...] }
  ON { HOME GRAPH | GRAPH[S] { * | name[, ...] } }
  TO role[, ...]
```

For example, to grant the role `regularUsers` the ability to `REMOVE` any label from nodes of the graph `neo4j`, use:

```
GRANT REMOVE LABEL * ON GRAPH neo4j TO regularUsers
```

> **Note:**
>
> Unlike many of the other `READ` and `WRITE` privileges, it is not possible to restrict the `REMOVE`

`LABEL` privilege to specific ELEMENTS, NODES or RELATIONSHIPS.

The `REMOVE LABEL` privilege can also be denied:

```
DENY [IMMUTABLE] REMOVE LABEL { * | label[, ...] }
  ON { HOME GRAPH | GRAPH[S] { * | name[, ...] } }
  TO role[, ...]
```

For example, denying the role `regularUsers` the ability to remove the label `foo` from nodes of all graphs, use:

```
DENY REMOVE LABEL foo ON GRAPH * TO regularUsers
```

> **Note:**
>
> If a label does not exist in the database, the user cannot use the corresponding privilege until it is created. See Privileges for nonexistent labels, relationship types, and property names for more information.

The `SET PROPERTY` privilege

The `SET PROPERTY` privilege allows a user to set a property on a node or relationship element in a graph by using the Cypher `SET` clause:

```
GRANT [IMMUTABLE] SET PROPERTY "{" { * | property[, ...] } "}"
  ON { HOME GRAPH | GRAPH[S] { * | name[, ...] } }
    [
      ELEMENT[S] { * | label-or-rel-type[, ...] }
      | NODE[S] { * | label[, ...] }
      | RELATIONSHIP[S] { * | rel-type[, ...] }
    ]
  TO role[, ...]
```

For example, to grant the role `regularUsers` the ability to `SET` any property on all elements of the graph `neo4j`, use:

```
GRANT SET PROPERTY {*} ON HOME GRAPH ELEMENTS * TO regularUsers
```

The `SET PROPERTY` privilege can also be denied:

```
DENY [IMMUTABLE] SET PROPERTY "{" { * | property[, ...] } "}"
  ON { HOME GRAPH | GRAPH[S] { * | name[, ...] } }
    [
      ELEMENT[S] { * | label-or-rel-type[, ...] }
      | NODE[S] { * | label[, ...] }
      | RELATIONSHIP[S] { * | rel-type[, ...] }
    ]
  TO role[, ...]
```

For example, to deny the role `regularUsers` the ability to `SET` the property `foo` on nodes with the label `bar` on all graphs, use:

```
DENY SET PROPERTY { foo } ON GRAPH * NODES bar TO regularUsers
```

> **Note:**
>
> If the user attempts to set a property with a property name that does not already exist on the database, the user must also possess the `CREATE NEW PROPERTY NAME` privilege.

> **Note:**
>
> If a label, a relationship type, or a property name does not exist in the database, the user cannot use the corresponding privilege until it is created. See Privileges for nonexistent labels, relationship types, and property names for more information.

The `MERGE` privilege

The `MERGE` privilege is a compound privilege that combines `TRAVERSE` and `READ` (i.e. `MATCH`) with `CREATE` and `SET PROPERTY`. This is intended to enable the use of the Cypher `MERGE` command, but it is also applicable to all reads and writes that require these privileges.

```
GRANT [IMMUTABLE] MERGE "{" { * | property[, ...] } "}"
  ON { HOME GRAPH | GRAPH[S] { * | name[, ...] } }
    [
      ELEMENT[S] { * | label-or-rel-type[, ...] }
      | NODE[S] { * | label[, ...] }
      | RELATIONSHIP[S] { * | rel-type[, ...] }
    ]
  TO role[, ...]
```

For example, to grant the role `regularUsers` the ability to `MERGE` on all elements of the graph `neo4j`, use:

```
GRANT MERGE {*} ON GRAPH neo4j ELEMENTS * TO regularUsers
```

It is not possible to deny the `MERGE` privilege. If you wish to prevent a user from creating elements and setting properties: use `DENY CREATE` or `DENY SET PROPERTY`.

> **Note:**
>
> If the user attempts to create nodes with a label that does not already exist on the database, the user must also possess the `CREATE NEW LABEL` privilege. The same applies to new relationships and properties - the `CREATE NEW RELATIONSHIP TYPE` or `CREATE NEW PROPERTY NAME` privileges are required.

> **Note:**
>
> If a label, a relationship type, or a property name does not exist in the database, the user cannot use the corresponding privilege until it is created. See Privileges for nonexistent labels, relationship types, and property names for more information.

The `WRITE` privilege

The `WRITE` privilege allows the user to execute any `WRITE` command on a graph.

```
GRANT [IMMUTABLE] WRITE
  ON { HOME GRAPH | GRAPH[S] { * | name[, ...] } }
  TO role[, ...]
```

For example, to grant the role `regularUsers` the ability to `WRITE` on the graph `neo4j`, use:

```
GRANT WRITE ON GRAPH neo4j TO regularUsers
```

Note:

Unlike the more specific `WRITE` commands, it is not possible to restrict `WRITE` privileges to specific ELEMENTS, NODES or RELATIONSHIPS. If you wish to prevent a user from writing to a subset of database objects, a `GRANT WRITE` can be combined with more specific `DENY` commands to target these elements.

The `WRITE` privilege can also be denied:

```
DENY [IMMUTABLE] WRITE
  ON { HOME GRAPH | GRAPH[S] { * | name[, ...] } }
  TO role[, ...]
```

For example, to deny the role `regularUsers` the ability to `WRITE` on the graph `neo4j`, use:

```
DENY WRITE ON GRAPH neo4j TO regularUsers
```

Note:

Users with `WRITE` privilege but restricted `TRAVERSE` privileges will not be able to do `DETACH DELETE` in all cases. See delete restricted user for more info.

The `ALL GRAPH PRIVILEGES` privilege

The `ALL GRAPH PRIVILEGES` privilege allows the user to execute any command on a graph:

```
GRANT [IMMUTABLE] ALL [ [ GRAPH ] PRIVILEGES ]
  ON { HOME GRAPH | GRAPH[S] { * | name[, ...] } }
  TO role[, ...]
```

For example, to grant the role `regularUsers` `ALL GRAPH PRIVILEGES` on the graph `neo4j`, use:

```
GRANT ALL GRAPH PRIVILEGES ON GRAPH neo4j TO regularUsers
```

Note:

Unlike the more specific `READ` and `WRITE` commands, it is not possible to restrict `ALL GRAPH PRIVILEGES` to specific ELEMENTS, +NODES or RELATIONSHIPS. If you wish to prevent a user from reading or writing to a subset of database objects, a `GRANT ALL GRAPH PRIVILEGES` can be combined with more specific `DENY` commands to target these elements.

Note:

The `ALL GRAPH PRIVILEGES` privilege does not allow creating new labels, relationship types, or property names. These are instead managed by the `NAME MANAGEMENT` privileges.

The `ALL GRAPH PRIVILEGES` privilege can also be denied:

```
DENY [IMMUTABLE] ALL [ [ GRAPH ] PRIVILEGES ]
  ON { HOME GRAPH | GRAPH[S] { * | name[, ...] } }
  TO role[, ...]
```

For example, to deny the role `regularUsers` all graph privileges on the graph `neo4j`, use:

```
DENY ALL GRAPH PRIVILEGES ON GRAPH neo4j TO regularUsers
```

Database privileges

Administrators can use the following Cypher commands to manage Neo4j database administrative rights.

The components of the database privilege commands are:

- *command*:
  - `GRANT` – gives privileges to roles.
  - `DENY` – denies privileges to roles.
  - `REVOKE` – removes granted or denied privileges from roles.
- *mutability*:
  - `IMMUTABLE` - When used in conjunction with `GRANT` or `DENY`, specifies that a privilege cannot subsequently be removed unless auth is disabled. Contrastingly, when `IMMUTABLE` is specified in conjunction with a `REVOKE` command, it will act as a filter and only remove matching *immutable* privileges. See also Immutable roles and privileges.
- *database-privilege*
  - `ACCESS` - allows access to a specific database or remote database alias.
  - `START` - allows the specified database to be started.
  - `STOP` - allows the specified database to be stopped.
  - `CREATE INDEX` - allows indexes to be created on the specified database.
  - `DROP INDEX` - allows indexes to be deleted on the specified database.
  - `SHOW INDEX` - allows indexes to be listed on the specified database.

- `INDEX [MANAGEMENT]` - allows indexes to be created, deleted, and listed on the specified database.
- `CREATE CONSTRAINT` - allows constraints to be created on the specified database.
- `DROP CONSTRAINT` - allows constraints to be deleted on the specified database.
- `SHOW CONSTRAINT` - allows constraints to be listed on the specified database.
- `CONSTRAINT [MANAGEMENT]` - allows constraints to be created, deleted, and listed on the specified database.
- `CREATE NEW [NODE] LABEL` - allows new node labels to be created.
- `CREATE NEW [RELATIONSHIP] TYPE` - allows new relationship types to be created.
- `CREATE NEW [PROPERTY] NAME` - allows property names to be created, so that nodes and relationships can have properties assigned with these names.
- `NAME [MANAGEMENT]` - allows all of the name management capabilities: node labels, relationship types, and property names.
- `ALL [[DATABASE] PRIVILEGES]` - allows access, index, constraint, and name management for the specified database or remote database alias.
- `SHOW TRANSACTION` - allows listing transactions and queries for the specified users on the specified database.
- `TERMINATE TRANSACTION` - allows ending transactions and queries for the specified users on the specified database.
- `TRANSACTION [MANAGEMENT]` - allows listing and ending transactions and queries for the specified users on the specified database.

- *name*
  - The database to associate the privilege with.

    Note:

    If you delete a database and create a new one with the same name, the new one will NOT have the same privileges previously assigned to the deleted one.

  - The *name* component can be `*`, which means all databases. Databases created after this command execution will also be associated with these privileges.
  - The `DATABASE[S] name` part of the command can be replaced by `HOME DATABASE`. This refers to the home database configured for a user or, if that user does not have a home database configured, the default database. If the user's home database changes for any reason after this command execution, the new one will be associated with these privileges. This can be quite powerful as it allows permissions to be switched from one database to another simply by changing a user's home database.
- *role[, ...]*
  - The role or roles to associate the privilege with, comma-separated.

Note:

For more details about the syntax descriptions, see Database management command syntax.

Table 470. General grant ON DATABASE privilege syntax

Command	`GRANT ... ON ... TO ...`
Syntax	`GRANT [IMMUTABLE] database-privilege ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } } TO role[, ...]`
Description	Grants a privilege to one or multiple roles.

Table 471. General deny ON DATABASE privilege syntax

Command	`DENY ... ON ... TO ...`
Syntax	`DENY [IMMUTABLE] database-privilege ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } } TO role[, ...]`
Description	Denies a privilege to one or multiple roles.

Table 472. General revoke ON DATABASE privilege syntax

Command	`REVOKE GRANT ... ON ... FROM ...`
Syntax	`REVOKE [IMMUTABLE] GRANT database-privilege ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } } FROM role[, ...]`
Description	Revoke a granted privilege from one or multiple roles.

Table 473. General revoke ON DATABASE privilege syntax

Command	`REVOKE DENY ... ON ... FROM ...`
Syntax	`REVOKE [IMMUTABLE] DENY database-privilege ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } } FROM role[, ...]`
Description	Revokes a denied privilege from one or multiple roles.

Table 474. General revoke ON DATABASE privilege syntax

Command	`REVOKE ... ON ... FROM ...`
Syntax	`REVOKE [IMMUTABLE] database-privilege ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } } FROM role[, ...]`
Description	Revokes a granted or denied privilege from one or multiple roles.

Note:

`DENY` does **not** erase a granted privilege. Use `REVOKE` if you want to remove a privilege.

Common errors, such as misspellings or attempts to revoke privileges that have not been granted or denied, will lead to notifications. Some of these notifications may be replaced with errors in a future major version of Neo4j. See Status Codes for Errors & Notifications → Server notifications for details on notifications.

The hierarchy between the different database privileges is shown in the image below.

Figure 31. Database privileges hierarchy

Table 475. Database privilege syntax

Command	`GRANT ACCESS`

Syntax	`GRANT [IMMUTABLE] ACCESS`
	`  ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } }`
	`  TO role[, ...]`

Description	Grants the specified roles the privilege to access: • The home database. • Specific database(s) or remote database alias(es). • All databases and remote database aliases.

Table 476. Database privilege syntax

Command	`GRANT { START | STOP }`

Syntax	`GRANT [IMMUTABLE] { START | STOP }`
	`  ON { HOME DATABASE | DATABASE[S] {* | name[, ...] } }`
	`  TO role[, ...]`

Description	Grants the specified roles the privilege to start or stop the home database, specific database(s), or all databases.

Table 477. Database privilege syntax

Command	`GRANT { CREATE | DROP | SHOW } INDEX`

Syntax	`GRANT [IMMUTABLE] { CREATE | DROP | SHOW } INDEX[ES]`
	`  ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } }`
	`  TO role[, ...]`

Description	Grants the specified roles the privilege to create, delete, or show indexes on the home database, specific database(s), or all databases.

Table 478. Database privilege syntax

Command	GRANT INDEX

Syntax	`GRANT [IMMUTABLE] INDEX[ES] [MANAGEMENT]`
	`  ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } }`
	`  TO role[, ...]`

Description	Grants the specified roles the privilege to manage indexes on the home database, specific database(s), or all databases.

Table 479. Database privilege syntax

Command	GRANT { CREATE | DROP | SHOW } CONSTRAINT

Syntax	`GRANT [IMMUTABLE] { CREATE | DROP | SHOW } CONSTRAINT[S]`
	`  ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } }`
	`  TO role[, ...]`

Description	Grants the specified roles the privilege to create, delete, or show constraints on the home database, specific database(s), or all databases.

Table 480. Database privilege syntax

Command	GRANT CONSTRAINT

Syntax	`GRANT [IMMUTABLE] CONSTRAINT[S] [MANAGEMENT]`
	`  ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } }`
	`  TO role[, ...]`

Description	Grants the specified roles the privilege to manage constraints on the home database, specific database(s), or all databases.

Table 481. Database privilege syntax

Command	GRANT CREATE NEW LABEL

Syntax	`GRANT [IMMUTABLE] CREATE NEW [NODE] LABEL[S]`
	`  ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } }`
	`  TO role[, ...]`

Description	Grants the specified roles the privilege to create new node labels in the home database, specific database(s), or all databases.

Table 482. Database privilege syntax

Command	GRANT CREATE NEW TYPE

Syntax	`GRANT [IMMUTABLE] CREATE NEW [RELATIONSHIP] TYPE[S]`
	`  ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } }`
	`  TO role[, ...]`

Description	Grants the specified roles the privilege to create new relationship types in the home database, specific database(s), or all databases.

Table 483. Database privilege syntax

Command	`GRANT CREATE NEW NAME`

Syntax	`GRANT [IMMUTABLE] CREATE NEW [PROPERTY] NAME[S]`
	`  ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } }`
	`  TO role[, ...]`

Description	Grants the specified roles the privilege to create new property names in the home database, specific database(s), or all databases.

Table 484. Database privilege syntax

Command	`GRANT NAME`

Syntax	`GRANT [IMMUTABLE] NAME [MANAGEMENT]`
	`  ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } }`
	`  TO role[, ...]`

Description	Grants the specified roles the privilege to manage new labels, relationship types, and property names in the home database, specific database(s), or all databases.

Table 485. Database privilege syntax

Command	`GRANT ALL`

Syntax	`GRANT [IMMUTABLE] ALL [[DATABASE] PRIVILEGES]`
	`  ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } }`
	`  TO role[, ...]`

Description	Grants the specified roles all privileges for the home, a specific, or all databases and remote database aliases.

Table 486. Database privilege syntax

Command	`GRANT { SHOW | TERMINATE } TRANSACTION`

Syntax	`GRANT [IMMUTABLE] { SHOW | TERMINATE } TRANSACTION[S] [( { * | user[, ...] } )]`
	`  ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } }`
	`  TO role[, ...]`

Description	Grants the specified roles the privilege to list and end the transactions and queries of all users or a particular user(s) in the home database, specific database(s), or all databases.

Table 487. Database privilege syntax

Command	`GRANT TRANSACTION`

Syntax	`GRANT [IMMUTABLE] TRANSACTION [MANAGEMENT] [( { * | user[, ...] } )]`
	`  ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } }`
	`  TO role[, ...]`

Description	Grants the specified roles the privilege to manage the transactions and queries of all users or a particular user(s) in the home database, specific database(s), or all databases.

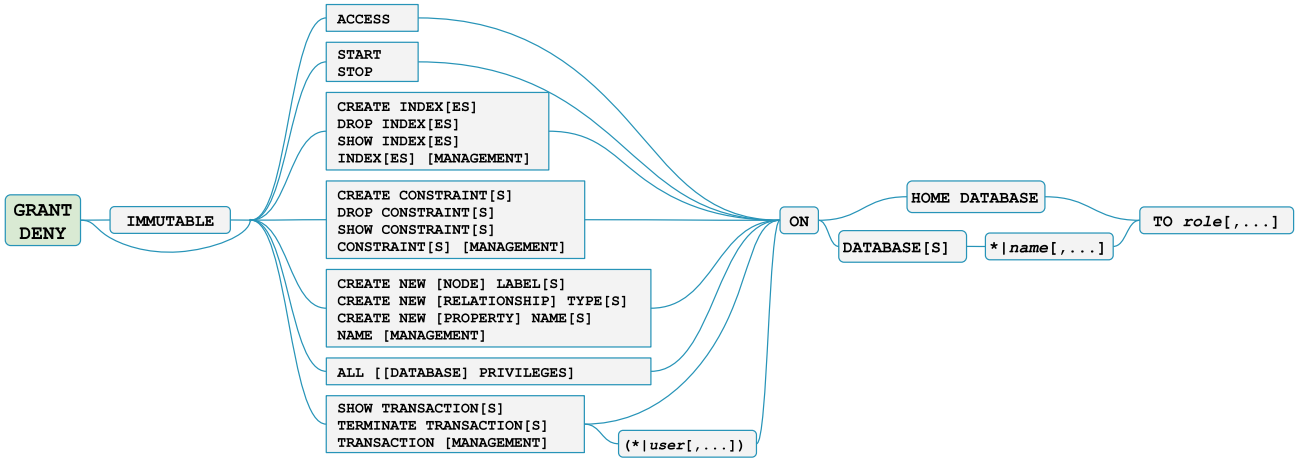


Figure 32. Syntax of GRANT and DENY Database Privileges

The database `ACCESS` privilege

The `ACCESS` privilege enables users to connect to a database or a remote database alias. With `ACCESS` you can run calculations, for example, `RETURN 2 * 5 AS answer` or call functions `RETURN timestamp() AS time`.

```
GRANT [IMMUTABLE] ACCESS
  ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } }
  TO role[, ...]
```

For example, to grant the role `regularUsers` the ability to access the database `neo4j`, use:

```
GRANT ACCESS ON DATABASE neo4j TO regularUsers
```

The `ACCESS` privilege can also be denied:

```
DENY [IMMUTABLE] ACCESS
  ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } }
  TO role[, ...]
```

For example, to deny the role `regularUsers` the ability to access to the remote database alias `remote-db`, use:

```
DENY ACCESS ON DATABASE `remote-db` TO regularUsers
```

The privileges granted can be seen using the `SHOW PRIVILEGES` command:

```
SHOW ROLE regularUsers PRIVILEGES AS COMMANDS
```

Table 488. Result

command
"DENY ACCESS ON DATABASE `remote-db` TO `regularUsers`"
"GRANT ACCESS ON DATABASE `neo4j` TO `regularUsers`"
Rows: 2

The database `START`/`STOP` privileges

The `START` privilege can be used to enable the ability to start a database:

```
GRANT [IMMUTABLE] START
  ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } }
  TO role[, ...]
```

For example, to grant the role `regularUsers` the ability to start the database `neo4j`, use:

```
GRANT START ON DATABASE neo4j TO regularUsers
```

The `START` privilege can also be denied:

```
DENY [IMMUTABLE] START
  ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } }
  TO role[, ...]
```

For example, to deny the role `regularUsers` the ability to start to the database `neo4j`, use:

```
DENY START ON DATABASE system TO regularUsers
```

The `STOP` privilege can be used to enable the ability to stop a database:

```
GRANT [IMMUTABLE] STOP
  ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } }
  TO role[, ...]
```

For example, to grant the role `regularUsers` the ability to stop the database `neo4j`, use:

```
GRANT STOP ON DATABASE neo4j TO regularUsers
```

The `STOP` privilege can also be denied:

```
DENY [IMMUTABLE] STOP
  ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } }
  TO role[, ...]
```

For example, to deny the role `regularUsers` the ability to stop the database `neo4j`, use:

```
DENY STOP ON DATABASE system TO regularUsers
```

The privileges granted can be seen using the `SHOW PRIVILEGES` command:

```
SHOW ROLE regularUsers PRIVILEGES AS COMMANDS
```

Table 489. Result

command
"DENY ACCESS ON DATABASE `remote-db` TO `regularUsers`"
"DENY START ON DATABASE `system` TO `regularUsers`"
"DENY STOP ON DATABASE `system` TO `regularUsers`"
"GRANT ACCESS ON DATABASE `neo4j` TO `regularUsers`"
"GRANT START ON DATABASE `neo4j` TO `regularUsers`"
"GRANT STOP ON DATABASE `neo4j` TO `regularUsers`"
Rows: 6

Note:

Note that `START` and `STOP` privileges are not included in the `ALL DATABASE PRIVILEGES`.

The `INDEX MANAGEMENT` privileges

Indexes can be created, deleted, or listed with the `CREATE INDEX`, `DROP INDEX`, and `SHOW INDEXES` commands. The privilege to do this can be granted with `GRANT CREATE INDEX`, `GRANT DROP INDEX`, and `GRANT SHOW INDEX` commands. The privilege to do all three can be granted with `GRANT INDEX MANAGEMENT` command.

Table 490. Index management privilege syntax

Command	`GRANT { CREATE | DROP | SHOW } INDEX`

Syntax	`GRANT [IMMUTABLE] { CREATE | DROP | SHOW } INDEX[ES]`
	`  ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } }`
	`  TO role[, ...]`

Description	Enables the specified roles to create, delete, or show indexes in the home database, specific database(s), or all databases.

Table 491. Index management privilege syntax

Command	`GRANT INDEX`

Syntax	`GRANT [IMMUTABLE] INDEX[ES] [MANAGEMENT]`
	`  ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } }`
	`  TO role[, ...]`

Description	Enables the specified roles to manage indexes in the home database, specific database(s), or all databases.

For example, to grant the role `regularUsers` the ability to create indexes on the database `neo4j`, use:

```
GRANT CREATE INDEX ON DATABASE neo4j TO regularUsers
```

The `CONSTRAINT MANAGEMENT` privileges

Constraints can be created, deleted, or listed with the `CREATE CONSTRAINT`, `DROP CONSTRAINT` and `SHOW CONSTRAINTS` commands. The privilege to do this can be granted with `GRANT CREATE CONSTRAINT`, `GRANT DROP CONSTRAINT`, `GRANT SHOW CONSTRAINT` commands. The privilege to do all three can be granted with `GRANT CONSTRAINT MANAGEMENT` command.

Table 492. Constraint management privilege syntax

Command	`GRANT { CREATE | DROP | SHOW } CONSTRAINT`
Syntax	`GRANT [IMMUTABLE] { CREATE | DROP | SHOW } CONSTRAINT[S] ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } } TO role[, ...]`
Description	Enables the specified roles to create, delete, or show constraints on the home database, specific database(s), or all databases.

Table 493. Constraint management privilege syntax

Command	`GRANT CONSTRAINT`
Syntax	`GRANT [IMMUTABLE] CONSTRAINT[S] [MANAGEMENT] ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } } TO role[, ...]`
Description	Enable the specified roles to manage constraints on the home database, specific database(s), or all databases.

For example, to grant the role `regularUsers` the ability to create constraints on the database `neo4j`, use:

```
GRANT CREATE CONSTRAINT ON DATABASE neo4j TO regularUsers
```

The `NAME MANAGEMENT` privileges

The right to create new labels, relationship types, and property names is different from the right to create nodes, relationships, and properties. The latter is managed using database `WRITE` privileges, while the former is managed using specific `GRANT/DENY CREATE NEW ...` commands for each type.

Table 494. Node label management privileges syntax

Command	`GRANT CREATE NEW LABEL`
Syntax	`GRANT [IMMUTABLE] CREATE NEW [NODE] LABEL[S] ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } } TO role[, ...]`

Description	Enables the specified roles to create new node labels in the home database, specific database(s), or all databases.

Table 495. Relationship type management privileges syntax

Command	`GRANT CREATE NEW TYPE`
Syntax	`GRANT [IMMUTABLE] CREATE NEW [RELATIONSHIP] TYPE[S] ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } } TO role[, ...]`
Description	Enables the specified roles to create new relationship types in the home database, specific database(s), or all databases.

Table 496. Property name management privileges syntax

Command	`GRANT CREATE NEW NAME`
Syntax	`GRANT [IMMUTABLE] CREATE NEW [PROPERTY] NAME[S] ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } } TO role[, ...]`
Description	Enables the specified roles to create new property names in the home database, specific database(s), or all databases.

Table 497. Node label, relationship type, and property name privileges management syntax

Command	`GRANT NAME`
Syntax	`GRANT [IMMUTABLE] NAME [MANAGEMENT] ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } } TO role[, ...]`
Description	Enables the specified roles to create new labels, relationship types, and property names in the home database, specific database(s), or all databases.

Note:

The `SHOW PRIVILEGES` commands return the `NAME MANAGEMENT` privilege as the action `token`, when not using `AS COMMANDS`.

For example, to grant the role `regularUsers` the ability to create new properties on nodes or relationships on the database `neo4j`, use:

```
GRANT CREATE NEW PROPERTY NAME ON DATABASE neo4j TO regularUsers
```

Granting `ALL DATABASE PRIVILEGES`

The right to access a database, create and drop indexes and constraints and create new labels, relationship types or property names can be achieved with a single command:

```
GRANT [IMMUTABLE] ALL [[DATABASE] PRIVILEGES]
  ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } }
  TO role[, ...]
```

> **Note:**
>
> Note that the privileges for starting and stopping all databases, and transaction management, are not included in the `ALL DATABASE PRIVILEGES` grant. These privileges are associated with administrators while other database privileges are of use to domain and application developers.

For example, granting the abilities above on the database `neo4j` to the role `databaseAdminUsers` is done using the following query.

```
GRANT ALL DATABASE PRIVILEGES ON DATABASE neo4j TO databaseAdminUsers
```

The privileges granted can be seen using the `SHOW PRIVILEGES` command:

```
SHOW ROLE databaseAdminUsers PRIVILEGES AS COMMANDS
```

Table 498. Result

command
"GRANT ALL DATABASE PRIVILEGES ON DATABASE `neo4j` TO `databaseAdminUsers`"
Rows: 1

Granting `TRANSACTION MANAGEMENT` privileges

The right to run the commands `SHOW TRANSACTIONS`, `TERMINATE TRANSACTIONS`, and the deprecated procedures `dbms.listTransactions`, `dbms.listQueries`, `dbms.killQuery`, `dbms.killQueries`, `dbms.killTransaction` and `dbms.killTransactions` is now managed through the `SHOW TRANSACTION` and `TERMINATE TRANSACTION` privileges.

Table 499. Database privilege syntax

Command	`GRANT SHOW TRANSACTION`

Syntax	`GRANT [IMMUTABLE] SHOW TRANSACTION[S] [( { * | user[, ...] } )]`
	`  ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } }`
	`  TO role[, ...]`

Description	Enables the specified roles to list transactions and queries for user(s) or all users in the home database, specific database(s), or all databases.

Table 500. Database privilege syntax

Command	`GRANT TERMINATE TRANSACTION`

Syntax	`GRANT [IMMUTABLE] TERMINATE TRANSACTION[S] [( { * | user[, ...] } )]`
	`  ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } }`
	`  TO role[, ...]`

Description	Enables the specified roles to end running transactions and queries for user(s) or all users in the home database, specific database(s), or all databases.

Table 501. Database privilege syntax

Command	`GRANT TRANSACTION`

Syntax	`GRANT [IMMUTABLE] TRANSACTION [MANAGEMENT] [( { * | user[, ...] } )]`
	`  ON { HOME DATABASE | DATABASE[S] { * | name[, ...] } }`
	`  TO role[, ...]`

Description	Enables the specified roles to manage transactions and queries for user(s) or all users in the home database, specific database(s), or all databases.

Note:

Note that the `TRANSACTION MANAGEMENT` privileges are not included in the `ALL DATABASE PRIVILEGES`.

For example, to grant the role `regularUsers` the ability to list transactions for user `jake` on the database `neo4j`, use:

```
GRANT SHOW TRANSACTION (jake) ON DATABASE neo4j TO regularUsers
```

DBMS privileges

All DBMS privileges are relevant system-wide. Like user management, they do not belong to one specific database or graph. For more details on the differences between graphs, databases, and the DBMS, refer to Cypher Manual → Cypher and Neo4j.

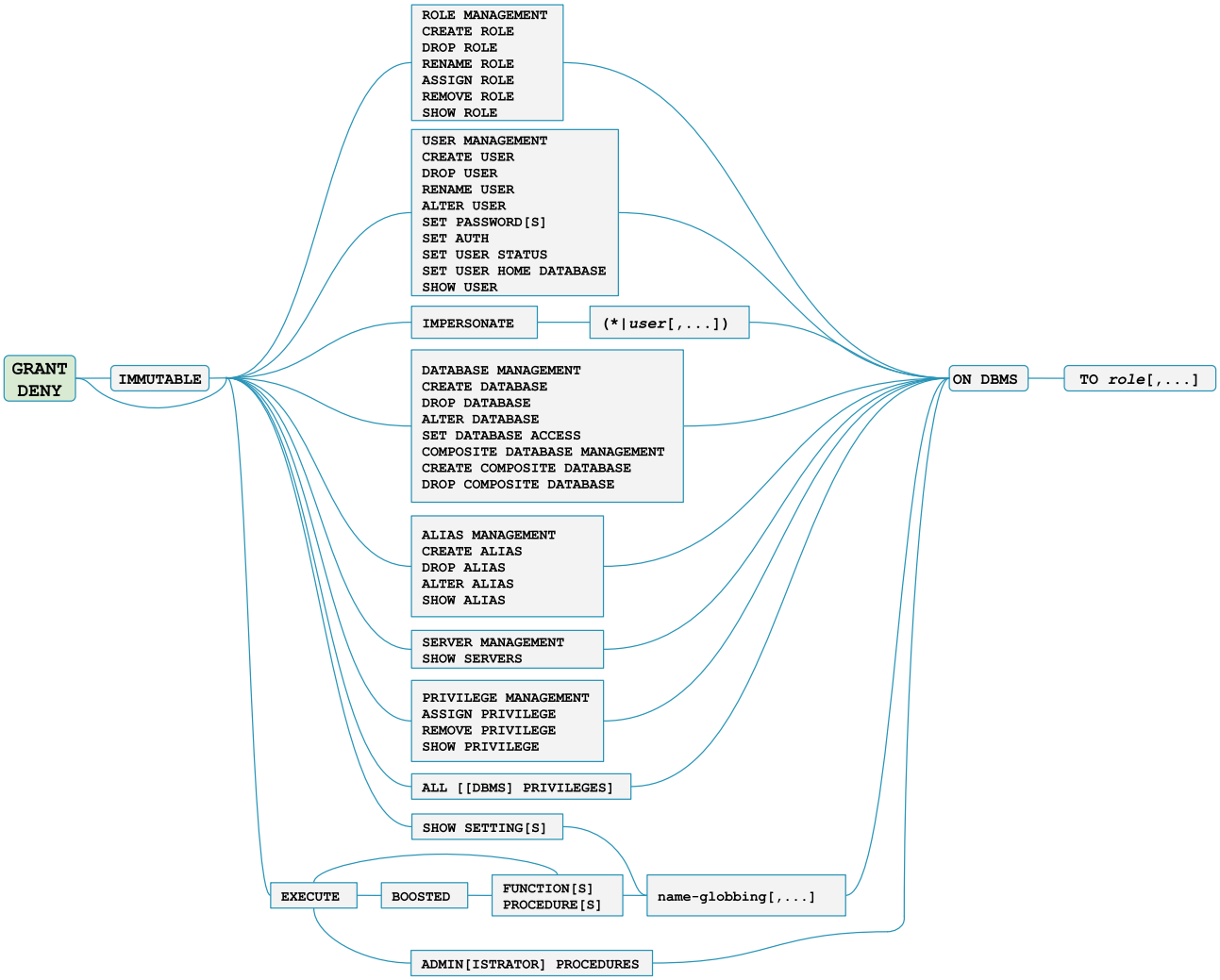


Figure 33. Syntax of `GRANT` and `DENY` DBMS privileges

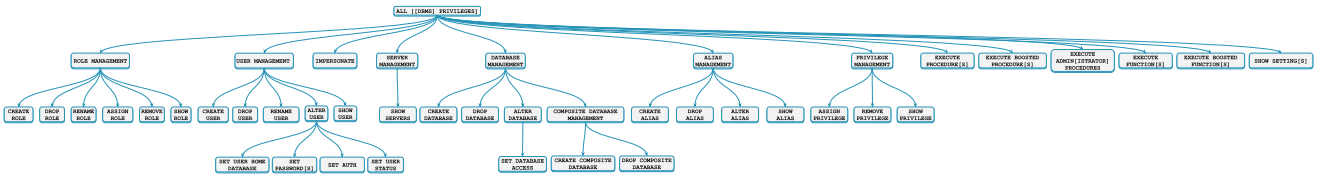


Figure 34. DBMS privileges hierarchy

> **Note:**
>
> You can only grant, deny, or revoke privileges to or from existing roles.

Granting `ALL DBMS PRIVILEGES`

You can grant the `ALL DBMS PRIVILEGES` privilege to a role.
The `ALL DBMS PRIVILEGES` privilege is equivalent to granting the following privileges:

- Create, drop, assign, remove, and show roles.
- Create, alter, drop, show, and impersonate users.
- Create, alter, drop, and show databases and aliases.

- Enable, alter, rename, reallocate, deallocate, drop, and show servers.
- Show, assign, and remove privileges.
- Execute all procedures with elevated privileges.
- Execute all user-defined functions with elevated privileges.
- Show all configuration settings.

Note:

For more details about the syntax descriptions, see Reading the administration commands syntax.

```
GRANT [IMMUTABLE] ALL [[DBMS] PRIVILEGES]
    ON DBMS
    TO role[, ...]
```

For example, to grant the role `dbmsManager` all DBMS abilities, use the following query:

```
GRANT ALL DBMS PRIVILEGES ON DBMS TO dbmsManager;
```

To list all privileges for the role `dbmsManager` as commands, use the following query:

```
SHOW ROLE dbmsManager PRIVILEGES AS COMMANDS;
```

Table 502. Result

command
"GRANT ALL DBMS PRIVILEGES ON DBMS TO `dbmsManager`"
Rows: 1

Name-globbing for procedures, user-defined functions, and settings

The name-globbing for procedures, user-defined functions, and setting names is a simplified version of globbing for filename expansions. It only allows two wildcard characters: `*` and `?`, which are used for multiple and single-character matches. In this case, `*` means 0 or more characters, and `?` matches exactly one character.

Note:

The name-globbing is subject to the Cypher Manual → Naming rules and recommendations, with the exception that it may include dots, stars, and question marks without the need for escaping using backticks.

Each part of the name-globbing separated by dots may be individually quoted. For example, `mine.\`procedureWith%\`` is allowed, but not `mine.procedure\`With%\``. Also, note that wildcard characters behave as wildcards even when quoted. For example, using `` `*` `` is equivalent to

using `*`, and thus allows executing all functions or procedures and not only the procedure or function named `*`.

Given the following list of procedures:

- `mine.public.exampleProcedure`
- `mine.public.exampleProcedure1`
- `mine.public.exampleProcedure2`
- `mine.public.with#Special§Characters`
- `mine.private.exampleProcedure`
- `mine.private.exampleProcedure1`
- `mine.private.exampleProcedure2`
- `mine.private.with#Special§Characters`
- `your.exampleProcedure`

The following examples demonstrate how name-globbing patterns can be used in controlling access to procedures. Note that the same rules apply to user-defined functions and settings.

```
GRANT EXECUTE PROCEDURE * ON DBMS TO globbing1;
```

Users with the role `globbing1` can run all the procedures.

```
GRANT EXECUTE PROCEDURE mine.*.exampleProcedure ON DBMS TO globbing2;
```

Users with the role `globbing2` can run procedures `mine.public.exampleProcedure` and `mine.private.exampleProcedure`, but no other procedures.

```
GRANT EXECUTE PROCEDURE mine.*.exampleProcedure? ON DBMS TO globbing3;
```

Users with the role `globbing3` can run procedures `mine.public.exampleProcedure1`, `mine.private.exampleProcedure1`, and `mine.private.exampleProcedure2`, but no other procedures.

```
GRANT EXECUTE PROCEDURE *.exampleProcedure ON DBMS TO globbing4;
```

Users with the role `globbing4` can run procedures `your.exampleProcedure`, `mine.public.exampleProcedure`, and `mine.private.exampleProcedure`, but no other procedures.

```
GRANT EXECUTE PROCEDURE mine.public.exampleProcedure* ON DBMS TO globbing5;
```

Users with the role `globbing5` can run procedures `mine.public.exampleProcedure`, `mine.public.exampleProcedure1` and `mine.public.exampleProcedure42`, but no other procedures.

```
GRANT EXECUTE PROCEDURE `mine.public.with#*§Characters`, mine.private.`with#Spec???§Characters` ON DBMS TO
```

```
globbing6;
```

Users with the role `globbing6` can run procedures `mine.public.with#Special§Characters`, and `mine.private.with#Special§Characters`, but no other procedures.

> **Note:**
>
> The name-globbing may be fully or partially quoted. Both `*` and `?` are interpreted as wildcards in both cases.

Administrator role privileges

Enterprise Edition | AuraDB Business Critical | AuraDB Virtual Dedicated Cloud

You can manage DBMS privileges using either the built-in `admin` role or by creating a custom role with the specific privileges you need.

Using the built-in `admin` role to manage DBMS privileges

The `admin` role is a built-in superuser role that has all privileges on the DBMS and can perform all administrative tasks.

This includes the following tasks and their relevant privileges:

- Create, delete, and modify databases and aliases.
- Change configuration parameters.
- Manage transactions.
- Manage users and roles.
- Manage privilege management.
- Manage read and write sub-graph privileges.
- Manage impersonation privileges.
- Manage procedure security.
- Manage load data security.

To enable a user to perform these tasks, you can grant them the `admin` role, but it is also possible to make a custom role with a subset of these privileges. All privileges are also assignable using Cypher commands.

Using a custom role to manage DBMS privileges

You can create a custom role to manage DBMS privileges by granting the privileges you want to the role. Alternatively, you can copy the `admin` role and revoke or deny the unwanted privileges. The following examples show how to create a custom role with a subset of the privileges that the `admin` role using both methods.

Create a custom administrator role from scratch

Create an administrator role that can only manage users and roles by creating a new role and granting the `USER MANAGEMENT` and `ROLE MANAGEMENT` privileges.

1. Create the new role:

```
CREATE ROLE userAndRoleAdministrator;
```

2. Grant the privilege to manage users:

```
GRANT USER MANAGEMENT ON DBMS TO userAndRoleAdministrator;
```

3. Grant the privilege to manage roles:

```
GRANT ROLE MANAGEMENT ON DBMS TO userAndRoleAdministrator;
```

   As a result, the `userAndRoleAdministrator` role has privileges that only allow user and role management.

4. To list all privileges for the role `userAndRoleAdministrator` as commands, use the following query:

```
SHOW ROLE userAndRoleAdministrator PRIVILEGES AS COMMANDS;
```

   Table 503. Result

command
"GRANT ROLE MANAGEMENT ON DBMS TO `userAndRoleAdministrator`"
"GRANT USER MANAGEMENT ON DBMS TO `userAndRoleAdministrator`"
Rows: 2

> **Note:**
>
> This role does not allow all DBMS capabilities. For example, the role is missing privileges for managing, creating, and dropping databases, as well as executing `admin` procedures. To create a more powerful administrator, you can grant a different set of privileges.

Create a more powerful custom administrator role from scratch

You can also create a custom administrator role with limited capabilities. This can be done by creating a new role and granting all `DBMS` privileges, then denying the ones you do not want the role to have, and then granting additional privileges you want to include. For example, you can create a custom administrator role `customAdministrator` that has all DBMS privileges except for creating, dropping, and modifying databases, and also has the privilege for managing transactions.

1. Create a new role:

```
CREATE ROLE customAdministrator;
```

2. Grant the privilege for all DBMS capabilities:

```
GRANT ALL DBMS PRIVILEGES ON DBMS TO customAdministrator;
```

3. Explicitly deny the privilege to manage databases:

```
DENY DATABASE MANAGEMENT ON DBMS TO customAdministrator;
```

4. Grant the transaction management privilege:

```
GRANT TRANSACTION MANAGEMENT (*) ON DATABASE * TO customAdministrator;
```

   As a result, the `customAdministrator` role has privileges that include managing transactions and all DBMS privileges except creating, dropping, and modifying databases.

5. To list all privileges for the role `customAdministrator` as commands, use the following query:

```
SHOW ROLE customAdministrator PRIVILEGES AS COMMANDS;
```

Table 504. Result

command
"DENY DATABASE MANAGEMENT ON DBMS TO `customAdministrator`"
"GRANT ALL DBMS PRIVILEGES ON DBMS TO `customAdministrator`"
"GRANT TRANSACTION MANAGEMENT (*) ON DATABASE * TO `customAdministrator`"
Rows: 3

Create a custom administrator role by copying the `admin` role

You can also create a custom administrator role by copying the `admin` role and then revoking or denying the privileges you do not want. For example, you can create a new role called `newAdministrator` that has all the privileges of the `admin` role, and then revoke the ability to read/write/load data, manage constraints, indexes, name, and remove ability to access all databases, except the `system` database.

1. Create a new role by copying the `admin` role:

```
CREATE ROLE newAdministrator AS COPY OF admin;
```

2. Revoke the ability to read/write/load data:

```
REVOKE GRANT MATCH {*} ON GRAPH * NODE * FROM newAdministrator;
REVOKE GRANT MATCH {*} ON GRAPH * RELATIONSHIP * FROM newAdministrator;
REVOKE GRANT WRITE ON GRAPH * FROM newAdministrator;
REVOKE GRANT LOAD ON ALL DATA FROM newAdministrator;
```

3. Revoke the ability to manage index/constraint/name:

```
REVOKE GRANT CONSTRAINT MANAGEMENT ON DATABASE * FROM newAdministrator;
REVOKE GRANT INDEX MANAGEMENT ON DATABASE * FROM newAdministrator;
REVOKE GRANT NAME MANAGEMENT ON DATABASE * FROM newAdministrator;
REVOKE GRANT SHOW CONSTRAINT ON DATABASE * FROM newAdministrator;
REVOKE GRANT SHOW INDEX ON DATABASE * FROM newAdministrator;
```

4. Revoke the ability to access all databases:

```
REVOKE GRANT ACCESS ON DATABASE * FROM newAdministrator;
```

5. Grant the ability to access the `system` database:

```
GRANT ACCESS ON DATABASE system TO newAdministrator;
```

6. To list all privileges for the role `newAdministrator` as commands, use the following query:

```
SHOW ROLE newAdministrator PRIVILEGES AS COMMANDS;
```

Table 505. Result

command
"GRANT ACCESS ON DATABASE `system` TO `newAdministrator`"
"GRANT ALL DBMS PRIVILEGES ON DBMS TO `newAdministrator`"
"GRANT START ON DATABASE * TO `newAdministrator`"
"GRANT STOP ON DATABASE * TO `newAdministrator`"
"GRANT TRANSACTION MANAGEMENT (*) ON DATABASE * TO `newAdministrator`"
Rows: 5

The DBMS `ROLE MANAGEMENT` privileges

Enterprise Edition | AuraDB Business Critical | AuraDB Virtual Dedicated Cloud

The DBMS privileges for role management can be granted, denied, or revoked like other privileges.

> **Note:**
>
> For more details about the syntax descriptions, see Reading the administration commands syntax.

Table 506. Role management privileges command syntax

Command	Description
`GRANT [IMMUTABLE] CREATE ROLE ON DBMS TO role[, ...]`	Enables the specified roles to create new roles.
`GRANT [IMMUTABLE] RENAME ROLE ON DBMS TO role[, ...]`	Enables the specified roles to change the name of roles.
`GRANT [IMMUTABLE] DROP ROLE ON DBMS TO role[, ...]`	Enables the specified roles to delete roles.
`GRANT [IMMUTABLE] ASSIGN ROLE ON DBMS TO role[, ...]`	Enables the specified roles to assign roles to users.
`GRANT [IMMUTABLE] REMOVE ROLE ON DBMS TO role[, ...]`	Enables the specified roles to remove roles from users.
`GRANT [IMMUTABLE] SHOW ROLE ON DBMS TO role[, ...]`	Enables the specified roles to list roles.
`GRANT [IMMUTABLE] ROLE MANAGEMENT ON DBMS TO role[, ...]`	Enables the specified roles to create, delete, assign, remove, and list roles.

Grant privilege to create roles

You can grant the privilege to add roles using the `CREATE ROLE` command.
For example:

```
GRANT CREATE ROLE ON DBMS TO roleAdder;
```

As a result, the `roleAdder` role has privileges that only allow adding roles. To list all privileges for the role `roleAdder` as commands, use the following query:

```
SHOW ROLE roleAdder PRIVILEGES AS COMMANDS;
```

Table 507. Result

command
"GRANT CREATE ROLE ON DBMS TO `roleAdder`"
Rows: 1

Grant privilege to rename roles

You can grant the privilege to rename roles using the `RENAME ROLE` privilege.

For example:

```
GRANT RENAME ROLE ON DBMS TO roleNameModifier;
```

As a result, the `roleNameModifier` role has privileges that only allow renaming roles. To list all privileges for the role `roleNameModifier`, use the following query:

```
SHOW ROLE roleNameModifier PRIVILEGES AS COMMANDS;
```

Table 508. Result

command
"GRANT RENAME ROLE ON DBMS TO `roleNameModifier`"
Rows: 1

Grant privilege to delete roles

You can grant the privilege to delete roles using the `DROP ROLE` privilege.
For example:

```
GRANT DROP ROLE ON DBMS TO roleDropper;
```

As a result, the `roleDropper` role has privileges that only allow deleting roles. To list all privileges for the role `roleDropper`, use the following query:

```
SHOW ROLE roleDropper PRIVILEGES AS COMMANDS;
```

Table 509. Result

command
"GRANT DROP ROLE ON DBMS TO `roleDropper`"
Rows: 1

Grant privilege to assign roles

You can grant the privilege to assign roles to users using the `ASSIGN ROLE` privilege.
For example:

```
GRANT ASSIGN ROLE ON DBMS TO roleAssigner;
```

As a result, the `roleAssigner` role has privileges that only allow assigning/granting roles. To list all privileges for the role `roleAssigner` as commands, use the following query:

```
SHOW ROLE roleAssigner PRIVILEGES AS COMMANDS;
```

Table 510. Result

command
"GRANT ASSIGN ROLE ON DBMS TO `roleAssigner`"
Rows: 1

Grant privilege to remove roles

You can grant the privilege to remove roles from users using the `REMOVE ROLE` privilege.
For example:

```
GRANT REMOVE ROLE ON DBMS TO roleRemover;
```

As a result, the `roleRemover` role has privileges that only allow removing/revoking roles. To list all privileges for the role `roleRemover` as commands, use the following query:

```
SHOW ROLE roleRemover PRIVILEGES AS COMMANDS;
```

Table 511. Result

command
"GRANT REMOVE ROLE ON DBMS TO `roleRemover`"
Rows: 1

Grant privilege to show roles

You can grant the privilege to show roles using the `SHOW ROLE` privilege. A role with this privilege is allowed to execute the `SHOW ROLES` and `SHOW POPULATED ROLES` administration commands.

Note:

In order to use `SHOW ROLES WITH USERS` and `SHOW POPULATED ROLES WITH USERS` administration commands, both the `SHOW ROLE` and the `SHOW USER` privileges are required.

The following query shows an example of how to grant the `SHOW ROLE` privilege:

```
GRANT SHOW ROLE ON DBMS TO roleViewer;
```

As a result, the `roleViewer` role has privileges that only allow showing roles. To list all privileges for the role `roleViewer` as commands, use the following query:

```
SHOW ROLE roleViewer PRIVILEGES AS COMMANDS;
```

Table 512. Result

command
"GRANT SHOW ROLE ON DBMS TO `roleViewer`"
Rows: 1

Grant privilege to manage roles

You can grant the privilege to create, rename, delete, assign, remove, and list roles using the `ROLE MANAGEMENT` privilege.
For example:

```
GRANT ROLE MANAGEMENT ON DBMS TO roleManager;
```

As a result, the `roleManager` role has all privileges to manage roles. To list all privileges for the role `roleManager` as commands, use the following query:

```
SHOW ROLE roleManager PRIVILEGES AS COMMANDS;
```

Table 513. Result

command
"GRANT ROLE MANAGEMENT ON DBMS TO `roleManager`"
Rows: 1

The DBMS `USER MANAGEMENT` privileges

Enterprise Edition | AuraDB Business Critical | AuraDB Virtual Dedicated Cloud

The DBMS privileges for user management can be granted, denied, or revoked like other privileges.

> **Note:**
>
> For more details about the syntax descriptions, see Reading the administration commands syntax.

Table 514. User management privileges command syntax

Command	Description
`GRANT [IMMUTABLE] CREATE USER ON DBMS TO role[, ...]`	Enables the specified roles to create new users.
`GRANT [IMMUTABLE] RENAME USER ON DBMS TO role[, ...]`	Enables the specified roles to change the name of users.

Command	Description
`GRANT [IMMUTABLE] ALTER USER ON DBMS TO role[, ...]`	Enables the specified roles to modify users.
`GRANT [IMMUTABLE] SET PASSWORD[S] ON DBMS TO role[, ...]`	Enables the specified roles to modify users' passwords and whether those passwords must be changed upon first login.
`GRANT [IMMUTABLE] SET AUTH ON DBMS TO role[, ...]`	Enables the specified roles to `SET` or `REMOVE` users' auth providers.
`GRANT [IMMUTABLE] SET USER HOME DATABASE ON DBMS TO role[, ...]`	Enables the specified roles to modify users' home database.
`GRANT [IMMUTABLE] SET USER STATUS ON DBMS TO role[, ...]`	Enables the specified roles to modify the account status of users.
`GRANT [IMMUTABLE] DROP USER ON DBMS TO role[, ...]`	Enables the specified roles to delete users.
`GRANT [IMMUTABLE] SHOW USER ON DBMS TO role[, ...]`	Enables the specified roles to list users.
`GRANT [IMMUTABLE] USER MANAGEMENT ON DBMS TO role[, ...]`	Enables the specified roles to create, delete, modify, and list users.

Grant privilege to create users

You can grant the privilege to add users using the `CREATE USER` privilege.
For example:

```
GRANT CREATE USER ON DBMS TO userAdder
```

As a result, the `userAdder` role has privileges that only allow adding users. To list all privileges for the role `userAdder` as commands, use the following query:

```
SHOW ROLE userAdder PRIVILEGES AS COMMANDS;
```

Table 515. Result

command
"GRANT CREATE USER ON DBMS TO `userAdder`"
Rows: 1

Grant privilege to rename users

You can grant the privilege to rename users using the `RENAME USER` privilege.
For example:

```
GRANT RENAME USER ON DBMS TO userNameModifier
```

As a result, the `userNameModifier` role has privileges that only allow renaming users. To list all privileges for the role `userNameModifier` as commands, use the following query:

```
SHOW ROLE userNameModifier PRIVILEGES AS COMMANDS;
```

Table 516. Result

command
"GRANT RENAME USER ON DBMS TO `userNameModifier`"
Rows: 1

Grant privilege to modify users

You can grant the privilege to modify users using the `ALTER USER` privilege.
For example:

```
GRANT ALTER USER ON DBMS TO userModifier
```

As a result, the `userModifier` role has privileges that only allow modifying users. To list all privileges for the role `userModifier` as commands, use the following query:

```
SHOW ROLE userModifier PRIVILEGES AS COMMANDS;
```

Table 517. Result

command
"GRANT ALTER USER ON DBMS TO `userModifier`"
Rows: 1

The `ALTER USER` privilege allows the user to run the `ALTER USER` administration command with one or several of the `SET PASSWORD`, `SET PASSWORD CHANGE [NOT] REQUIRED`, `SET AUTH`, `REMOVE AUTH`, `SET STATUS`, `SET HOME DATABASE`, and `REMOVE HOME DATABASE` parts.
For example:

```
ALTER USER jake SET PASSWORD 'verysecret' SET STATUS SUSPENDED
```

Note:

Note that the combination of the `SET PASSWORDS`, `SET AUTH`, `SET USER STATUS`, and `SET USER HOME DATABASE` privileges is equivalent to the `ALTER USER` privilege.

Grant privilege to modify users' passwords

You can grant the privilege to modify users' passwords and whether those passwords must be changed upon first login using the `SET PASSWORDS` privilege.
For example:

```
GRANT SET PASSWORDS ON DBMS TO passwordModifier
```

As a result, the `passwordModifier` role has privileges that only allow modifying users' passwords and whether those passwords must be changed upon first login. To list all privileges for the role `passwordModifier` as commands, use the following query:

```
SHOW ROLE passwordModifier PRIVILEGES AS COMMANDS;
```

Table 518. Result

command
"GRANT SET PASSWORD ON DBMS TO `passwordModifier`"
Rows: 1

The `SET PASSWORDS` privilege allows the user to run the `ALTER USER` administration command with one or both of the `SET PASSWORD` and `SET PASSWORD CHANGE [NOT] REQUIRED` parts.

```
ALTER USER jake SET PASSWORD 'abcd5678' CHANGE NOT REQUIRED
```

Grant privilege to modify users' auth information

You can grant the privilege to modify users' auth information using the `SET AUTH` privilege.
For example:

```
GRANT SET AUTH ON DBMS TO authModifier
```

As a result, the `authModifier` role has privileges that only allow modifying users' auth information.

The `SET AUTH` privilege allows the user to run the `ALTER USER` administration command with one or both of the `SET AUTH` and `REMOVE AUTH` parts.
For example:

```
ALTER USER jake REMOVE AUTH 'native' SET AUTH 'oidc-okta' { SET id 'jakesUniqueOktaUserId' }
```

Grant privilege to modify the account status of users

You can grant the privilege to modify the account status of users using the `SET USER STATUS` privilege.

For example:

```
GRANT SET USER STATUS ON DBMS TO statusModifier
```

As a result, the `statusModifier` role has privileges that only allow modifying the account status of users.
To list all privileges for the role `statusModifier` as commands, use the following query:

```
SHOW ROLE statusModifier PRIVILEGES AS COMMANDS;
```

Table 519. Result

command
"GRANT SET USER STATUS ON DBMS TO `statusModifier`"
Rows: 1

The `SET USER STATUS` privilege allows the user to run the `ALTER USER` administration command with only the `SET STATUS` part:

```
ALTER USER jake SET STATUS ACTIVE
```

Grant privilege to modify the home database of users

You can grant the privilege to modify the home database of users using the `SET USER HOME DATABASE` privilege.
For example:

```
GRANT SET USER HOME DATABASE ON DBMS TO homeDbModifier
```

As a result, the `homeDbModifier` role has privileges that only allow modifying the home database of users.
To list all privileges for the role `homeDbModifier` as commands, use the following query:

```
SHOW ROLE homeDbModifier PRIVILEGES AS COMMANDS;
```

Table 520. Result

command
"GRANT SET USER HOME DATABASE ON DBMS TO `homeDbModifier`"
"GRANT SET USER STATUS ON DBMS TO `homeDbModifier`"
Rows: 2

The `SET USER HOME DATABASE` privilege allows the user to run the `ALTER USER` administration command with only the `SET HOME DATABASE` or `REMOVE HOME DATABASE` part:

```
ALTER USER jake SET HOME DATABASE otherDb
```

```
ALTER USER jake REMOVE HOME DATABASE
```

Grant privilege to delete users

You can grant the privilege to delete users using the `DROP USER` privilege.
For example:

```
GRANT DROP USER ON DBMS TO userDropper
```

As a result, the `userDropper` role has privileges that only allow deleting users. To list all privileges for the role `userDropper` as commands, use the following query:

```
SHOW ROLE userDropper PRIVILEGES AS COMMANDS;
```

Table 521. Result

command
"GRANT DROP USER ON DBMS TO `userDropper`"
Rows: 1

Grant privilege to show users

You can grant the privilege to show users using the `SHOW USER` privilege.
For example:

```
GRANT SHOW USER ON DBMS TO userViewer
```

As a result, the `userViewer` role has privileges that only allow showing users. To list all privileges for the role `userViewer` as commands, use the following query:

```
SHOW ROLE userViewer PRIVILEGES AS COMMANDS;
```

Table 522. Result

command
"GRANT SHOW USER ON DBMS TO `userViewer`"
Rows: 1

Grant privilege to manage users

You can grant the privilege to create, rename, modify, delete, and list users using the `USER MANAGEMENT` privilege.
For example:

```
GRANT USER MANAGEMENT ON DBMS TO userManager
```

As a result, the `userManager` role has all privileges to manage users. To list all privileges for the role `userManager` as commands, use the following query:

```
SHOW ROLE userManager PRIVILEGES AS COMMANDS;
```

Table 523. Result

command
"GRANT SHOW USER ON DBMS TO `userManager`"
Rows: 1

The DBMS `IMPERSONATE` privileges

Enterprise Edition | AuraDB Business Critical | AuraDB Virtual Dedicated Cloud

The DBMS privileges for impersonation can be granted, denied, or revoked like other privileges.

Impersonation is the ability of a user to assume another user's roles (and therefore privileges), with the restriction of not being able to execute updating `admin` commands as the impersonated user (i.e. they would still be able to use `SHOW` commands).

You can use the `IMPERSONATE` privilege to allow a user to impersonate another user.

> **Note:**
>
> For more details about the syntax descriptions, see Reading the administration commands syntax.

Table 524. Impersonation privileges command syntax

Command	Description
`GRANT [IMMUTABLE] IMPERSONATE [(*)] ON DBMS TO role[, ...]`	Enables the specified roles to impersonate any user.
`GRANT [IMMUTABLE] IMPERSONATE (user[, ...]) ON DBMS TO role[, ...]`	Enables the specified roles to impersonate the specified users.

Grant privilege to impersonate all users

You can grant the privilege to impersonate all users using the `IMPERSONATE (*)` privilege.
For example:

Query

```
GRANT IMPERSONATE (*) ON DBMS TO allUserImpersonator
```

As a result, the `allUserImpersonator` role has privileges that allow impersonating all users. To list all privileges for the role `allUserImpersonator` as commands, use the following query:

Query

```
SHOW ROLE allUserImpersonator PRIVILEGES AS COMMANDS;
```

Table 525. Result

command
"GRANT IMPERSONATE (*) ON DBMS TO `allUserImpersonator`"
Rows: 1

Grant privilege to impersonate specific users

You can also grant the privilege to impersonate specific users or a subset of users.
For example:

Query

```
GRANT IMPERSONATE (alice, bob) ON DBMS TO userImpersonator;
```

As a result, the `userImpersonator` role has privileges that allow impersonating only `alice` and `bob`. Then, you deny the privilege to impersonate `alice`:

Query

```
DENY IMPERSONATE (alice) ON DBMS TO userImpersonator;
```

As a result, the `userImpersonator` user would be able to impersonate only `bob`.

To list all privileges for the role `userImpersonator` as commands, use the following query:

Query

```
SHOW ROLE userImpersonator PRIVILEGES AS COMMANDS;
```

Table 526. Result

command
"DENY IMPERSONATE (alice) ON DBMS TO `userImpersonator`"
"GRANT IMPERSONATE (alice) ON DBMS TO `userImpersonator`"
"GRANT IMPERSONATE (bob) ON DBMS TO `userImpersonator`"
Rows: 3

The DBMS `DATABASE MANAGEMENT` privileges

Enterprise Edition | AuraDB Business Critical | AuraDB Virtual Dedicated Cloud

The DBMS privileges for database management can be granted, denied, or revoked like other privileges.

> **Note:**
>
> For more details about the syntax descriptions, see Reading the administration commands syntax.

Table 527. Database management privileges command syntax

Command	Description
`GRANT [IMMUTABLE] CREATE DATABASE ON DBMS TO role[, ...]`	Enables the specified roles to create new standard databases.
`GRANT [IMMUTABLE] DROP DATABASE ON DBMS TO role[, ...]`	Enables the specified roles to delete standard databases.
`GRANT [IMMUTABLE] ALTER DATABASE ON DBMS TO role[, ...]`	Enables the specified roles to modify standard databases.
`GRANT [IMMUTABLE] SET DATABASE ACCESS ON DBMS TO role[, ...]`	Enables the specified roles to modify access to standard databases.
`GRANT CREATE COMPOSITE DATABASE ON DBMS TO role[, ...]`	Enables the specified roles to create new composite databases.
`GRANT DROP COMPOSITE DATABASE ON DBMS TO role[, ...]`	Enables the specified roles to delete composite databases.
`GRANT COMPOSITE DATABASE MANAGEMENT ON DBMS TO role[, ...]`	Enables the specified roles to create and delete composite databases.
`GRANT [IMMUTABLE] DATABASE MANAGEMENT ON DBMS TO role[, ...]`	Enables the specified roles to create, delete, and modify databases.

Grant privilege to create standard databases

You can grant the privilege to create standard databases using the `CREATE DATABASE` privilege.
For example:

```
GRANT CREATE DATABASE ON DBMS TO databaseAdder
```

As a result, the `databaseAdder` role has privileges that only allow creating standard databases. To list all privileges for the role `databaseAdder` as commands, use the following query:

```
SHOW ROLE databaseAdder PRIVILEGES AS COMMANDS;
```

Table 528. Result

command
"GRANT CREATE DATABASE ON DBMS TO `databaseAdder`"
Rows: 1

Grant privilege to create composite databases

You can grant the privilege to create composite databases using the `CREATE COMPOSITE DATABASE` privilege.
For example:

```
GRANT CREATE COMPOSITE DATABASE ON DBMS TO compositeDatabaseAdder
```

As a result, the `compositeDatabaseAdder` role has privileges that only allow creating composite databases.
To list all privileges for the role `compositeDatabaseAdder` as commands, use the following query:

```
SHOW ROLE compositeDatabaseAdder PRIVILEGES AS COMMANDS;
```

Table 529. Result

command
"GRANT CREATE COMPOSITE DATABASE ON DBMS TO `compositeDatabaseAdder`"
Rows: 1

Grant privilege to delete standard databases

You can grant the privilege to delete standard databases using the `DROP DATABASE` privilege.
For example:

```
GRANT DROP DATABASE ON DBMS TO databaseDropper
```

As a result, the `databaseDropper` role has privileges that only allow deleting standard databases. To list all privileges for the role `databaseDropper` as commands, use the following query:

```
SHOW ROLE databaseDropper PRIVILEGES AS COMMANDS;
```

Table 530. Result

command
"GRANT DROP DATABASE ON DBMS TO `databaseDropper`"

command
Rows: 1

Grant privilege to delete composite databases

You can grant the privilege to delete composite databases using the `DROP COMPOSITE DATABASE` privilege.
For example:

```
GRANT DROP COMPOSITE DATABASE ON DBMS TO compositeDatabaseDropper
```

As a result, the `compositeDatabaseDropper` role has privileges that only allow deleting composite databases. To list all privileges for the role `compositeDatabaseDropper` as commands, use the following query:

```
SHOW ROLE compositeDatabaseDropper PRIVILEGES AS COMMANDS;
```

Table 531. Result

command
"GRANT DROP COMPOSITE DATABASE ON DBMS TO `compositeDatabaseDropper`"
Rows: 1

Grant privilege to modify standard databases

You can grant the privilege to modify standard databases using the `ALTER DATABASE` privilege.
For example:

```
GRANT ALTER DATABASE ON DBMS TO databaseModifier;
```

As a result, the `databaseModifier` role has privileges that only allow modifying standard databases. To list all privileges for the role `databaseModifier` as commands, use the following query:

```
SHOW ROLE databaseModifier PRIVILEGES AS COMMANDS;
```

Table 532. Result

command
"GRANT ALTER DATABASE ON DBMS TO `databaseModifier`"
Rows: 1

Grant privilege to modify access to standard databases

You can grant the privilege to modify access to standard databases using the `SET DATABASE ACCESS` privilege.
For example:

```
GRANT SET DATABASE ACCESS ON DBMS TO accessModifier
```

As a result, the `accessModifier` role has privileges that only allow modifying access to standard databases.
To list all privileges for the role `accessModifier` as commands, use the following query:

```
SHOW ROLE accessModifier PRIVILEGES AS COMMANDS;
```

Table 533. Result

command
"GRANT SET DATABASE ACCESS ON DBMS TO `accessModifier`"
Rows: 1

Grant privilege to manage composite databases

You can grant the privilege to create, delete, and modify composite databases using the `COMPOSITE DATABASE MANAGEMENT` privilege.
For example:

```
GRANT COMPOSITE DATABASE MANAGEMENT ON DBMS TO compositeDatabaseManager;
```

As a result, the `compositeDatabaseManager` role has all privileges to manage composite databases. To list all privileges for the role `compositeDatabaseManager` as commands, use the following query:

```
SHOW ROLE compositeDatabaseManager PRIVILEGES AS COMMANDS;
```

Table 534. Result

command
"GRANT COMPOSITE DATABASE MANAGEMENT ON DBMS TO `compositeDatabaseManager`"
Rows: 1

Grant privilege to manage standard and composite databases

You can grant the privilege to create, delete, and modify standard and composite databases using the `DATABASE MANAGEMENT` privilege.
For example:

```
GRANT DATABASE MANAGEMENT ON DBMS TO databaseManager;
```

As a result, the `databaseManager` role has all privileges to manage standard and composite databases. To list all privileges for the role `databaseManager` as commands, use the following query:

```
SHOW ROLE databaseManager PRIVILEGES AS COMMANDS;
```

Table 535. Result

command
"GRANT DATABASE MANAGEMENT ON DBMS TO `databaseManager`"
Rows: 1

The DBMS `ALIAS MANAGEMENT` privileges

Enterprise Edition | AuraDB Business Critical | AuraDB Virtual Dedicated Cloud

The DBMS privileges for alias management can be granted, denied, or revoked like other privileges. This applies to both local and remote aliases.

> **Note:**
>
> For more details about the syntax descriptions, see Reading the administration commands syntax.

Table 536. Alias management privileges command syntax

Command	Description
`GRANT [IMMUTABLE] CREATE ALIAS ON DBMS TO role[, ...]`	Enables the specified roles to create new aliases.
`GRANT [IMMUTABLE] DROP ALIAS ON DBMS TO role[, ...]`	Enables the specified roles to delete aliases.
`GRANT [IMMUTABLE] ALTER ALIAS ON DBMS TO role[, ...]`	Enables the specified roles to modify aliases.
`GRANT [IMMUTABLE] SHOW ALIAS ON DBMS TO role[, ...]`	Enables the specified roles to list aliases.
`GRANT [IMMUTABLE] ALIAS MANAGEMENT ON DBMS TO role[, ...]`	Enables the specified roles to list, create, delete, and modify aliases.

Grant privilege to create aliases

You can grant the privilege to create aliases using the `CREATE ALIAS` privilege.
For example:

```
GRANT CREATE ALIAS ON DBMS TO aliasAdder;
```

As a result, the `aliasAdder` role has privileges that only allow creating aliases. To list all privileges for the

role `aliasAdder` as commands, use the following query:

```
SHOW ROLE aliasAdder PRIVILEGES AS COMMANDS;
```

Table 537. Result

command
"GRANT CREATE ALIAS ON DBMS TO `aliasAdder`"
Rows: 1

Grant privilege to delete aliases

You can grant the privilege to delete aliases using the `DROP ALIAS` privilege.
For example:

```
GRANT DROP ALIAS ON DBMS TO aliasDropper;
```

As a result, the `aliasDropper` role has privileges that only allow deleting aliases. See all privileges for the role `aliasDropper` as commands, use the following query:

```
SHOW ROLE aliasDropper PRIVILEGES AS COMMANDS;
```

Table 538. Result

command
"GRANT DROP ALIAS ON DBMS TO `aliasDropper`"
Rows: 1

Grant privilege to modify aliases

You can grant the privilege to modify aliases using the `ALTER ALIAS` privilege.
For example:

```
GRANT ALTER ALIAS ON DBMS TO aliasModifier;
```

As a result, the `aliasModifier` role has privileges that only allow modifying aliases. To list all privileges for the role `aliasModifier` as commands, use the following query:

```
SHOW ROLE aliasModifier PRIVILEGES AS COMMANDS;
```

Table 539. Result

command
"GRANT ALTER ALIAS ON DBMS TO `aliasModifier`"

command
Rows: 1

Grant privilege to list aliases

You can grant the privilege to list aliases using the `SHOW ALIAS` privilege.
For example:

```
GRANT SHOW ALIAS ON DBMS TO aliasViewer;
```

As a result, the `aliasViewer` role has privileges that only allow modifying aliases. To list all privileges for the role `aliasViewer` as commands, use the following query:

```
SHOW ROLE aliasViewer PRIVILEGES AS COMMANDS;
```

Table 540. Result

command
"GRANT SHOW ALIAS ON DBMS TO `aliasViewer`"
Rows: 1

Grant privilege to manage aliases

You can grant the privilege to create, delete, modify, and list aliases using the `ALIAS MANAGEMENT` privilege.
For example:

```
GRANT ALIAS MANAGEMENT ON DBMS TO aliasManager;
```

As a result, the `aliasManager` role has all privileges to manage aliases. To list all privileges for the role `aliasManager` as commands, use the following query:

```
SHOW ROLE aliasManager PRIVILEGES AS COMMANDS;
```

Table 541. Result

command
"GRANT ALIAS MANAGEMENT ON DBMS TO `aliasManager`"
Rows: 1

The DBMS `SERVER MANAGEMENT` privileges

Enterprise Edition | AuraDB Business Critical | AuraDB Virtual Dedicated Cloud

The DBMS privileges for server management can be granted, denied, or revoked like other privileges.

Note:

For more details about the syntax descriptions, see Reading the administration commands syntax.

Table 542. Server management privileges command syntax

Command	Description
`GRANT [IMMUTABLE] SERVER MANAGEMENT ON DBMS TO role[, ...]`	Enables the specified roles to show, enable, rename, alter, reallocate, deallocate, and drop servers.
`GRANT [IMMUTABLE] SHOW SERVERS ON DBMS TO role[, ...]`	Enables the specified roles to show servers.

Grant privilege to manage servers

You can grant the privilege to show, enable, rename, alter, reallocate, deallocate, and drop servers using the `SERVER MANAGEMENT` privilege.
For example:

```
GRANT SERVER MANAGEMENT ON DBMS TO serverManager;
```

As a result, the `serverManager` role has all privileges to manage servers. To list all privileges for the role `serverManager` as commands, use the following query:

```
SHOW ROLE serverManager PRIVILEGES AS COMMANDS;
```

Table 543. Result

command
"GRANT SERVER MANAGEMENT ON DBMS TO `serverManager`"
Rows: 1

Grant privilege to show servers

You can grant the privilege to show servers using the `SHOW SERVERS` privilege.
For example:

```
GRANT SHOW SERVERS ON DBMS TO serverViewer;
```

As a result, the `serverViewer` role has privileges that only allow showing servers. To list all privileges for the role `serverViewer` as commands, use the following query:

```
SHOW ROLE serverViewer PRIVILEGES AS COMMANDS;
```

Table 544. Result

command
"GRANT SHOW SERVERS ON DBMS TO `serverViewer`"
Rows: 1

The DBMS `PRIVILEGE MANAGEMENT` privileges

Enterprise Edition | AuraDB Business Critical | AuraDB Virtual Dedicated Cloud

The DBMS privileges for privilege management can be granted, denied, or revoked like other privileges.

> **Note:**
>
> For more details about the syntax descriptions, see Reading the administration commands syntax.

Table 545. Privilege management privileges command syntax

Command	Description
`GRANT [IMMUTABLE] SHOW PRIVILEGE ON DBMS TO role[, ...]`	Enables the specified roles to list privileges.
`GRANT [IMMUTABLE] ASSIGN PRIVILEGE ON DBMS TO role[, ...]`	Enables the specified roles to assign privileges using the `GRANT` and `DENY` commands.
`GRANT [IMMUTABLE] REMOVE PRIVILEGE ON DBMS TO role[, ...]`	Enables the specified roles to remove privileges using the `REVOKE` command.
`GRANT [IMMUTABLE] PRIVILEGE MANAGEMENT ON DBMS TO role[, ...]`	Enables the specified roles to list, assign, and remove privileges.

Grant privilege to list privileges

You can grant the `SHOW PRIVILEGE` privilege to allow a user to list privileges using the `SHOW PRIVILEGE`, `SHOW ROLE roleName PRIVILEGES`, and `SHOW USER username PRIVILEGES` administration commands. The `SHOW USER username PRIVILEGES` command also requires the `SHOW USER` privilege.

For example:

```
GRANT SHOW PRIVILEGE ON DBMS TO privilegeViewer;
```

As a result, the `privilegeViewer` role has privileges that only allow showing privileges. To list all privileges for the role `privilegeViewer` as commands, use the following query:

```
SHOW ROLE privilegeViewer PRIVILEGES AS COMMANDS;
```

Table 546. Result

command
"GRANT SHOW PRIVILEGE ON DBMS TO `privilegeViewer`"
Rows: 1

Note:

No specific privileges are required for showing the current user's privileges through the `SHOW USER username PRIVILEGES` or `SHOW USER PRIVILEGES` commands.

If a non-native auth provider like LDAP is in use, `SHOW USER PRIVILEGES` will only work with a limited capacity by making it only possible for a user to show their own privileges. Other users' privileges cannot be listed when using a non-native auth provider.

Grant privilege to assign privileges

You can grant the privilege to assign privileges using the `ASSIGN PRIVILEGE` privilege.
A user with this privilege is allowed to execute `GRANT` and `DENY` administration commands.
For example:

```
GRANT ASSIGN PRIVILEGE ON DBMS TO privilegeAssigner;
```

As a result, the `privilegeAssigner` role has privileges that only allow assigning privileges. To list all privileges for the role `privilegeAssigner` as commands, use the following query:

```
SHOW ROLE privilegeAssigner PRIVILEGES AS COMMANDS;
```

Table 547. Result

command
"GRANT ASSIGN PRIVILEGE ON DBMS TO `privilegeAssigner`"
Rows: 1

Grant privilege to remove privileges

You can grant the privilege to remove privileges from roles using the `REMOVE PRIVILEGE` privilege.
A user with this privilege is allowed to execute `REVOKE` administration commands.
For example:

```
GRANT REMOVE PRIVILEGE ON DBMS TO privilegeRemover;
```

As a result, the `privilegeRemover` role has privileges that only allow removing privileges. To list all

privileges for the role `privilegeRemover` as commands, use the following query:

```
SHOW ROLE privilegeRemover PRIVILEGES AS COMMANDS;
```

Table 548. Result

command
"GRANT REMOVE PRIVILEGE ON DBMS TO `privilegeRemover`"
Rows: 1

Grant privilege to manage privileges

You can grant the privilege to list, assign, and remove privileges using the `PRIVILEGE MANAGEMENT` privilege. For example:

```
GRANT PRIVILEGE MANAGEMENT ON DBMS TO privilegeManager;
```

As a result, the `privilegeManager` role has all privileges to manage privileges. To list all privileges for the role `privilegeManager` as commands, use the following query:

```
SHOW ROLE privilegeManager PRIVILEGES AS COMMANDS;
```

Table 549. Result

command
"GRANT PRIVILEGE MANAGEMENT ON DBMS TO `privilegeManager`"
Rows: 1

The DBMS `EXECUTE` privileges

Enterprise Edition | AuraDB Business Critical | AuraDB Virtual Dedicated Cloud

The DBMS privileges for procedure and user-defined function execution can be granted, denied, or revoked like other privileges.

> **Note:**
>
> For more details about the syntax descriptions, see Reading the administration commands syntax.

Table 550. Execute privileges command syntax

Command	Description

`GRANT [IMMUTABLE] EXECUTE PROCEDURE[S] name-globbing[, ...]`
`  ON DBMS`
`  TO role[, ...]`
	Enables the specified roles to execute the given procedures.

`GRANT [IMMUTABLE] EXECUTE BOOSTED PROCEDURE[S] name-globbing[, ...]`
`  ON DBMS`
`  TO role[, ...]`
	Enables the specified roles to use elevated privileges when executing the given procedures.

`GRANT [IMMUTABLE] EXECUTE ADMIN[ISTRATOR] PROCEDURES`
`  ON DBMS`
`  TO role[, ...]`
	Enables the specified roles to execute procedures annotated with `@Admin`. The procedures are executed with elevated privileges.

`GRANT [IMMUTABLE] EXECUTE [USER [DEFINED]] FUNCTION[S] name-globbing[, ...]`
`  ON DBMS`
`  TO role[, ...]`
	Enables the specified roles to execute the given user-defined functions.

`GRANT [IMMUTABLE] EXECUTE BOOSTED [USER [DEFINED]] FUNCTION[S] name-globbing[, ...]`
`  ON DBMS`
`  TO role[, ...]`
	Enables the specified roles to use elevated privileges when executing the given user-defined functions.

Grant privilege to execute procedures

You can grant the privilege to execute procedures using the `EXECUTE PROCEDURE` privilege.
A role with this privilege is allowed to execute the procedures matched by the name-globbing.

Grant privilege to execute some procedures

The following query allow the execution of procedures starting with `db.schema`:

```
GRANT EXECUTE PROCEDURE db.schema.* ON DBMS TO procedureExecutor;
```

Users with the role `procedureExecutor` can run any procedure in the `db.schema` namespace. The procedures are executed using the user's own privileges.

To list all privileges for the role `procedureExecutor` as commands, use the following query:

```
SHOW ROLE procedureExecutor PRIVILEGES AS COMMANDS;
```

Table 551. Result

command
"GRANT EXECUTE PROCEDURE db.schema.* ON DBMS TO `procedureExecutor`"
Rows: 1

Grant privilege to execute all but some procedures

You can grant the privilege to execute all except a few procedures using `EXECUTE PROCEDURES *` and deny the unwanted procedures. For example, the following queries allow the execution of all procedures, except those starting with `dbms.cluster`:

```
GRANT EXECUTE PROCEDURE * ON DBMS TO deniedProcedureExecutor;
```

```
DENY EXECUTE PROCEDURE dbms.cluster* ON DBMS TO deniedProcedureExecutor;
```

Users with the role `deniedProcedureExecutor` can run any procedure except those starting with `dbms.cluster`. The procedures are executed using the user's own privileges.

To list all privileges for the role `deniedProcedureExecutor` as commands, use the following query:

```
SHOW ROLE deniedProcedureExecutor PRIVILEGES AS COMMANDS;
```

Table 552. Result

command
"DENY EXECUTE PROCEDURE dbms.cluster* ON DBMS TO `deniedProcedureExecutor`"
"GRANT EXECUTE PROCEDURE * ON DBMS TO `deniedProcedureExecutor`"
Rows: 2

The `dbms.cluster.checkConnectivity`, `dbms.cluster.cordonServer`, `dbms.cluster.protocols`, `dbms.cluster.readReplicaToggle`, `dbms.cluster.routing.getRoutingTable`, `dbms.cluster.secondaryReplicationDisable`, `dbms.cluster.setAutomaticallyEnableFreeServers`, and `dbms.cluster.uncordonServer` procedures are blocked, as well as any others starting with `dbms.cluster`.

Grant privilege to execute procedures with elevated privileges

You can grant the privilege to execute procedures with elevated privileges using the `EXECUTE BOOSTED PROCEDURE` privilege.
A user with this privilege will not be restricted to their other privileges when executing the procedures matched by the name-globbing. The `EXECUTE BOOSTED PROCEDURE` privilege only affects the elevation, and not the execution of the procedure. Therefore, it is needed to grant `EXECUTE PROCEDURE` privilege for the procedures as well. Both `EXECUTE PROCEDURE` and `EXECUTE BOOSTED PROCEDURE` are needed to execute a procedure with elevated privileges.

Grant privilege to execute some procedures with elevated privileges

You can grant the privilege to execute some procedures with elevated privileges using `EXECUTE BOOSTED PROCEDURE *`.

For example, the following query allow the execution of the procedures `db.labels` and `db.relationshipTypes` with elevated privileges, and all other procedures with the user's own privileges:

```
GRANT EXECUTE PROCEDURE * ON DBMS TO boostedProcedureExecutor;
```

```
GRANT EXECUTE BOOSTED PROCEDURE db.labels, db.relationshipTypes ON DBMS TO boostedProcedureExecutor
```

Users with the role `boostedProcedureExecutor` can thus run the `db.labels` and the `db.relationshipTypes` procedures with full privileges, seeing everything in the graph and not just the labels and types that the user has `TRAVERSE` privilege on. Without the `EXECUTE PROCEDURE`, no procedures could be executed at all.

To list all privileges for the role `boostedProcedureExecutor` as commands, use the following query:

```
SHOW ROLE boostedProcedureExecutor PRIVILEGES AS COMMANDS;
```

Table 553. Result

command
"GRANT EXECUTE PROCEDURE * ON DBMS TO `boostedProcedureExecutor`"
"GRANT EXECUTE BOOSTED PROCEDURE db.labels ON DBMS TO `boostedProcedureExecutor`"
"GRANT EXECUTE BOOSTED PROCEDURE db.relationshipTypes ON DBMS TO `boostedProcedureExecutor`"
Rows: 3

Combination of granting execution and denying privilege elevation

As with grant, denying `EXECUTE BOOSTED PROCEDURE` on its own only affects the elevation and not the execution of the procedure.

For example:

```
GRANT EXECUTE PROCEDURE * ON DBMS TO deniedBoostedProcedureExecutor1;
DENY EXECUTE BOOSTED PROCEDURE db.labels ON DBMS TO deniedBoostedProcedureExecutor1;
```

As a result, the `deniedBoostedProcedureExecutor1` role has privileges that allow the execution of all procedures using the user's own privileges. They also prevent the `db.labels` procedure from being elevated. Still, the denied `EXECUTE BOOSTED PROCEDURE` does not block execution of `db.labels`.

To list all privileges for role `deniedBoostedProcedureExecutor1` as commands, use the following query:

```
SHOW ROLE deniedBoostedProcedureExecutor1 PRIVILEGES AS COMMANDS;
```

Table 554. Result

command
"DENY EXECUTE BOOSTED PROCEDURE db.labels ON DBMS TO `deniedBoostedProcedureExecutor1`"
"GRANT EXECUTE PROCEDURE * ON DBMS TO `deniedBoostedProcedureExecutor1`"
Rows: 2

Combination of granting privilege elevation and denying execution

You can also grant the privilege to execute procedures with elevated privileges and deny the execution of

specific procedures.

For example:

```
GRANT EXECUTE BOOSTED PROCEDURE * ON DBMS TO deniedBoostedProcedureExecutor2;
```

```
DENY EXECUTE PROCEDURE db.labels ON DBMS TO deniedBoostedProcedureExecutor2;
```

As a result, the `deniedBoostedProcedureExecutor2` role has privileges that allow elevating the privileges for all procedures, but cannot execute any due to missing or denied `EXECUTE PROCEDURE` privileges.

To list all privileges for the role `deniedBoostedProcedureExecutor2` as commands, use the following query:

```
SHOW ROLE deniedBoostedProcedureExecutor2 PRIVILEGES AS COMMANDS;
```

Table 555. Result

command
"DENY EXECUTE PROCEDURE db.labels ON DBMS TO `deniedBoostedProcedureExecutor2`"
"GRANT EXECUTE BOOSTED PROCEDURE * ON DBMS TO `deniedBoostedProcedureExecutor2`"
Rows: 2

Combination of granting and denying privilege elevation

You can also grant the privilege to execute procedures with elevated privileges and deny the elevation for specific procedures.

For example, the following queries allow has privileges that allow elevating the privileges for all procedures except `db.labels`. However, no procedures can be executed due to a missing `EXECUTE PROCEDURE` privilege.

```
GRANT EXECUTE BOOSTED PROCEDURE * ON DBMS TO deniedBoostedProcedureExecutor3;
```

```
DENY EXECUTE BOOSTED PROCEDURE db.labels ON DBMS TO deniedBoostedProcedureExecutor3;
```

As a result, the `deniedBoostedProcedureExecutor3` role has privileges that allow elevating the privileges for all procedures except `db.labels`. However, no procedures can be executed due to missing `EXECUTE PROCEDURE` privilege.

To list all privileges for the role `deniedBoostedProcedureExecutor3` as commands, use the following query:

```
SHOW ROLE deniedBoostedProcedureExecutor3 PRIVILEGES AS COMMANDS;
```

Table 556. Result

command
"DENY EXECUTE BOOSTED PROCEDURE db.labels ON DBMS TO `deniedBoostedProcedureExecutor3`"
"GRANT EXECUTE BOOSTED PROCEDURE * ON DBMS TO `deniedBoostedProcedureExecutor3`"
Rows: 2

Control procedure output with privileges

You can control the output of procedures based on the privileges granted or denied to a role using the `EXECUTE PROCEDURE` and `EXECUTE BOOSTED PROCEDURE` privileges. For example, assume there is a procedure called `myProc`.

This procedure gives the result `A` and `B` for a user with only the `EXECUTE PROCEDURE` privilege and `A`, `B` and `C` for a user with both the `EXECUTE PROCEDURE` and `EXECUTE BOOSTED PROCEDURE` privileges.

Now, adapt the privileges from sections Combination of granting execution and denying privilege elevation (example 1), Combination of granting privilege elevation and denying execution (example 2), and Combination of granting and denying privilege elevations (example 3) to be applied to this procedure and show what is returned.

With the privileges from example 1, granted `EXECUTE PROCEDURE *` and denied `EXECUTE BOOSTED PROCEDURE myProc`, the `myProc` procedure returns the result `A` and `B`.

With the privileges from example 2, granted `EXECUTE BOOSTED PROCEDURE *` and denied `EXECUTE PROCEDURE myProc`, execution of the `myProc` procedure is not allowed.

With the privileges from example 3, granted `EXECUTE BOOSTED PROCEDURE *` and denied `EXECUTE BOOSTED PROCEDURE myProc`, execution of the `myProc` procedure is not allowed.

For comparison, when granted:

- `EXECUTE PROCEDURE myProc`: the `myProc` procedure returns the result `A` and `B`.
- `EXECUTE BOOSTED PROCEDURE myProc`: execution of the `myProc` procedure is not allowed.
- `EXECUTE PROCEDURE myProc` and `EXECUTE BOOSTED PROCEDURE myProc`: the `myProc` procedure returns the result `A`, `B`, and `C`.

Grant privilege to execute admin procedures

Admin procedures (annotated with `@Admin`) are special in that they require elevated privileges to be executed at all. This means that to execute an admin procedure you need both the `EXECUTE PROCEDURE` and `EXECUTE BOOSTED PROCEDURE` privileges for that procedure.

For a user to be allowed to execute all admin procedures, they can either be granted the two privileges for each of the admin procedures (which would need to be updated each time a new admin procedure is added), all procedures (which would then affect all non-admin procedures as well) or the `EXECUTE ADMIN PROCEDURES` privilege.

The `EXECUTE ADMIN PROCEDURES` privilege is equivalent to granting the `EXECUTE PROCEDURE` and `EXECUTE`

`BOOSTED PROCEDURE` privileges on each of the admin procedures. This also have the additional advantage that any newly added `admin` procedure is automatically included in this privilege.
For example:

```
GRANT EXECUTE ADMIN PROCEDURES ON DBMS TO adminProcedureExecutor;
```

Users with the role `adminProcedureExecutor` can run any `admin` procedure with elevated privileges. As a result, the `adminProcedureExecutor` role has privileges that allow the execution of all admin procedures. To list all privileges for the role `adminProcedureExecutor` as commands, use the following query:

```
SHOW ROLE adminProcedureExecutor PRIVILEGES AS COMMANDS;
```

Table 557. Result

command
"GRANT EXECUTE ADMIN PROCEDURES ON DBMS TO `adminProcedureExecutor`"
Rows: 1

In order to compare this with the `EXECUTE PROCEDURE` and `EXECUTE BOOSTED PROCEDURE` privileges, revisit the `myProc` procedure, but this time as an `admin` procedure, which will give the result `A`, `B` and `C` when allowed to execute.

By starting with a user only granted the `EXECUTE PROCEDURE myProc` or the `EXECUTE BOOSTED PROCEDURE myProc` privilege, execution of the `myProc` procedure is not allowed.

However, for a user granted the `EXECUTE ADMIN PROCEDURES` or both `EXECUTE PROCEDURE myProc` and `EXECUTE BOOSTED PROCEDURE myProc`, the `myProc` procedure returns the result `A`, `B` and `C`.

Any denied `EXECUTE` privilege results in the procedure not being allowed to be executed. In this case, it does not matter whether `EXECUTE PROCEDURE`, `EXECUTE BOOSTED PROCEDURE` or `EXECUTE ADMIN PROCEDURES` is being denied.

Grant privilege to execute user-defined functions

You can grant the privilege to execute user-defined functions (UDFs) using the `EXECUTE USER DEFINED FUNCTION` privilege. A role with this privilege is allowed to execute the UDFs matched by the name-globbing.

Important:

The `EXECUTE USER DEFINED FUNCTION` privilege does not apply to built-in functions, which are always executable.

Grant privilege to execute some user-defined functions

The following query shows an example of how to grant the `EXECUTE USER DEFINED FUNCTION` privilege:

```
GRANT EXECUTE USER DEFINED FUNCTION apoc.coll.* ON DBMS TO functionExecutor;
```

Or in short form:

```
GRANT EXECUTE FUNCTION apoc.coll.* ON DBMS TO functionExecutor;
```

Users with the role `functionExecutor` can thus run any UDF in the `apoc.coll` namespace. The functions are executed using the user's own privileges.

As a result, the `functionExecutor` role has privileges that only allow executing UDFs in the `apoc.coll` namespace. To list all privileges for the role `functionExecutor` as commands, use the following query:

```
SHOW ROLE functionExecutor PRIVILEGES AS COMMANDS;
```

Table 558. Result

command
"GRANT EXECUTE FUNCTION apoc.coll.* ON DBMS TO `functionExecutor`"
Rows: 1

Grant privilege to execute all but some user-defined functions

To allow the execution of all but a few UDFs, you can grant `EXECUTE USER DEFINED FUNCTIONS *` and deny the unwanted UDFs. For example, the following queries allow the execution of all UDFs except those starting with `apoc.any.prop`:

```
GRANT EXECUTE USER DEFINED FUNCTIONS * ON DBMS TO deniedFunctionExecutor;
```

```
DENY EXECUTE USER DEFINED FUNCTION apoc.any.prop* ON DBMS TO deniedFunctionExecutor;
```

Or in short form:

```
GRANT EXECUTE FUNCTIONS * ON DBMS TO deniedFunctionExecutor;
```

```
DENY EXECUTE FUNCTION apoc.any.prop* ON DBMS TO deniedFunctionExecutor;
```

As a result, the `deniedFunctionExecutor` role has privileges that only allow the execution of all UDFs except those starting with `apoc.any.prop`. The functions are executed using the user's own privileges. To list all privileges for the role `deniedFunctionExecutor` as commands, use the following query:

```
SHOW ROLE deniedFunctionExecutor PRIVILEGES AS COMMANDS;
```

Table 559. Result

command
"DENY EXECUTE FUNCTION apoc.any.prop* ON DBMS TO `deniedFunctionExecutor`"
"GRANT EXECUTE FUNCTION * ON DBMS TO `deniedFunctionExecutor`"
Rows: 2

The `apoc.any.property` and `apoc.any.properties` are blocked, as well as any other UDFs starting with `apoc.any.prop`.

Grant privilege to execute user-defined functions with elevated privileges

You can grant the privilege to execute user-defined functions (UDFs) with elevated privileges using the `EXECUTE BOOSTED USER DEFINED FUNCTION` privilege.
A user with this privilege will not be restricted to their other privileges when executing the UDFs matched by the name-globbing. The `EXECUTE BOOSTED USER DEFINED FUNCTION` privilege only affects the elevation and not the execution of the function. Therefore, it is needed to grant `EXECUTE USER DEFINED FUNCTION` privilege for the UDFs as well. Both `EXECUTE USER DEFINED FUNCTION` and `EXECUTE BOOSTED USER DEFINED FUNCTION` are needed to execute a function with elevated privileges.

Important:

The `EXECUTE BOOSTED USER DEFINED FUNCTION` privilege does not apply to built-in functions, as they have no concept of elevated privileges.

Grant privilege to execute some user-defined functions with elevated privileges

The following query shows an example of how to grant the `EXECUTE BOOSTED USER DEFINED FUNCTION` privilege:

```
GRANT EXECUTE USER DEFINED FUNCTION * ON DBMS TO boostedFunctionExecutor;
GRANT EXECUTE BOOSTED USER DEFINED FUNCTION apoc.any.properties ON DBMS TO boostedFunctionExecutor;
```

Or in short form:

```
GRANT EXECUTE FUNCTION * ON DBMS TO boostedFunctionExecutor;
GRANT EXECUTE BOOSTED FUNCTION apoc.any.properties ON DBMS TO boostedFunctionExecutor;
```

Users with the role `boostedFunctionExecutor` can thus run `apoc.any.properties` with full privileges and see every property on the node/relationship, not just the properties that the user has `READ` privilege on. Without the `EXECUTE USER DEFINED FUNCTION`, you cannot execute any UDFs at all.

As a result, the `boostedFunctionExecutor` role has privileges that allow executing the UDF `apoc.any.properties` with elevated privileges, and all other UDFs with the users' own privileges. To list all privileges for the role `boostedFunctionExecutor` as commands, use the following query:

```
SHOW ROLE boostedFunctionExecutor PRIVILEGES AS COMMANDS;
```

Table 560. Result

command
"GRANT EXECUTE FUNCTION * ON DBMS TO `boostedFunctionExecutor`"
"GRANT EXECUTE BOOSTED FUNCTION apoc.any.properties ON DBMS TO `boostedFunctionExecutor`"
Rows: 2

The DBMS `SETTING` privileges

Enterprise Edition | AuraDB Business Critical | AuraDB Virtual Dedicated Cloud

You can grant the privilege to show configuration settings using the `SHOW SETTING` privilege. A role with this privilege is allowed to list the configuration settings matched by the name-globbing.

> **Note:**
>
> For more details about the syntax descriptions, see Reading the administration commands syntax.

Table 561. Setting privileges command syntax

Command	Description

`GRANT [IMMUTABLE] SHOW SETTING[S] name-globbing[, ...]`
`  ON DBMS`
`  TO role[, ...]`
	Enables the specified roles to list given configuration settings.

Grant privilege to show all settings

You can grant the privilege to show all settings using `SHOW SETTING *` or all settings in a namespace using `SHOW SETTING namespace.*`. The following query shows an example of how to grant `SHOW SETTING` privilege to view all settings in the `server.bolt` namespace:

```
GRANT SHOW SETTING server.bolt.* ON DBMS TO configurationViewer;
```

Users with the role `configurationViewer` can then view any setting in the `server.bolt` namespace.

As a result, the `configurationViewer` role has privileges that only allow listing settings in the `server.bolt` namespace. To list all privileges for the role `configurationViewer` as commands, use the following query:

```
SHOW ROLE configurationViewer PRIVILEGES AS COMMANDS;
```

Table 562. Result

command
"GRANT SHOW SETTING server.bolt.* ON DBMS TO `configurationViewer`"

command
Rows: 1

Grant privilege to show all but some settings

You can grant the privilege to show all but a few settings using `SHOW SETTINGS *` and deny the unwanted settings.
For example, the following queries allow you to view all settings, except those starting with `dbms.security`:

```
GRANT SHOW SETTINGS * ON DBMS TO deniedConfigurationViewer;
```

```
DENY SHOW SETTING dbms.security* ON DBMS TO deniedConfigurationViewer;
```

As a result, the `deniedConfigurationViewer` role has privileges that allow listing all settings except those starting with `dbms.security`. To list all privileges for the role `deniedConfigurationViewer` as commands, use the following query:

```
SHOW ROLE deniedConfigurationViewer PRIVILEGES AS COMMANDS;
```

Table 563. Result

command
"DENY SHOW SETTING dbms.security* ON DBMS TO `deniedConfigurationViewer`"
"GRANT SHOW SETTING * ON DBMS TO `deniedConfigurationViewer`"
Rows: 2

As the query result shows, viewing settings starting with `dbms.security` is blocked, but the rest can still be listed.

Load privileges

This feature is available from Neo4j 5.13.

This section explains how to use Cypher to manage load privileges. All load privileges apply to the whole system. Like DBMS privileges, they do not belong to one specific database or graph. For more details on the differences between graphs, databases, and the DBMS, refer to Cypher Manual → Cypher and Neo4j.

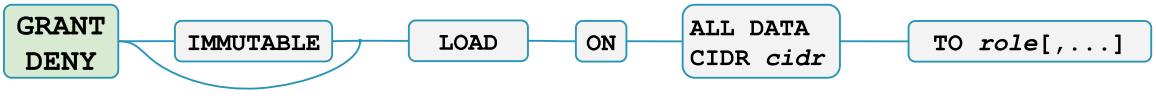


Figure 35. Syntax of GRANT and DENY load Privileges

The load privileges apply to the Cypher `LOAD CSV` clause, deciding whether or not the data can be loaded from the given source.

Load privileges syntax

The load privileges are assigned using Cypher administrative commands. They can be granted, denied, and revoked in the same way as other privileges. For more details, see RBAC and fine-grained privileges.

Table 564. Load privileges command syntax

Command	Description
`GRANT [IMMUTABLE] LOAD ON ALL DATA TO role[, ...]`	Enables the specified roles to load external data in queries.
`GRANT [IMMUTABLE] LOAD ON CIDR cidr TO role[, ...]`	Enables the specified roles to load external data from the given CIDR range in queries. Introduced in 5.16

> **Note:**
>
> More details about the syntax descriptions can be found Cypher syntax for administration commands.

Unlike other privileges, the `LOAD` privilege is not granted, denied, or revoked on `DBMS`, `DATABASE`, or `GRAPH`, but instead on the data source to load from. Adding `ON ALL DATA` means a role has the privilege to load data from all sources. To only allow loading data from sources given by a specific CIDR range use `ON CIDR cidr`.

> **Important:**
>
> It is strongly recommended to permit resource loading only over secure protocols such as HTTPS instead of insecure protocols like HTTP. This can be done by limiting the accessible ranges to only trusted sources that use secure protocols. If allowing an insecure protocol is absolutely unavoidable, Neo4j takes measures internally to enhance the security of these requests within their limitations. However, this means that insecure URLs on virtual hosts will not function unless you add the JVM argument `-Dsun.net.http.allowRestrictedHeaders=true` to the configuration setting `server.jvm.additional`.

The `ALL DATA` privilege

The load privilege on `ALL DATA` enables or disables loading data. If granted, the user can load data from any source. If missing or denied, no data can be loaded at all.

Example 105. Grant users with the role `roleLoadAllData` the ability to load data with `LOAD CSV`

```
GRANT LOAD ON ALL DATA TO roleLoadAllData
```

Example 106. List all privileges for the role `roleLoadAllData` as commands

```
SHOW ROLE roleLoadAllData PRIVILEGES AS COMMANDS
```

Table 565. Result

command
"GRANT LOAD ON ALL DATA TO `roleLoadAllData`"
Rows: 1

The `LOAD ON ALL DATA` privilege is granted to the `PUBLIC` role by default.

The `CIDR` privilege

This feature is available from Neo4j 5.16.

The load privilege on `CIDR cidr` enables or disables loading data from the given IPv4 or IPv6 CIDR range. If granted, the user can load data from sources in the given CIDR range. If missing or denied, no data can be loaded from sources in the given CIDR range.

Example 107. Denies users with the role `roleLoadCidr` the ability to load data with `LOAD CSV` from `127.0.0.1/32`

```
DENY LOAD ON CIDR "127.0.0.1/32" TO roleLoadCidr
```

Example 108. List all privileges for the role `roleLoadCidr` as commands

```
SHOW ROLE roleLoadCidr PRIVILEGES AS COMMANDS
```

Table 566. Result

command
"DENY LOAD ON CIDR "127.0.0.1/32" TO `roleLoadCidr`"
Rows: 1

Limitations

Neo4j's role-based access control (RBAC) has limitations that can lead to unexpected results if not

carefully considered during security model design and when writing Cypher queries. Depending on the restrictions applied to the roles a user are assigned to, query results may differ in the following ways:

- Access control and indexes: Indexes are used to speed up queries, but what they return to the user depends on the security model. If a user has restrictions on certain labels, relationship types, or properties, the system will not return any results that contain those elements, regardless of the indexes used. Also, full-text indexes may return fewer results than expected because Lucene prevents Neo4j from checking security rules for each returned index entry. As a result, Neo4j only returns results it can guarantee do not violate security rules and excludes results that might violate security rules, even if they do not.
- Fail-open `DENY` behavior: A `DENY` rule fails open when its criteria are not met, so Neo4j does not apply the restriction, and it grants access by default if a broader `GRANT` exists. This can lead to unintended data exposure if the `DENY` rule is not carefully crafted.
- Access control and labels: Query results differ when nodes have multiple labels, and the user has access to traverse on some of those labels but not others. This is because the user can see all labels attached to a node they have access to, even if they do not have access to traverse on all of those labels.
- Access control and performance: Complex security rules and large graphs with restrictive access can lead to significant performance degradation, especially for queries that would otherwise be efficient.

Access control and indexes

Neo4j uses indexes to speed up Cypher queries. See the Cypher Manual → Indexes for more details on the different types of indexes available in Neo4j.

However, Neo4j's security model still controls what results you see, regardless of whether or not you use indexes.

Search-performance indexes (range, text, point, and token lookup)

Search-performance indexes are automatically used by the Cypher planner when they are relevant to a query. The results returned by these indexes are filtered by the security model, so that users only see results they have access to. If the security model causes fewer results to be returned due to restricted read access in graph and sub-graph access control, the index will also return the same fewer results.

Semantic index (full-text)

Full-text indexes differ from search-performance indexes because they are built on Lucene and are not automatically used by the Cypher planner.

Lucene implications

Lucene prevents Neo4j from checking the security rules for each specific entry returned from the index. As a result, Neo4j needs to take a more conservative approach to ensure that it does not return results that violate the security rules. If a result **might violate** the security rules, Neo4j treats it as if it **does violate** the security rules and excludes it from the results returned by the index. This means it either returns zero results (if all results from the index are potentially affected) or a partial result (if some rows can be guaranteed not to be affected).

Usage implications

Cypher does not automatically use full-text indexes. You must explicitly state the index you want to use when calling procedures for full-text indexes, such as `db.index.fulltext.queryNodes` and `db.index.fulltext.queryRelationships`. This avoids a situation in which the same Cypher query returns different results because of an underlying index. The problem is that if you do not know this behavior, you might expect the full-text index to return the same results that a different but semantically similar Cypher query does.

The following examples illustrate the implications of using full-text indexes with different security rules, compared to using search-performance indexes.

Example 1: Denied labels and index on multiple properties

The example assumes that the database contains nodes with the labels `:User` and `:Person`, and the properties `name` and `surname`.

1. Create a single property range index on `name`, a composite range index on `name` and `surname`, and a full-text index on both properties:

```
CREATE INDEX singleProp FOR (n:User) ON (n.name);
CREATE INDEX composite  FOR (n:User) ON (n.name, n.surname);
CREATE FULLTEXT INDEX userNames FOR (n:User|Person) ON EACH [n.name, n.surname];
```

Note:

Full-text indexes support multiple labels. See Cypher Manual → Indexes for full-text search for more details on creating and using full-text indexes.

After creating these indexes, it may appear that the latter two indexes accomplish the same thing. However, this is not completely accurate.
The composite and full-text indexes behave differently and are focused on different use cases. A key difference is that full-text indexes are backed by *Lucene* and use the *Lucene* syntax for querying. This has consequences for users restricted to the labels or properties involved in the indexes. Ideally, if the labels and properties in the index are denied, they can correctly return zero results from both native indexes and full-text indexes. However, there are borderline cases where this is not as simple as it seems.

2. Let's add the following nodes to the database:

```
CREATE (:User {name: 'Sandy'});
CREATE (:User {name: 'Mark', surname: 'Andy'});
CREATE (:User {name: 'Andy', surname: 'Anderson'});
CREATE (:User:Person {name: 'Mandy', surname: 'Smith'});
CREATE (:User:Person {name: 'Joe', surname: 'Andy'});
```

3. Deny the label `:Person`:

```
DENY TRAVERSE ON GRAPH * NODES Person TO users;
```

4. Now run a query that uses one of the indexes:

Run a query that uses the single property range index on `name`:

```
MATCH (n:User)
WHERE n.name CONTAINS 'ndy'
RETURN n.name
```

This query performs several checks:

- Scans the index to create a stream of results of nodes with the `name` property, which leads to five results.
- Filters the results to include only nodes where `n.name CONTAINS 'ndy'`, filtering out `Mark` and `Joe`, which leads to three results.
- Filters the results to exclude nodes that also have the denied label `:Person`, filtering out `Mandy`, leaving two results.

Finally, the query returns two results, but only one has the `surname` property.

Run a query that uses the composite range index on `name` and `surname`. To use the composite range index on `name` and `surname`, the query needs to include a predicate on the `surname` property as well:

```
MATCH (n:User)
WHERE n.name CONTAINS 'ndy' AND n.surname IS NOT NULL
RETURN n.name
```

This query performs several checks, which are almost identical to the single property index query:

- Scans the index to create a stream of results of nodes with the `name` and `surname` property, which leads to four results.
- Filters the results to include only nodes where `n.name CONTAINS 'ndy'`, filtering out `Mark` and `Joe`, which leads to two results.
- Filters the results to exclude nodes that also have the denied label `:Person`, filtering out `Mandy`, leaving only one result.

However, in the end, it returns only one result.

Run the same query as for the composite index, but using the full-text index instead:

```
CALL db.index.fulltext.queryNodes("userNames", "ndy") YIELD node, score
RETURN node.name
```

The problem now is that it is not clear whether the results provided by the index are achieved due to a match to the `name` or the `surname` property.

The query performs the following steps:

- Run a *Lucene* query on the full-text index to return results containing `ndy` in either property, leading to five results.
- Filter the results to exclude nodes that also have the label `:Person`, filtering out `Mandy` and `Joe`, leading to three results.

This difference in results is caused by the `OR` relationship between the two properties in the index creation.

Example 2: Denied properties on all labels

The example uses the same database as in Example 1: Denied labels and index on multiple properties, but instead of denying the `:Person` label, it denies the `surname` property for all labels.

1. Deny the `surname` property for all labels:

```
DENY READ {surname} ON GRAPH * TO users;
```

2. Run the same queries as in the previous example using one of the indexes:

Run a query that uses the single property range index on `name`:

```
MATCH (n:User)
WHERE n.name CONTAINS 'ndy'
RETURN n.name
```

This query operates exactly as before, except for the check on the `:Person` label, because nothing in it relates to the denied property:

- Scans the index to create a stream of results of nodes with the `name` property, which leads to five results.
- Filters the results to include only nodes where `n.name CONTAINS 'ndy'`, filtering out `Mark` and `Joe`, which leads to three results.

Finally, the query returns three results, only one of which has the `surname` property.

Run a query that uses the composite range index on `name` and `surname`:

```
MATCH (n:User)
WHERE n.name CONTAINS 'ndy' AND n.surname IS NOT NULL
RETURN n.name
```

Since the `surname` property is denied, it always appears to be `null` and the composite index to be empty. Therefore, the query returns no result.

Run the same query as for the composite index, but using the full-text index instead:

```
CALL db.index.fulltext.queryNodes("userNames", "ndy") YIELD node, score
RETURN node.name
```

The problem now is that it is not clear whether the results provided by the index are achieved due to a match to the `name` or the `surname` property. Results from the `surname` property now need to be excluded by the security rules, because they require the user not to be able to see any `surname` properties. However, the security model cannot introspect the *Lucene* query to determine what it will actually do, whether it works only on the allowed `name` property or also on the disallowed `surname` property. What is known is that the earlier query returned a match for `Mark Andy`, which should now be filtered out. Therefore, to prevent the index from returning results that the user should not see, the index treats all properties as denied rather than just the `surname` property. The query performs the following steps:

- Run a *Lucene* query on the full-text index to return results containing `ndy` in either property, leading to five results.
- Filter the results to exclude all nodes, filtering out everyone, leading to no result.

In the end, the query returns zero results rather than the expected results `Andy`, `Mandy`, and `Sandy`.

Example 3: Denied properties on a specific label

Instead of denying the `surname` property for all labels, like the example in Example 2: Denied properties on all labels, consider denying reading it only for nodes with the `:Person` label.

1. Deny read access to the `surname` property for nodes with the `:Person` label:

```
DENY READ {surname} ON GRAPH * NODES Person TO users;
```

2. Run the same queries as in the previous examples using one of the indexes:

Run a query that uses the single property range index on `name`:

```
MATCH (n:User)
WHERE n.name CONTAINS 'ndy'
RETURN n.name
```

The query operates exactly as in Example 2: Denied properties on all labels, returning the

same three results, because nothing in it relates to the denied property.

Run a query that uses the composite range index on `name` and `surname`:

```
MATCH (n:User)
WHERE n.name CONTAINS 'ndy' AND n.surname IS NOT NULL
RETURN n.name
```

Since the `surname` property is denied only for nodes with the label `:Person`, it appears to be `null` only for those nodes, and the composite index does not contain them. The composite index still contains the remaining `:User` nodes that have both properties. Therefore, the query returns only one result.

Run a query that uses the full-text index on `name` and `surname`:

```
CALL db.index.fulltext.queryNodes("userNames", "ndy") YIELD node, score
RETURN node.name
```

The problem now is that it is not clear whether the results provided by the index are achieved due to a match to the `name` or the `surname` property. However, the impact is lessened, as only nodes with the label `:Person` need to exclude the results from the `surname` property. Nodes without the `:Person` label can read the `surname` property and do not need to be excluded.

However, the security model cannot introspect the *Lucene* query to determine what it will actually do, whether it works only on the allowed `name` property or also on the disallowed `surname` property. Therefore, to prevent the index from returning results that the user should not see, the index treats all properties of nodes with the label `:Person` as denied, rather than just the `surname` property. Since the `surname` property is denied only for nodes with the label `:Person`, the index can safely return any nodes that do not have that label.

The query performs the following steps:

- Run a *Lucene* query on the full-text index to return results containing `ndy` in either property, leading to five results.
- Filter the results to exclude nodes that also have the label `:Person`, filtering out `Mandy` and `Joe`, leading to three results.

In the end, the query returns three results rather than zero, as in Example 2: Denied properties on all labels, because the security model can still allow results from nodes that do not have the `:Person` label, even if they match on the denied `surname` property.

Fail-open `DENY` behavior

A `DENY` rule fails open when its criteria are not met, so Neo4j does not apply the restriction, and it grants

access by default if a broader `GRANT` exists. This can lead to unintended data exposure if the `DENY` rule is not carefully crafted. To avoid this, apply the principle of least privilege and grant access only to the specific data the user needs to see.

For example, consider the following scenarios:

An unmet `DENY` failing open with property-based RBAC

You grant a user access to a property and try to restrict it with a `DENY` rule. However, if the `DENY` rule does not match any data, for example, if the property is null or misspelled, the `DENY` rule will not apply, and the user can still access the property.

Let's take a few examples:

Example 109. Grant read access to the `salary` property on all nodes with the `Employee` label, but deny it for employees whose position is 'CEO':

```
GRANT READ {salary} ON GRAPH * NODES Employee TO myRole;
DENY READ {salary} ON GRAPH * FOR (e:Employee) WHERE e.position = 'CEO' TO myRole;
```

In this case, if the `e.position` property is null or misspelled, the `DENY` rule will not apply, and `myRole` will see the `salary` property.

A better way is to apply the principle of least privilege and only grant access to the `salary` property for employees whose position is not 'CEO':

Example 110. Grant read access to the `salary` property on all nodes with the `Employee` label, whose position is not 'CEO':

```
GRANT READ {salary} ON GRAPH * FOR (e:Employee) WHERE e.position <> 'CEO' TO myRole;
```

In this way, if `e.position` is null or misspelled, the user will not see the `salary` property, and the `DENY` will not be needed.

Or, if for some reason using `DENY` is unavoidable, the problem can be mitigated by adding an additional `DENY` to cover the case where `e.position` is null:

Example 111. Deny read access to the `salary` property for employees whose position is null:

```
DENY READ {salary} ON GRAPH * FOR (e:Employee) WHERE e.position IS NULL TO myRole;
```

This way, if `e.position` is null, the user will not see the `salary` property, and the `DENY` will not apply.

Alternatively, you can add a constraint to ensure that the `e.position` property cannot be null, so the `DENY` condition is always checkable:

Example 112. Add a constraint to ensure that the `position` property cannot be null:

```
CREATE CONSTRAINT FOR (e:Employee) REQUIRE e.position IS NOT NULL;
```

This way, the `DENY` will never apply due to null values, and the user will not see the `salary` property for employees whose position is 'CEO'.

An unmet `DENY` failing open with label-based RBAC

A `DENY` rule does not apply when it is too broad and does not match the data.

For example, if you grant read access to the `salary` property on all nodes and then try to restrict it with a `DENY` rule for nodes with a specific label, if the nodes with that label are not present in the graph, the `DENY` rule will not apply, and the user can still access the `salary` property on all nodes.

Example 113. Grant read access to the `salary` property on all nodes:

```
GRANT READ {salary} ON GRAPH * NODES * TO myRole;
```

This grants read access to the `salary` property on all nodes, including those that should not be accessible.

Then, you try to restrict it with a `DENY` rule to prevent access to the `salary` property on nodes labeled `Management`:

Example 114. Deny read access to the `salary` property for nodes with the `Management` label:

```
DENY READ {salary} ON GRAPH * NODES Management TO myRole;
```

In this case, if the `Management` label is not present on a node that has the `salary` property, the `DENY` rule does not apply, and `myRole` can still see the `salary` property on that node.

A better way is to apply the principle of least privilege and only grant access to the `salary` property for nodes that have a specific label, such as `IndividualContributor`:

Example 115. Grant read access to the `salary` property only for nodes with the `IndividualContributor` label:

```
GRANT READ {salary} ON GRAPH * NODES IndividualContributor TO myRole;
```

This way, the user only sees the `salary` property on nodes with the `IndividualContributor` label, not on any other nodes.

Access control and labels

In Neo4j, nodes can have multiple labels, but relationships only have one type. This is important when it comes to controlling who can see what.

The following scenarios only focus on nodes because they can have multiple labels. The same general rules apply to relationships, but they are simpler.

For details on the general influence of access control privileges on graph traversal, see Graph and sub-graph access control.

Traversing multi-labeled nodes with partial access to labels

To get information about labels attached to a node by calling the built-in `labels()` function, a user needs to have the privilege to traverse on one or multiple labels attached to that node using `GRANT TRAVERSE` or `GRANT MATCH`. In the case of nodes with multiple labels, the user will be able to see all labels attached to the node, even if they were not granted access to traverse on some of those labels.

For example, let's assume that a graph contains three nodes: one labeled `:A`, another labeled `:B`, and one with both labels `:A` and `:B`.

1. Grant traversal access to nodes with the label `:A` to the `custom` role:

```
GRANT TRAVERSE ON GRAPH * NODES A TO custom;
```

2. Then, run one of the following queries and see the results:

   Run a query to get all labels of nodes with the `:A` label:

```
MATCH (n:A)
RETURN n, labels(n)
```

   The query returns a result with two nodes: the node with label `:A` and the node with labels `:A :B`.

   In contrast, if you run a query to get the labels of nodes with the `:B` label:

```
MATCH (n:B)
RETURN n, labels(n)
```

   The query returns only the node that has both labels: `:A` and `:B`. Even though `:B` does not have access to traversals, there is one node with that label accessible in the dataset due to the allow-listed label `:A` that is attached to the same node.

Traversing multi-labeled nodes with denied access to some of the labels

If a user is denied access to a label, they will never get results from any node with that label. Thus, the

label name will never appear to them.

For example, let's assume that a graph contains three nodes: one labeled `:A`, another labeled `:B`, and one with both labels `:A` and `:B` and that the user has access to traverse on label `:A` but not on label `:B`.

1. Grant traversal access to nodes with the label `:A` to the `custom` role, but deny traversal access to nodes with the label `:B`:

```
GRANT TRAVERSE ON GRAPH * NODES A TO custom;
DENY TRAVERSE ON GRAPH * NODES B TO custom;
```

2. Then, run one of the following queries and see the results:

   Run a query to get all labels of nodes with the `:A` label:

   ```
   MATCH (n:A)
   RETURN n, labels(n)
   ```

   The query returns the node that only has the `:A` label. It does not return the node with both labels `:A` and `:B`, because the user does not have access to traverse on it because of `:B`, even though it also has the label `:A` that is allow-listed.

   Run a query to get the labels of nodes with the `:B` label:

   ```
   MATCH (n:B)
   RETURN n, labels(n)
   ```

   The query returns no nodes.

The db.labels() procedure

In contrast to the normal graph traversal described in the previous section, the built-in `db.labels()` procedure does not process the data graph itself, but the security rules defined on the system graph. That means:

- If a label is explicitly whitelisted (granted), it will be returned by this procedure.
- If a label is denied or is not explicitly allowed, it will not be returned by this procedure.

For example, let's assume that a graph contains three nodes: one labeled `:A`, another labeled `:B`, and one with both labels `:A` and `:B`.

1. Grant traversal access to nodes with the label `:A` to the `custom` role:

```
GRANT TRAVERSE ON GRAPH * NODES A TO custom;
```

2. Then, run the following query:

```
CALL db.labels()
```

   The query returns only label `:A`, because the user does not have access to traverse on label `:B`.

Privileges for nonexistent labels, relationship types, and property names

Privileges for nonexistent labels, relationship types, and property names take effect only after they are created. In other words, when authorizing a user, only privileges for existing labels, relationship types, and property names are applied. This is because the graph elements must be resolved internally to check against privileges when users later try to use them. If a label, relationship type, or property name does not yet exist, it will not resolve, and therefore, the privileges will not apply.

A workaround is to create the label, relationship type, or property name using the `db.createLabel()`, `db.createRelationshipType()`, and `db.createProperty()` procedures in the relevant database when creating the privileges.

Labels, relationship types, and property names are considered nonexistent in a database if:

- There has never been a node with that label, a relationship with that relationship type, or a property with that name. Nor any index or constraint on them.
- There has been no attempt to add a node with that label, a relationship with that relationship type, or a property with that name. An attempted creation adds the label, relationship type, or property name to the known labels, relationship types, and property names, even if the operation ultimately fails, unless it fails due to missing or denied privileges to create new labels, relationship types, or property names.
- There has been no attempt to create an index or constraint using that label, relationship type, or property name. Although many index or constraint errors are detected before the label, relationship type, or property name is created, it may still be created in some cases (for example, if a property existence constraint fails because existing data violates it).
- They have not been created using any of the `db.createLabel()`, `db.createRelationshipType()`, or `db.createProperty()` procedures.

There is currently no way to remove a label, relationship type, or property name from the database. Once existent in the database, they cannot return to nonexistent.

For example, let's assume you have a new, empty database called `testing`.

> The example focuses only on nodes and their labels, though the same principle applies to relationships and their relationship type, and properties (on both nodes and relationships) and their names.

1. Define some privileges to the `custom` role that include a nonexistent label `:A`:

```
GRANT MATCH {*} ON GRAPH testing NODES * TO custom;
GRANT CREATE ON GRAPH testing NODES A TO custom;
GRANT SET LABEL A ON GRAPH testing TO custom;
GRANT CREATE NEW NODE LABEL ON DATABASE testing TO custom;
```

2. Then, connect as a user with the `custom` role, for example, `Alice`, and run the following query to create a node with the nonexistent label `:A`:

   ```
   CREATE (:A)
   ```

   You get the following exception even though you are allowed to create new labels:

   ```
   Create node with labels 'A' on database 'testing' is not allowed for user 'Alice' with roles [PUBLIC,
   custom].
   ```

> **Important:**
>
> However, rerunning the same query creates the node. This is because the failed creation still creates the label, making it no longer nonexistent when the query is run a second time.

To ensure success on the first attempt, when setting up the privileges for the `custom` role, the administrator should run the `db.createLabel()` procedure on the affected databases for all nonexistent labels that get assigned privileges. In this example, when creating the custom role, connect to `testing` and run `CALL db.createLabel('A')` to ensure you create the node successfully on your first attempt.

Access control and performance

Neo4j's role-based access control can have performance implications, especially when security rules are complex or when querying large graphs with restrictive access. For example, count store operations, which are usually fast lookups, may experience significant performance differences because the database must check each node or relationship against the security rules rather than simply retrieving counts from the count store.

Access control and performance of database operations

When security rules that restrict access to certain nodes, relationships, or properties are in place, Neo4j must perform additional checks to ensure that the user only sees what they are allowed to see.

For example, consider a graph with nodes labeled `:Person` and `:Customer`, where some nodes have both labels.

1. Define two roles with different privileges:
   - `restricted` — with some restrictions on traversals.
   - `unrestricted` — with no restrictions on traversals.

   ```
   GRANT TRAVERSE ON GRAPH * NODES Person TO restricted;
   DENY TRAVERSE ON GRAPH * NODES Customer TO restricted;
   GRANT TRAVERSE ON GRAPH * ELEMENTS * TO unrestricted;
   ```

   The `restricted` role allows traversing nodes with the label `:Person`, but denies traversing nodes with the label `:Customer`, while the `unrestricted` role allows traversing all nodes and relationships.

2. When you run the following query, the execution plan looks the same regardless of the role:

Cypher query to count nodes with the `:Person` label

```
MATCH (n:Person)
RETURN count(n)
```

Execution plan for both users

```
+--------------------------+
| Operator                 |
+--------------------------+
| +ProduceResults          |
| |                        +
| +NodeCountFromCountStore |
+--------------------------+
```

However, the underlying operations can differ significantly due to the security rules applied to each role:

User with `unrestricted` role	User with `restricted` role
The database accesses the count store to retrieve the total number of nodes with the label `:Person`, without needing to check each node against any security rules. This is a very quick operation.	The database cannot access the count store because it must make sure that only traversable nodes with the desired label `:Person` are counted. It needs to check each node with the `:Person` label to see whether it also has the `:Customer` label, and deny access to it if it does. Therefore, due to the additional data access required by the security checks, this operation is slower compared to executing the query as an unrestricted user.

Access control and performance of property rules

Extra node or relationship-level security checks are necessary when adding security rules based on property rules, and these can have a significant performance impact.

The following example shows how the database behaves when adding security rules for nodes to roles `restricted` and `unrestricted`. The same limitations apply to relationships.

1. Define two roles with different privileges:

```
GRANT TRAVERSE ON GRAPH * FOR (n:Customer) WHERE n.secret <> true TO restricted;
GRANT TRAVERSE ON GRAPH * ELEMENTS * TO unrestricted;
```

2. When you run the following query, the execution plan looks the same regardless of the role:

Cypher query to get all nodes with the `:Customer` label

```
MATCH (n:Customer)
RETURN n
```

Execution plan for users with either role

```
+--------------------------+
| Operator                 |
+--------------------------+
| +ProduceResults          |
| |                        +
```

```
| +AllNodesScan            |
+--------------------------+
```

However, the underlying operations can differ significantly due to the security rules applied to each role:

User with `unrestricted` role	User with `restricted` role
The database scans all nodes and quickly identifies accessible nodes based solely on the presence of the `:Customer` label. This is a relatively quick operation.	The database scans all nodes, identifies potentially accessible nodes based on the presence of the specified label, then accesses the properties of those nodes and inspects their values to ensure the property rule criteria are met (i.e., that `secret` is not set to `true` in this case). Due to the additional data access required by the security checks, this operation is slower than executing the query as an unrestricted user.

Procedure and user-defined function privileges

To be able to run a procedure or user-defined function, the user needs to have the corresponding execute privilege. Procedures and user-defined functions are executed according to the same security rules as regular Cypher statements, e.g. a procedure performing writes will fail if called by a user that only has *read* privileges.

Procedures and user-defined functions can also be run with privileges exceeding the users' own privileges. This is called *execution boosting*. The elevated privileges only apply within the procedure or user-defined function; any operation performed outside will still use the users' original privileges.

The steps below assume that the procedure or user-defined function is already developed and installed.

Please refer to Java Reference → Extending Neo4j for a description of creating and using user-defined procedures and functions.

Manage procedure permissions

Procedure permissions can be managed using the native execute privileges. These control whether the user is allowed to execute a procedure and which set of privileges apply during the execution.

A procedure may be run using the `EXECUTE PROCEDURE` privilege.

This allows the user to execute procedures that match the globbed procedures.

Example 116. Grant privilege to execute a procedure

```
GRANT EXECUTE PROCEDURE db.schema.visualization ON DBMS TO visualizer
```

This will allow any user with the `visualizer` role to execute the `db.schema.visualization`. E.g. a user that also has the following privileges:

```
GRANT TRAVERSE ON GRAPH * NODES A, B TO role
GRANT TRAVERSE ON GRAPH * RELATIONSHIP R1 TO role
```

When calling the `db.schema.visualization` procedure, the user will only see the `A` and `B` nodes and `R1` relationships, even though there might exist other nodes and relationships.

A procedure may also be executed with elevated privileges using the `EXECUTE BOOSTED PROCEDURE` privilege.

The `EXECUTE BOOSTED PROCEDURE` privilege only controls the privileges used during the execution and not the execution itself. The user needs both `EXECUTE PROCEDURE` and `EXECUTE BOOSTED PROCEDURE` to execute the procedure with elevated privileges.

This allows the user to successfully execute procedures that would otherwise fail during execution with their assigned roles. The user is given full privileges for the procedure, during the execution of the procedure only.

Example 117. Grant privilege to use elevated privileges during procedure execution

```
GRANT EXECUTE BOOSTED PROCEDURE db.schema.visualization ON DBMS TO visualizer
```

This will allow any user with the `visualizer` role to execute the `db.schema.visualization` with elevated privileges. When calling the `db.schema.visualization` procedure, the user will see all nodes and relationships that exist in the graph, even though they have no traversal privileges.

Manage user-defined function permissions

User-defined function permissions can be managed using the native execute privileges. These control if the user is both allowed to execute a user-defined function and which set of privileges apply during the execution.

A user-defined function may be executed using the `EXECUTE USER DEFINED FUNCTION` privilege.

This allows the user to execute user-defined functions that match the globbed user-defined function.

Example 118. Grant privilege to execute a user-defined function

```
GRANT EXECUTE USER DEFINED FUNCTION apoc.any.properties ON DBMS TO custom
```

This will allow any user with the `custom` role to execute the `apoc.any.properties`. E.g. a user that also has the following privilege:

```
GRANT MATCH {visibleProp} ON GRAPH * NODES A TO role
```

When calling the user-defined function `MATCH (a:A) RETURN apoc.any.properties(a) AS`

`properties`, they will only see the `visibleProp` even though there might exist other properties.

A user-defined function may also be executed with elevated privileges using the `EXECUTE BOOSTED USER DEFINED FUNCTION` privilege.

> The `EXECUTE BOOSTED USER DEFINED FUNCTION` privilege only controls the privileges used during the execution and not the execution itself. The user needs both `EXECUTE USER DEFINED FUNCTION` and `EXECUTE BOOSTED USER DEFINED FUNCTION` to execute the user-defined function with elevated privileges.

This allows the user to successfully execute user-defined functions that would otherwise fail during execution with their assigned roles. The user is given full privileges for the user-defined function, during the execution of the function only.

Example 119. Grant privilege to use elevated privileges during user-defined function execution

```
GRANT EXECUTE BOOSTED USER DEFINED FUNCTION apoc.any.properties ON DBMS TO custom
```

This will allow any user with the `custom` role to execute the `apoc.any.properties` with elevated privileges. E.g. a user that also has the following privileges:

```
GRANT TRAVERSE ON GRAPH * NODES A TO role
```

When calling the user-defined function `MATCH (a:A) RETURN apoc.any.properties(a) AS properties`, they will see all properties that exist on the matched nodes even though they have no read privileges.

Built-in roles and privileges

Enterprise Edition | AuraDB Business Critical | AuraDB Virtual Dedicated Cloud

Introduction

Neo4j provides a set of built-in roles that can be used to control access to the database. The `PUBLIC` role is the default role for all users. It does not give any rights or capabilities regarding the data, not even read privileges. The rest of the built-in roles are hierarchical, with the `reader` role at the bottom and the `admin` role at the top with all privileges.

A user may have more than one assigned role, and the union of these determines what action(s) on the data may be undertaken by the user. For instance, a user assigned to the `reader` role can execute procedures, because all users are also assigned to the `PUBLIC` role, which enables that capability.

The built-in roles have the following default privileges:

`PUBLIC`

- Access to the home database.
- Execute procedures with the users' own privileges.
- Execute user-defined functions with the users' own privileges.
- Load data.

`reader`

- Access to all databases.
- Traverse and read on the data graph (all nodes, relationships, properties).
- Show indexes and constraints along with any other future schema constructs.

`editor`

- Access to all databases.
- Traverse, read, and write on the data graph.
- Write access, limited to creating and changing **existing** property keys, node labels, and relationship types of the graph. In other words, the `editor` role cannot add to the schema but can only make changes to already existing objects.
- Show indexes and constraints along with any other future schema constructs.

`publisher`

- Access to all databases.
- Traverse, read, and write on the data graph.
- Show indexes and constraints along with any other future schema constructs.

`architect`

- Access to all databases.
- Traverse, read, and write on the data graph.
- Create/drop/show indexes and constraints along with any other future schema constructs.

`admin`

- Access to all databases.
- Traverse, read, and write on the data graph.
- Load data.
- Create/drop/show indexes and constraints along with any other future schema constructs.
- Execute procedures using boosted privileges.
- Execute admin procedures.
- Execute user-defined functions using boosted privileges.
- View/terminate queries.
- Manage databases, users, roles, and privileges.

When an administrator suspends or deletes another user, the following rules apply:

- Administrators can suspend or delete any other user (including other administrators), but not themselves.
- When suspended, the user is no longer able to log back in until re-activated by an administrator.
- There is no need to remove assigned roles from a user before deleting the user.

> Deleting a user does not automatically terminate associated connections, sessions, transactions, or queries.

Neo4j provides the following built-in roles with default privileges and capabilities. The subset of the functionality that is available with Community Edition is also included. All of the commands require that the user executing the commands has the rights to do so.

Table 567. Built-in roles capabilities

Action	`reader`	`editor`	`publisher`	`architect`	`admin`	`PUBLIC`	Available in Community Edition
Change own password	✓	✓	✓	✓	✓	✓	✓
View own details	✓	✓	✓	✓	✓	✓	✓
View own transactions	✓	✓	✓	✓	✓	✓	✓
Terminate own transactions	✓	✓	✓	✓	✓	✓	✓
View own privileges	✓	✓	✓	✓	✓	✓	
View all databases	✓	✓	✓	✓	✓	✓	✓
Access home database	✓	✓	✓	✓	✓	✓	✓
Access all databases	✓	✓	✓	✓	✓		✓
Read data	✓	✓	✓	✓	✓		✓
View index/constraint	✓	✓	✓	✓	✓		✓
Write/update/delete existing data		✓	✓	✓	✓		✓

Action	`reader`	`editor`	`publisher`	`architect`	`admin`	`PUBLIC`	Available in Community Edition
Create new types of properties key			✓	✓	✓		✓
Create new types of nodes labels			✓	✓	✓		✓
Create new types of relationship types			✓	✓	✓		✓
Create/drop index/constraint				✓	✓		✓
Create/delete user					✓		✓
Change another user's name					✓		✓
Change another user's password					✓		✓
Change another user's home database					✓		
Suspend/activate user					✓		
Create/drop roles					✓		
Change role names					✓		
Assign/remove role to/from user					✓		
Create/drop/alter databases					✓		
Start/stop databases					✓		
Manage database access					✓		
Grant/deny/revoke privileges					✓		

Action	`reader`	`editor`	`publisher`	`architect`	`admin`	`PUBLIC`	Available in Community Edition
View all users					✔		✔
View all roles					✔		
View all roles for a user					✔		
View all users for a role					✔		
View another user's privileges					✔		
View all transactions					✔		✔
Terminate all transactions					✔		✔
Load data					✔	✔	✔
Execute procedures					✔	✔	✔
Execute functions					✔	✔	✔
Execute admin procedures					✔		✔
Dynamically change configuration [1]					✔		

The `PUBLIC` role

All users are granted the `PUBLIC` role, and it can not be revoked or dropped. By default, it gives access to the default database and allows loading data, executing all procedures and user-defined functions.

Important:

The `PUBLIC` role cannot be dropped or revoked from any user, but the specific privileges for the role can be modified. In contrast to the `PUBLIC` role, the other built-in roles can be granted, revoked, dropped, and re-created.

Listing `PUBLIC` role privileges

```
SHOW ROLE PUBLIC PRIVILEGES AS COMMANDS
```

Table 568. Result

command
"GRANT ACCESS ON HOME DATABASE TO `PUBLIC`"
"GRANT EXECUTE FUNCTION * ON DBMS TO `PUBLIC`"
"GRANT EXECUTE PROCEDURE * ON DBMS TO `PUBLIC`"
"GRANT LOAD ON ALL DATA TO `PUBLIC`"
Rows: 4

Recreating the `PUBLIC` role

The `PUBLIC` role can not be dropped and thus there is no need to recreate the role itself. To restore the role to its original capabilities, two steps are needed.

First, all `GRANT` or `DENY` privileges on this role should be revoked (see output of `SHOW ROLE PUBLIC PRIVILEGES AS REVOKE COMMANDS` on what to revoke). Secondly, run these queries:

```
GRANT ACCESS ON HOME DATABASE TO PUBLIC
```

```
GRANT EXECUTE PROCEDURES * ON DBMS TO PUBLIC
```

```
GRANT EXECUTE USER DEFINED FUNCTIONS * ON DBMS TO PUBLIC
```

```
GRANT LOAD ON ALL DATA TO PUBLIC
```

The resulting `PUBLIC` role now has the same privileges as the original built-in `PUBLIC` role.

The `reader` role

The `reader` role can perform read-only queries on all graphs except for the `system` database.

Listing `reader` role privileges

```
SHOW ROLE reader PRIVILEGES AS COMMANDS
```

Table 569. Result

command
"GRANT ACCESS ON DATABASE * TO `reader`"
"GRANT MATCH {*} ON GRAPH * NODE * TO `reader`"

command
"GRANT MATCH {*} ON GRAPH * RELATIONSHIP * TO `reader`"
"GRANT SHOW CONSTRAINT ON DATABASE * TO `reader`"
"GRANT SHOW INDEX ON DATABASE * TO `reader`"
Rows: 5

Recreating the `reader` role

To restore the role to its original capabilities two steps are needed. First, execute `DROP ROLE reader`. Secondly, run these queries:

```
CREATE ROLE reader
```

```
GRANT ACCESS ON DATABASE * TO reader
```

```
GRANT MATCH {*} ON GRAPH * TO reader
```

```
GRANT SHOW CONSTRAINT ON DATABASE * TO reader
```

```
GRANT SHOW INDEX ON DATABASE * TO reader
```

The resulting `reader` role now has the same privileges as the original built-in `reader` role.

The `editor` role

The `editor` role can perform read and write operations on all graphs except for the `system` database, but it cannot create new labels, property keys or relationship types.

Listing `editor` role privileges

```
SHOW ROLE editor PRIVILEGES AS COMMANDS
```

Table 570. Result

command
"GRANT ACCESS ON DATABASE * TO `editor`"
"GRANT MATCH {*} ON GRAPH * NODE * TO `editor`"
"GRANT MATCH {*} ON GRAPH * RELATIONSHIP * TO `editor`"
"GRANT SHOW CONSTRAINT ON DATABASE * TO `editor`"
"GRANT SHOW INDEX ON DATABASE * TO `editor`"

command
"GRANT WRITE ON GRAPH * TO `editor`"
Rows: 6

Recreating the `editor` role

To restore the role to its original capabilities two steps are needed. First, execute `DROP ROLE editor`. Secondly, run these queries:

```
CREATE ROLE editor
```

```
GRANT ACCESS ON DATABASE * TO editor
```

```
GRANT MATCH {*} ON GRAPH * TO editor
```

```
GRANT WRITE ON GRAPH * TO editor
```

```
GRANT SHOW CONSTRAINT ON DATABASE * TO editor
```

```
GRANT SHOW INDEX ON DATABASE * TO editor
```

The resulting `editor` role now has the same privileges as the original built-in `editor` role.

The `publisher` role

The `publisher` role can do the same as `editor`, as well as create new labels, property keys and relationship types.

Listing `publisher` role privileges

```
SHOW ROLE publisher PRIVILEGES AS COMMANDS
```

Table 571. Result

command
"GRANT ACCESS ON DATABASE * TO `publisher`"
"GRANT MATCH {*} ON GRAPH * NODE * TO `publisher`"
"GRANT MATCH {*} ON GRAPH * RELATIONSHIP * TO `publisher`"
"GRANT NAME MANAGEMENT ON DATABASE * TO `publisher`"
"GRANT SHOW CONSTRAINT ON DATABASE * TO `publisher`"

command
"GRANT SHOW INDEX ON DATABASE * TO `publisher`"
"GRANT WRITE ON GRAPH * TO `publisher`"
Rows: 7

Recreating the `publisher` role

To restore the role to its original capabilities two steps are needed. First, execute `DROP ROLE publisher`. Secondly, run these queries:

```
CREATE ROLE publisher
```

```
GRANT ACCESS ON DATABASE * TO publisher
```

```
GRANT MATCH {*} ON GRAPH * TO publisher
```

```
GRANT WRITE ON GRAPH * TO publisher
```

```
GRANT NAME MANAGEMENT ON DATABASE * TO publisher
```

```
GRANT SHOW CONSTRAINT ON DATABASE * TO publisher
```

```
GRANT SHOW INDEX ON DATABASE * TO publisher
```

The resulting `publisher` role now has the same privileges as the original built-in `publisher` role.

The `architect` role

The `architect` role can do the same as the `publisher`, as well as create and manage indexes and constraints.

Listing `architect` role privileges

```
SHOW ROLE architect PRIVILEGES AS COMMANDS
```

Table 572. Result

command
"GRANT ACCESS ON DATABASE * TO `architect`"
"GRANT CONSTRAINT MANAGEMENT ON DATABASE * TO `architect`"
"GRANT INDEX MANAGEMENT ON DATABASE * TO `architect`"

command
"GRANT MATCH {*} ON GRAPH * NODE * TO `architect`"
"GRANT MATCH {*} ON GRAPH * RELATIONSHIP * TO `architect`"
"GRANT NAME MANAGEMENT ON DATABASE * TO `architect`"
"GRANT SHOW CONSTRAINT ON DATABASE * TO `architect`"
"GRANT SHOW INDEX ON DATABASE * TO `architect`"
"GRANT WRITE ON GRAPH * TO `architect`"
Rows: 9

Recreating the `architect` role

To restore the role to its original capabilities two steps are needed. First, execute `DROP ROLE architect`. Secondly, run these queries:

```
CREATE ROLE architect
```

```
GRANT ACCESS ON DATABASE * TO architect
```

```
GRANT MATCH {*} ON GRAPH * TO architect
```

```
GRANT WRITE ON GRAPH * TO architect
```

```
GRANT NAME MANAGEMENT ON DATABASE * TO architect
```

```
GRANT SHOW CONSTRAINT ON DATABASE * TO architect
```

```
GRANT CONSTRAINT MANAGEMENT ON DATABASE * TO architect
```

```
GRANT SHOW INDEX ON DATABASE * TO architect
```

```
GRANT INDEX MANAGEMENT ON DATABASE * TO architect
```

The resulting `architect` role now has the same privileges as the original built-in `architect` role.

The `admin` role

The `admin` role can do the same as the `architect`, as well as manage databases, aliases, users, roles and privileges.

The `admin` role can perform administrative tasks. These include the rights to perform the following classes

of tasks:

- Manage database privileges to control the rights to perform actions on specific databases:
  - Manage access to a database and the right to start and stop a database.
  - Manage constraints.
  - Allow the creation of labels, relationship types, or property names.
  - Manage transactions.
- Manage DBMS privileges to control the rights to perform actions on the entire system:
  - Manage multiple databases.
  - Manage users and roles.
  - Change configuration parameters.
  - Manage sub-graph privileges.
  - Manage procedure security.
  - Manage load privileges to control the rights to load data from external sources.

These rights are conferred using privileges that can be managed through the `GRANT`, `DENY` and `REVOKE` commands.

Listing `admin` role privileges

```
SHOW ROLE admin PRIVILEGES AS COMMANDS
```

Table 573. Result

command
"GRANT ACCESS ON DATABASE * TO `admin`"
"GRANT ALL DBMS PRIVILEGES ON DBMS TO `admin`"
"GRANT CONSTRAINT MANAGEMENT ON DATABASE * TO `admin`"
"GRANT INDEX MANAGEMENT ON DATABASE * TO `admin`"
"GRANT LOAD ON ALL DATA TO `admin`"
"GRANT MATCH {*} ON GRAPH * NODE * TO `admin`"
"GRANT MATCH {*} ON GRAPH * RELATIONSHIP * TO `admin`"
"GRANT NAME MANAGEMENT ON DATABASE * TO `admin`"
"GRANT SHOW CONSTRAINT ON DATABASE * TO `admin`"
"GRANT SHOW INDEX ON DATABASE * TO `admin`"
"GRANT START ON DATABASE * TO `admin`"
"GRANT STOP ON DATABASE * TO `admin`"
"GRANT TRANSACTION MANAGEMENT (*) ON DATABASE * TO `admin`"

command
"GRANT WRITE ON GRAPH * TO `admin`"
Rows: 14

If the built-in `admin` role has been altered or dropped and needs to be restored to its original state, see Password and user recovery.

Recreating the `admin` role

To restore the role to its original capabilities two steps are needed. First, execute `DROP ROLE admin`. Secondly, run these queries:

```
CREATE ROLE admin
```

```
GRANT ALL DBMS PRIVILEGES ON DBMS TO admin
```

```
GRANT TRANSACTION MANAGEMENT ON DATABASE * TO admin
```

```
GRANT START ON DATABASE * TO admin
```

```
GRANT STOP ON DATABASE * TO admin
```

```
GRANT MATCH {*} ON GRAPH * TO admin
```

```
GRANT WRITE ON GRAPH * TO admin
```

```
GRANT LOAD ON ALL DATA TO admin
```

```
GRANT ALL ON DATABASE * TO admin
```

The resulting `admin` role now has the same effective privileges as the original built-in `admin` role.

Additional information about restoring the `admin` role can be found in the Recover the admin role.

Immutable roles and privileges

Immutable privileges are useful for restricting the actions of users who can themselves administer privileges. Starting with Neo4j 5.26, Neo4j also introduces immutable roles. Immutable roles are useful for providing *system roles*, which appear as permanent parts of the DBMS.

Caution:

Immutable privileges and roles should only be used in situations where changes are rare. They are intentionally difficult to modify, so changes should be undertaken with caution (e.g., when the DBMS has been isolated by some other means and unauthorized access can be reliably prevented). Typically, this type of modification should only be made once during the commissioning phase of a DBMS.

Administer immutable roles and privileges

After the DBMS is safely isolated from external connections, follow these steps to administer immutable roles and privileges:

1. Change the config setting `dbms.security.auth_enabled` to `false`.
2. Restart the DBMS.
3. Create or remove immutable privileges and roles in the same way as regular privileges and roles but with the addition of the `IMMUTABLE` keyword. See Examples.
4. Change the config setting `dbms.security.auth_enabled` back to `true`.
5. Restart the DBMS.

Privileges and roles created in this way now appear as an immutable part of the DBMS. If you want to change or remove them, you must repeat the process of setting `dbms.security.auth_enabled` to `false`.

Examples

The following examples demonstrate how to use Cypher to manage immutable roles and privileges.

Restricting the actions of users who can manage privileges

To prevent all users (including those with `PRIVILEGE MANAGEMENT` privileges) from performing **database management**, attach an immutable privilege to the `PUBLIC` role. The `PUBLIC` role implicitly and irrevocably applies to all users.

1. Ensure that you have completed steps 1 and 2 from Administer immutable roles and privileges.
2. Run the following command to deny the `IMMUTABLE DATABASE MANAGEMENT` privilege to the `PUBLIC` role:

```
DENY IMMUTABLE DATABASE MANAGEMENT ON DBMS TO PUBLIC
```

3. Verify that the `IMMUTABLE` keyword has been added to the privilege:

```
SHOW PRIVILEGES WHERE IMMUTABLE
```

Table 574. Result

access	action	resource	graph	segment	role	immutable
"DENIED"	"database_management"	"database"	"*"	"database"	"PUBLIC"	true

access	action	resource	graph	segment	role	immutable
Rows: 2						

The result shows that all users are restricted from adding or removing privileges, including the `admin` user.

4. Ensure you have completed steps 4 and 5 from Administer immutable roles and privileges.

Creating permanent roles that cannot be changed

You can use immutable roles to create permanent built-in system roles that cannot be modified even by users who have `ROLE MANAGEMENT` privileges.

For example, you want to create an `analyst` role that cannot be dropped, renamed, or have any of its privileges changed (even by users with the `ROLE MANAGEMENT` and `PRIVILEGE MANAGEMENT` privileges).

1. Ensure that you have completed steps 1 and 2 from Administer immutable roles and privileges.
2. Create an immutable role to hold the immutable privileges:

```
CREATE IMMUTABLE ROLE analyst
```

3. Immutably grant the `MATCH` privilege:

```
GRANT IMMUTABLE MATCH {*} ON GRAPH * ELEMENTS * TO analyst
```

4. Ensure you have completed steps 4 and 5 from Administer immutable roles and privileges.

   Now, even users with `ROLE MANAGEMENT` and `PRIVILEGE MANAGEMENT` privileges will not be able do any of the following:

   Drop the `analyst` role

```
DROP ROLE analyst
```

   Revoke the `MATCH` privilege from the `analyst` role

```
REVOKE MATCH {*} ON GRAPH * ELEMENTS * FROM analyst
```

   Rename the `analyst` role

```
RENAME ROLE analyst TO dataReader
```

> **Note:**
>
> While the make-up (name, existence, associated privileges) of immutable roles is immutable, their assignment to users is not. This means that an immutable role can itself be granted to or revoked from a user by any user with `ROLE MANAGEMENT` privileges.

Note:

Only immutable privileges (e.g. `GRANT IMMUTABLE MATCH {*} ON GRAPH * ELEMENTS * TO analyst` in the example above) can be assigned to immutable roles. This is to make sure that an immutable role and all of its privileges is explicitly and completely immutable.

[1] For more information, see Update dynamic settings

Integration with auth systems

User auth providers

Authentication and authorization can be controlled on a user level using Cypher by setting auth providers on users.

To use auth providers, you need to set the `dbms.security.require_local_user` configuration setting to `true`. This setting mandates that users with the relevant auth provider attached to them must exist in the database before they can authenticate and authorize with that auth provider.

User auth providers allow you to link externally-defined users (e.g., in a third-party ID provider like OIDC or LDAP) to the Neo4j internal user model. The internal model can define roles (authorization), `SUSPENDED` status, `HOME DATABASE`, and metadata such as the unique displayed name of the user. For consistency, you can also define `native` (password-based) auth using the auth provider syntax, including native-only users (i.e., users who can only authenticate with a password).

Use cases

User auth providers can be used for a variety of use cases, including:

- Provisioning different auth providers (including native username/password auth) for different users.
- Setting an arbitrary easy username for a user while using an external unique identifier (like `sub` for OIDC auths, which itself is not a user-friendly value).
- Setting `HOME DATABASE` for externally authenticated users.
- Setting `SUSPENDED` status for an externally authenticated user.
- Using native authorization to manage roles for externally authenticated users.
- Retaining full control of which users can authenticate from within the database.

How it works

When a user authenticates, their identifying attributes are checked against the relevant property of the auth providers in the database. If there is a match, then the user is linked to the Neo4j user and authorized according to the DBMS security configuration settings that match the name of the matching auth provider.

How the matching lookup is done depends on the type of provider. For example:

- For an OIDC provider, the claim configured by `dbms.security.oidc.mysso.claims.username` (default `sub`) is taken from the token and is used to look up an auth provider whose `ID` and `provider` properties match the `sub` and provider respectively of the OIDC provider.
- For an LDAP provider, the `dn` is used to look up an auth provider with a `provider` property of `ldap` and an `ID` property that matches the supplied `dn`.
- For the `native` (username/password) provider, the supplied username itself is used to look up the auth provider.

Enabling user auth providers mode

To enable user auth providers mode, set the configuration setting `dbms.security.require_local_user` to `true`. This setting mandates that users with the relevant auth provider attached to them must exist in the database before they can authenticate and authorize with that auth provider.

When the user authenticates, Neo4j searches for a user with a matching authentication provider. If a match is found, the user can log in and be authorized successfully.

Migrating to auth providers mode

If you have existing users in the database and want to migrate to auth providers mode, you can use the `ALTER USER … SET AUTH` command to attach an auth provider to each of them. Until you change `dbms.security.require_local_user` to `true`, this will not impact the users' ability to authenticate and authorize as they always have done.

Once the process of adding auth providers to your users finishes, you can set `dbms.security.require_local_user` to `true` and restart the DBMS to complete the migration. After this time, only users with a corresponding auth provider in the database will be able to authenticate and authorize.

Note:

Existing users created using the original `CREATE USER … SET PASSWORD` command implicitly have the native (username/password) auth provider, so you do not need to add it explicitly using `SET AUTH`.

To verify which auth providers are attached to a user, use the `SHOW USERS WITH AUTH` command.

Examples

For examples of how to use auth providers with different authentication providers, see the following sections:

- Configure SSO at the user level using auth providers
- Creating users
- Configure authentication/authorization at the user level using LDAP as an auth provider

LDAP integration

Neo4j supports LDAP, which allows for integration with Active Directory (AD), OpenLDAP, or other LDAP-compatible authentication services. This means that you use the LDAP service for managing federated users, while the native Neo4j user and role administration are completely turned off.

The following configuration settings are important to consider when configuring LDAP. For a more detailed overview of the LDAP configuration options, see Configuration settings.

LDAP dynamic configuration settings

The following configuration settings can be updated while the database is running, see Update dynamic settings. Altering any of these settings clears the authentication and authorization cache.

Parameter name	Default value	Description
dbms.security.ldap.authentication.user_dn_template	`uid={0},ou=users,dc=example,dc=com`	Convert usernames into LDAP-specific fully qualified names required for logging in.
dbms.security.ldap.authorization.user_search_base	`ou=users,dc=example,dc=com`	Set the base object or named context to search for user objects.
dbms.security.ldap.authorization.user_search_filter	`(&(objectClass=*)(uid={0}))`	Set an LDAP search filter for a user principal.
dbms.security.ldap.authorization.group_membership_attributes	`memberOf`	List attribute names of a user object that contains groups to be used for mapping to roles. Common values: `memberOf` and `gidNumber`.
dbms.security.ldap.authorization.nested_groups_enabled	`false`	This setting determines whether multiple LDAP search results will be processed. This must be set to `true` in order to resolve nested group membership.
dbms.security.ldap.authorization.group_to_role_mapping		List an authorization mapping from groups to the pre-defined built-in roles `admin`, `architect`, `publisher`, `editor`, and `reader`, or to any other custom-defined roles.
dbms.security.ldap.authentication.attribute	`samaccountname`	Set the attribute to search for users with a system account.
dbms.security.ldap.authorization.access_permitted_group		Set an LDAP group of users with access rights. Users passing authentication are mapped to at least the `PUBLIC` role in addition to any roles assigned by the group to role mapping and have access to the database that those roles provide. If this attribute is set, users not part of this LDAP group will fail authentication, even if their credentials are correct.

Parameter name	Default value	Description
dbms.security.logs.ldap.groups_at_debug_level_enabled	`false`	When set to `true`, it logs the result from the group lookup into the security log (provided the security log level is also set to `DEBUG`).

All settings are defined at server startup time in the default configuration file *neo4j.conf* or can be modified at runtime using `dbms.setConfigValue()`.

Set Neo4j to use LDAP

First, you configure Neo4j to use LDAP as an authentication and authorization provider.

1. Uncomment the setting `dbms.security.auth_enabled=false` and change its value to `true` to turn on the security feature.
2. Uncomment the settings `dbms.security.authentication_providers` and `dbms.security.authorization_providers` and change their value to `ldap`. This way, the LDAP connector is used as a security provider for both authentication and authorization.

   If you want, you can still use the `native` provider for mixed-mode authentication and authorization. The values are comma-separated and queried in the declared order.

   Example 120. Configure Neo4j to use LDAP and the native authentication and authorization provider

   ```
   dbms.security.authentication_providers=ldap,native
   dbms.security.authorization_providers=ldap,native
   ```

Map the LDAP groups to the Neo4j roles

To assign privileges to users based on their LDAP groups, you have to map the LDAP groups to the Neo4j built-in and custom-defined roles. To do that, you need to know what privileges the Neo4j roles have, and based on these privileges, to create the mapping to the groups defined in the LDAP server. The map must be formatted as a semicolon separated list of key-value pairs, where the key is a LDAP group name and the value is a comma-separated list of the corresponding role names.

For example, `group1=role1;group2=role2;group3=role3,role4,role5;group4=role6;group5=role6`.

Example 121. Example of LDAP groups to Neo4j roles mapping

```
dbms.security.ldap.authorization.group_to_role_mapping=\
    "cn=Neo4j Read Only,cn=users,dc=example,dc=com"      = reader;    \
    "cn=Neo4j Read-Write,cn=users,dc=example,dc=com"     = editor,publisher; \
    "cn=Neo4j Read-Write,cn=users,dc=example,dc=com","cn=Neo4j Create
Data,cn=users,dc=example,dc=com"     = publisher; \
    "cn=Neo4j Create Data,cn=users,dc=example,dc=com","cn=Neo4j Schema
Manager,cn=users,dc=example,dc=com" = architect; \
    "cn=Neo4j Administrator,cn=users,dc=example,dc=com"  = admin; \
    "cn=Neo4j Procedures,cn=users,dc=neo4j,dc=com"       = rolename
```

Mapping of an LDAP group to a Neo4j built-in role.

Mapping of an LDAP group to two Neo4j built-in roles.

Mapping of two LDAP groups to a Neo4j built-in role.

Mapping of an LDAP group to a custom-defined role. Custom-defined roles, such as `rolename`, must be explicitly created using the `CREATE ROLE rolename` command before they can be used to grant privileges. See Manage roles.

Configure Neo4j to use Active Directory

You configure Neo4j to use the LDAP security provider to access and manage your Active Directory. There are three alternative ways to do that depending on your specific use case.

Configure Neo4j to support LDAP user ID authentication

This option allows users to log in with their LDAP user ID.

In the *neo4j.conf* file, uncomment and configure the following settings:

1. Configure LDAP to point to the AD server:

```
dbms.security.ldap.host=ldap://myactivedirectory.example.com
```

2. Provide details on the user structure of the LDAP directory:

```
dbms.security.ldap.authentication.user_dn_template=cn={0},cn=Users,dc=example,dc=com
dbms.security.ldap.authorization.user_search_base=cn=Users,dc=example,dc=com
dbms.security.ldap.authorization.user_search_filter=(&(objectClass=*)(cn={0}))
dbms.security.ldap.authorization.group_membership_attributes=memberOf
```

3. Map the groups in the LDAP system to the Neo4j built-in and custom roles. See Map the LDAP groups to the Neo4j roles.

Configure Neo4j to support attribute authentication

This is an alternative configuration for Active Directory that allows users to log in by providing an attribute to search for, by default `sAMAccountName`. The attribute has to be unique to be used as a lookup. You create a system account that has read-only access to the parts of the LDAP directory that you want. However, it does not need to have access rights to Neo4j or any other systems.

In the *neo4j.conf* file, uncomment and configure the following settings:

1. Configure LDAP to point to the AD server:

```
dbms.security.ldap.host=ldap://myactivedirectory.example.com
```

2. Provide details on the user structure of the LDAP directory (replacing `myattribute` with the actual attribute name):

```
dbms.security.ldap.authorization.user_search_base=cn=Users,dc=example,dc=com
dbms.security.ldap.authorization.user_search_filter=(&(objectClass=*)(myattribute={0}))
dbms.security.ldap.authorization.group_membership_attributes=memberOf
```

3. Map the groups in the LDAP system to the Neo4j built-in and custom roles. See Map the LDAP groups to the Neo4j roles.
4. Configure Neo4j to use a system account with read access to all users and groups in the LDAP server.
   a. Set `dbms.security.ldap.authorization.use_system_account` value to `true`.
   b. Set `dbms.security.ldap.authorization.system_username` value to the full Distinguished Name (DN) as the `dbms.security.ldap.authentication.user_dn_template` will not be applied to this username. For example,

   ```
   dbms.security.ldap.authorization.system_username=cn=search-account,cn=Users,dc=example,dc=com
   ```

   c. Configure the LDAP system account password.

   ```
   dbms.security.ldap.authorization.system_password=your_password
   ```

   d. Configure which attribute to search for by adding the following lines to the *neo4j.conf* file (replacing `myattribute` with the actual attribute name):

   ```
   dbms.security.ldap.authentication.search_for_attribute=true
   dbms.security.ldap.authentication.attribute=myattribute
   ```

   e. (Optional) Create an LDAP group to restrict authentication against the database to a subset of LDAP users:

   ```
   dbms.security.ldap.authorization.access_permitted_group=cn=Neo4j Access,cn=users,dc=example,dc=com
   ```

Configure Neo4j to support `sAMAccountName` authentication by setting `user_dn_template`

This is an alternative configuration for Active Directory that allows all users from the specified domain to log in using `sAMAccountName`. With this option, you do not have to create a system account and store a system username/password in the config file. Instead, you set `{0}@example.com` as a value of the `user_dn_template` to enable the authentication to start at the root domain. This way, the whole tree is checked to find the user, regardless of where it is located within the LDAP directory tree.

In the *neo4j.conf* file, uncomment and configure the following settings:

1. Configure LDAP to point to the AD server:

```
dbms.security.ldap.host=ldap://myactivedirectory.example.com
```

2. Provide details on the user structure of the LDAP directory:

```
dbms.security.ldap.authentication.user_dn_template={0}@example.com
dbms.security.ldap.authorization.user_search_base=dc=example,dc=com
```

```
dbms.security.ldap.authorization.user_search_filter=(&(objectClass=user)(sAMAccountName={0}))
dbms.security.ldap.authorization.group_membership_attributes=memberOf
```

3. Map the groups in the LDAP system to the Neo4j built-in and custom roles. For more information, see Map the LDAP groups to the Neo4j roles.

> The setting `dbms.security.ldap.authentication.search_for_attribute` should be set to the default value of false.

Configure Neo4j to perform nested group lookup

When a user is a member of a group (e.g. `engineers`) and that group is a member of another group (e.g. `employees`), Active Directory can be configured to perform a nested search such that a user in the group `engineers` would also be a member of the group `employees`. This in turn means that it is possible to configure a group to role mapping for `employees` which will transitively apply to `engineers`.

Active Directory facilitates nested search via the extensible match operator `LDAP_MATCHING_RULE_IN_CHAIN` (whose Object Identifier is 1.2.840.113556.1.4.1941). This operator walks the chain of ancestry in objects all the way to the root.

To set up nested search in the *neo4j.conf* file, configure the following settings:

1. Enable nested groups.

```
dbms.security.ldap.authorization.nested_groups_enabled=true
```

2. Provide details on the user structure of the LDAP directory:

```
dbms.security.ldap.authentication.user_dn_template=cn={0},cn=users,dc=example,dc=com
dbms.security.ldap.authorization.user_search_base=dc=example,dc=com
dbms.security.ldap.authorization.user_search_filter=(&(objectClass=*)(uid={0}))
```

3. Provide the nested groups search filter.
   This is the filter which will be used to perform the nested lookup of the user's groups. It should contain the placeholder token `\{0}`, which will be substituted with the user's Distinguished Name (which is found for the specified user principle using `dbms.security.ldap.authorization.user_search_filter`).
   This example features Active Directory's `LDAP_MATCHING_RULE_IN_CHAIN` (aka `1.2.840.113556.1.4.1941`) implementation:

```
dbms.security.ldap.authorization.nested_groups_search_filter=(&(objectclass=group)(member:1.2.840.113556.1.4.1941:={0}))
```

4. Provide group to role mappings, including ancestor groups if required:

```
dbms.security.ldap.authorization.group_to_role_mapping=\
"cn=engineers,cn=users,dc=example,dc=com"=procedures;\
"cn=employees,cn=users,dc=example,dc=com"=reader
```

> In contrast to a non-nested-LDAP lookup, a nested group lookup does not perform an

attribute-based lookup on the user object. Instead, the `dbms.security.ldap.authorization.group_membership_attributes` setting is ignored and the `dbms.security.ldap.authorization.user_search_filter` is only used to determine the Distinguished Name of the user. This is then substituted into the `dbms.security.ldap.authorization.nested_groups_search_filter` to perform a separate, nested lookup of the user's groups.

Configure Neo4j to use OpenLDAP

You configure the LDAP security provider to access and manage your OpenLDAP directory service.

In the neo4j.conf file, uncomment and configure the following settings:

1. Configure LDAP to point to the OpenLDAP server:

```
dbms.security.ldap.host=myopenldap.example.com
```

2. Provide details on the user structure of the LDAP directory:

```
dbms.security.ldap.authentication.user_dn_template=cn={0},ou=users,dc=example,dc=com
dbms.security.ldap.authorization.user_search_base=ou=users,dc=example,dc=com
dbms.security.ldap.authorization.user_search_filter=(&(objectClass=*)(uid={0}))
dbms.security.ldap.authorization.group_membership_attributes=gidNumber
```

3. (Optional) Create an LDAP group to restrict authentication against the database to a subset of LDAP users:

```
dbms.security.ldap.authorization.access_permitted_group=501
```

4. Map the groups in the LDAP system to the Neo4j built-in and custom roles. For more information, see Map the LDAP groups to the Neo4j roles.

Configure authentication/authorization at the user level using auth providers

User auth providers can be used to determine which users can authenticate and authorize using the configured providers, including LDAP.

Introduced in 5.24

You must change the `dbms.security.require_local_user` configuration setting to `true` to use auth providers. This means that a user with a matching auth provider **must** exist in order to be able to authenticate and authorize. This applies to all providers.

Conversely, when `dbms.security.require_local_user` is set to `false`, users' auth providers have no bearing on the way that they are authenticated and authorized, instead authentication and authorization is controlled centrally (for all users) by the database configuration.

The following examples show how to configure users with auth provider `ldap` using Cypher.

Example 122. Create a user with an auth provider who can authenticate and authorize using `LDAP`

```
CREATE USER alice
SET AUTH PROVIDER 'ldap' { SET ID 'cn=alice,ou=engineering,dc=example,dc=com' }
```

The command creates the user `alice` who can authenticate and authorize using LDAP provided their LDAP `dn` is `cn=alice,ou=engineering,dc=example,dc=com`.

Example 123. Create a user with two auth providers allowing the user to authenticate and authorize with either LDAP or the `mysso` provider

```
CREATE USER alice
SET HOME DATABASE anotherDb
SET AUTH PROVIDER 'ldap' { SET ID 'cn=alice,ou=engineering,dc=example,dc=com' }
SET AUTH 'oidc-mysso' {SET ID 'alicesUniqueMySsoId'}
```

The command creates the user `alice` who can authenticate and authorize using `ldap` or `mysso`. See Configure SSO at the user level using auth providers for more information on setting up an OIDC provider. The example also illustrates that the user can have their home database set even when using only external auth providers.

Example 124. Alter a user to remove one of their auth providers

```
ALTER USER alice
REMOVE AUTH 'ldap'
```

The command prevents the user `alice` from being able to authenticate and authorize using `ldap`.

Example 125. Alter a user to allow them to authenticate and authorize using username and password

```
ALTER USER alice
SET AUTH 'native' {SET PASSWORD 'changeme' SET PASSWORD CHANGE REQUIRED}
```

The command allows the user `alice` to authenticate and authorize using the specified username and password (in addition to what they are already configured to use).

Example 126. Configure the database to allow authentication via `ldap` and authorization via the `native` provider

1. Set the following database config:

   ```
   dbms.security.authentication_providers=ldap
   dbms.security.authorization_providers=native
   ```

2. Create a user with an `ldap` auth provider:

```
CREATE USER alice
SET AUTH PROVIDER 'ldap' { SET ID 'cn=alice,ou=engineering,dc=example,dc=com' }
```

3. Natively grant the `READER` role to the user:

```
GRANT ROLE READER TO alice
```

The command allows the user `alice` to authenticate using `ldap` and receive the `READER` role from the `native` provider.

4. You can also give the user the union of roles from `ldap` **and** `native` roles by setting `ldap` as an authorization provider too:

```
dbms.security.authentication_providers=ldap
dbms.security.authorization_providers=native,ldap
```

Example 127. Suspend a user

```
ALTER USER alice
SET STATUS SUSPENDED
```

The command completely prevents the user from being able to authenticate/authorize by any means.

Example 128. Disambiguate users with the same name in different LDAP trees

Suppose there are two users both with the name `alice`, one is part of the `engineering` tree (`cn=alice,ou=engineering,dc=example,dc=com`) and the other is part of the `sales` tree (`cn=alice,ou=sales,dc=example,dc=com`).

To disambiguate these users, you can create two users in the database, each with a different `ID` that corresponds to the `dn` of the user in the LDAP tree.

```
CREATE USER aliceEngineering
SET AUTH 'ldap' { SET ID 'cn=alice,ou=engineering,dc=example,dc=com' }

CREATE USER aliceSales
SET AUTH 'ldap' { SET ID 'cn=alice,ou=sales,dc=example,dc=com' }
```

Verify the LDAP configuration

You can verify that your LDAP configuration is correct, and that the LDAP server responds, by using the LDAP command-line tool `ldapsearch`.

The `ldapsearch` command accepts the LDAP configuration setting values as input and verifies both the authentication (using the `simple` mechanism) and authorization of a user. See the ldapsearch official documentation for more advanced usage and how to use SASL authentication mechanisms.

1. Verify the authentication and authorization of a user. For example, `john`.
   - With `dbms.security.ldap.authorization.use_system_account=false` (default):

```
# ldapsearch -v -H ldap://<dbms.security.ldap.host> -x -D
<dbms.security.ldap.authentication.user_dn_template : replace {0}> -W -b
<dbms.security.ldap.authorization.user_search_base>
"<dbms.security.ldap.authorization.user_search_filter : replace {0}>"
<dbms.security.ldap.authorization.group_membership_attributes>

ldapsearch -v -H ldap://myactivedirectory.example.com:389 -x -D cn=john,cn=Users,dc=example,dc=com
-W -b cn=Users,dc=example,dc=com "(&(objectClass=*)(cn=john))" memberOf
```

   - With `dbms.security.ldap.authorization.use_system_account=true`:

```
# ldapsearch -v -H ldap://<dbms.security.ldap.host> -x -D
<dbms.security.ldap.authorization.system_username> -w
<dbms.security.ldap.authorization.system_password> -b
<dbms.security.ldap.authorization.user_search_base>
"<dbms.security.ldap.authorization.user_search_filter>"
<dbms.security.ldap.authorization.group_membership_attributes>

ldapsearch -v -H ldap://myactivedirectory.example.com:389 -x -D cn=search-account,cn=Users,dc
=example,dc=com -w your_password -b cn=Users,dc=example,dc=com "(&(objectClass=*)(cn=john))"
memberOf
```

2. Verify that the value of the returned membership attribute is a group that is mapped to a role in `dbms.security.ldap.authorization.group_to_role_mapping`.

```
# extended LDIF
#
# LDAPv3
# base <cn=Users,dc=example,dc=com> with scope subtree
# filter: (cn=john)
# requesting: memberOf
#

# john, Users, example.com
dn: CN=john,CN=Users,DC=example,DC=com
memberOf: CN=Neo4j Read Only,CN=Users,DC=example,DC=com

# search result
search: 2
result: 0 Success

# numResponses: 2
# numEntries: 1
```

The auth cache

The *auth cache* is the mechanism by which Neo4j caches the result of authentication via the LDAP server in order to aid performance. It is configured with the parameters `dbms.security.ldap.authentication.cache_enabled`, and `dbms.security.auth_cache_ttl`.

```
# Turn on authentication caching to ensure performance.

dbms.security.ldap.authentication.cache_enabled=true
dbms.security.auth_cache_ttl=10m
```

Table 575. Auth cache parameters

Parameter name	Default value	Description
dbms.security.ldap.authentication.cache_enabled	`true`	Determines whether or not to cache the result of authentication via the LDAP server. Whether authentication caching should be enabled or not must be considered in view of your company's security guidelines.
dbms.security.auth_cache_ttl	`600 seconds`	Is the time to live (TTL) for cached authentication and authorization info. Setting the TTL to 0 disables all auth caching. A short TTL requires more frequent re-authentication and re-authorization, which can impact performance. A very long TTL means that changes to the users settings on an LDAP server may not be reflected in the Neo4j authorization behaviour in a timely manner. Valid units are `ms`, `s`, `m`; default unit is `s`.

An administrator can clear the auth cache to force the re-querying of authentication and authorization information from the federated auth provider system. Use Neo4j Browser or Neo4j Cypher Shell to execute this statement:

```
CALL dbms.security.clearAuthCache()
```

Available methods of encryption

Specifying the `dbms.security.ldap.host` parameter configures using LDAP without encryption. Not specifying the protocol or port results in `ldap` being used over the default port `389`.

```
dbms.security.ldap.host=myactivedirectory.example.com
dbms.security.ldap.host=myactivedirectory.example.com:389
dbms.security.ldap.host=ldap://myactivedirectory.example.com
dbms.security.ldap.host=ldap://myactivedirectory.example.com:389
```

Use LDAP with encryption via StartTLS

To configure Active Directory with encryption via StartTLS, set the following parameters:

```
dbms.security.ldap.use_starttls=true
dbms.security.ldap.host=ldap://myactivedirectory.example.com
```

Use LDAP with encrypted LDAPS

To configure Active Directory with encrypted LDAPS, set `dbms.security.ldap.host` to one of the following. If you do not specify the port, the default one `636` is used.

```
dbms.security.ldap.host=ldaps://myactivedirectory.example.com
dbms.security.ldap.host=ldaps://myactivedirectory.example.com:636
```

Use a self-signed certificate (SSL) in a test environment

Production environments should always use an SSL certificate issued by a Certificate Authority for secure access to the LDAP server. However, there are scenarios, for example in test environments, where you may want to use an SSL certificate on the LDAP server.

To configure an SSL certificate on LDAP server, enter the details of the certificate using `server.jvm.additional` in *neo4j.conf*. The path to the certificate file `MyCert.jks` is an absolute path to the Neo4j server.

```
server.jvm.additional=-Djavax.net.ssl.keyStore=/path/to/MyCert.jks
server.jvm.additional=-Djavax.net.ssl.keyStorePassword=mypasword
server.jvm.additional=-Djavax.net.ssl.trustStore=/path/to/MyCert.jks
server.jvm.additional=-Djavax.net.ssl.trustStorePassword=mypasword
```

Debug logging of group result

While setting up an LDAP integration, it is sometimes necessary to perform troubleshooting. In these cases, it can be useful to view the group result from the LDAP server. To enable the logging of these claims at `DEBUG` level in the security log, set dbms.security.logs.ldap.groups_at_debug_level_enabled to be `true` and the security log level to `DEBUG`.

> **Warning:**
>
> Make sure to set dbms.security.logs.ldap.groups_at_debug_level_enabled back to `false` for production environments to avoid unwanted logging of potentially sensitive information. Also, bear in mind that the group result provided by the LDAP server can change over time.

Single sign-on integration

Neo4j supports OpenID Connect (OIDC), which allows for integration with many identity providers including Okta, Microsoft Entra ID, and Google. This integration permits federated users, managed by the identity provider, to access Neo4j instead of, or in addition to the native users and roles. For examples with different providers and troubleshooting, see the SSO configuration tutorial.

OIDC configuration settings

Neo4j supports multiple OIDC identity providers at the same time, as such each provider configuration must be assigned a prefix to differentiate it from others. In the configuration examples below the provider-specific prefix is represented by `<provider>`, which should be replaced with a name representing your provider. For example, if you are using Okta as your identity provider you might use `okta` in the place of `<provider>` below.

The following configuration settings are important to consider when configuring single sign-on. For a more detailed overview of the single sign-on configuration options, see Configuration settings. Some of these settings can also be updated while the database is running, see Dynamic settings for more information on how to do this. Altering any of these settings causes users to re-authenticate as their permissions may have changed as a result.

Parameter name	Default value	Dynamic	Description
dbms.security.oidc.<provider>.display_name		false	The display name for the provider. This is displayed in clients such as Neo4j Browser and Bloom.
dbms.security.oidc.<provider>.auth_flow	pkce	true	The OIDC auth_flow for clients such as Neo4j Browser and Bloom to use. Supported values are `pkce` and `implicit`.
dbms.security.oidc.<provider>.well_known_discovery_uri		true	The OpenID Connect Discovery URL for the provider.
dbms.security.oidc.<provider>.auth_endpoint		true	URL of the provider's Authorization Endpoint.
dbms.security.oidc.<provider>.auth_params		true	Optional parameters that clients may require with the Authorization Endpoint. The map is a semicolon-separated list of key-value pairs. For example: k1=v1;k2=v2.
dbms.security.oidc.<provider>.token_endpoint		true	URL of the provider's OAuth 2.0 Token Endpoint.
dbms.security.oidc.<provider>.token_params		true	Option parameters that clients may require with the Token Endpoint. The map is a semicolon-separated list of key-value pairs. For example: k1=v1;k2=v2.
dbms.security.oidc.<provider>.jwks_uri		true	URL of the provider's JSON Web Key Set.
dbms.security.oidc.<provider>.user_info_uri		true	URL of the provider's UserInfo Endpoint.
dbms.security.oidc.<provider>.issuer		true	URL that the provider asserts as its issuer identifier. This will be checked against the `iss` claim in the token.
dbms.security.oidc.<provider>.audience		true	The expected value for the `aud` claim.

Parameter name	Default value	Dynamic	Description
dbms.security.oidc.<provider>.params		true	Option parameters that clients may require. The map is a semicolon-separated list of key-value pairs. For example: k1=v1;k2=v2.
dbms.security.oidc.<provider>.config		true	Option additional configuration that clients may require. The map is a semicolon-separated list of key-value pairs. For example: k1=v1;k2=v2.
dbms.security.oidc.<provider>.get_groups_from_user_info	false	true	Whether to fetch the groups claim from the user info endpoint on the identity provider. The default is `false`, to read the claim from the token.
dbms.security.oidc.<provider>.get_username_from_user_info	false	true	Whether to fetch the username claim from the user info endpoint on the identity provider. The default is `false`, to read the claim from the token.
dbms.security.oidc.<provider>.claims.username	sub	true	The claim to use for the database username. Neo4j expects to find a string claim in the JWT or user_info response with this name.
dbms.security.oidc.<provider>.claims.groups		true	The claim to use for the database roles. Neo4j expects to find a claim in the JWT or user_info response with this name. The claim may be a string claim representing a single role or a string array claim representing multiple roles. From Neo4j 5.4, the JWT claim may also contain a single group returned as a string as well as a list of groups as was previously required.
dbms.security.oidc.<provider>.authorization.group_to_role_mapping		true	List an authorization mapping from groups to the pre-defined built-in roles `admin`, `architect`, `publisher`, `editor`, and `reader`, or to any custom-defined roles.
dbms.security.logs.oidc.jwt_claims_at_debug_level_enabled	false	false	When set to `true`, it logs the claims from the JWT into the security log (provided the security log level is also set to `DEBUG`).

Configure Neo4j to use OpenID Connect

First, you configure Neo4j to use OpenID Connect as an authentication and authorization provider in the *neo4j.conf* file.

1. Make sure security is turned on. The default value for `dbms.security.auth_enabled` is `true`.
2. Uncomment the settings `dbms.security.authentication_providers` and `dbms.security.authorization_providers` and change their value to `oidc-<provider>`, where `<provider>` maps to the provider name used in the configuration settings. This way, the OIDC connector is used as a security provider for both authentication and authorization. If you want, you can still use the `native` provider for mixed-mode authentication and authorization. The values are comma-separated and queried in the declared order.

   Example 129. Configure Neo4j to use two OpenID Connect and the native authentication and authorization providers.

   ```
   dbms.security.authentication_providers=oidc-newsso,oidc-oldsso,native
   dbms.security.authorization_providers=oidc-newsso,oidc-oldsso,native
   ```

3. Check connectivity. Neo4j needs to connect to the identity provider to discover settings and fetch public keys to verify tokens. Check firewall settings and security controls, and, if necessary, logs to ensure that the Neo4j server is able to connect to the identity provider using HTTPS. If a proxy is required, this can be configured in the Java virtual machine using the configuration setting server.jvm.additional. Proxies that require credentials are not supported.

Map the identity provider groups to the Neo4j roles

Before identity provider managed groups can be used with Neo4j, you have to decide on an approach for mapping identity provider groups to Neo4j roles. The simplest approach is to create identity provider groups with the same names as Neo4j roles. If you decide to go this way, no mapping configuration is necessary. Assuming, however, that identity provider groups do not directly map 1:1 to the desired Neo4j roles, it is necessary to map the identity provider groups to the Neo4j built-in and custom-defined roles. To do that, you need to know what privileges the Neo4j roles have, and based on these privileges, create the mapping to the groups defined in the identity provider. The map must be formatted as a semicolon-separated list of key-value pairs, where the key is the identity provider group name and the value is a comma-separated list of the corresponding role names. For example, `group1=role1;group2=role2;group3=role3,role4,role5;group4=role6;group5=role6`.

Example 130. Example of identity provider groups to Neo4j roles mapping

```
dbms.security.oidc.mysso.authorization.group_to_role_mapping=\
    neo4j_readonly  = reader;    \
    neo4j_rw        = editor,publisher; \
    neo4j_rw        = publisher; \
    neo4j_create    = publisher; \
    neo4j_dba       = admin; \
    neo4j_exec      = rolename
```

Mapping of an identity provider group to a Neo4j built-in role.

Mapping of an identity provider group to two Neo4j built-in roles.

Mapping of two identity provider groups to a Neo4j built-in role.

Mapping of an identity provider group to a custom-defined role. Custom-defined roles, such as

`rolename`, must be explicitly created using the `CREATE ROLE rolename` command before they can be used to grant privileges. See Manage roles.

> **Note:**
>
> When specifying explicit group to role mapping the automatic mapping for groups and roles sharing a name is disabled. This means that all groups and roles need to be specified to be mapped, even if they share a name.

Configure Neo4j to use an OpenID Connect identity provider

This option allows users to log in through an OIDC compliant identity provider by offering a token from the provider instead of a username and password. Typically, these tokens take the form of a signed JSON Web Token (JWT). The following configuration examples use `mysso` as the provider's name. It is recommended to use a name describing the provider that is being integrated.

OpenID Connect using JWT claims

In this configuration, Neo4j receives a JWT from the identity provider containing claims representing the database username (e.g. email), and the Neo4j roles.

1. Set a display name.

   In the *neo4j.conf* file, uncomment and configure the following settings:

   ```
   dbms.security.oidc.mysso.display_name=SSO Provider
   ```

   This is displayed on a button on the login page of clients, such as Neo4j Browser and Bloom so that you can identify the provider you are using to login.

2. Configure discovery.

   Uncomment and configure the following settings:

   ```
   dbms.security.oidc.mysso.well_known_discovery_uri=https://my-idp.example.com/.well-known/openid-configuration
   ```

   The `well_known_discovery` endpoint of the identity provider supplies the OpenID provider metadata to allow Neo4j to interact with that provider. It is also possible to configure the provider settings manually:

   ```
   dbms.security.oidc.mysso.auth_endpoint=https://my-idp.example.com/openid-connect/auth
   dbms.security.oidc.mysso.token_endpoint=https://my-idp.example.com/openid-connect/token
   dbms.security.oidc.mysso.jwks_uri=https://my-idp.example.com/openid-connect/certs
   dbms.security.oidc.mysso.user_info_uri=https://my-idp.example.com/openid-connect/userinfo
   dbms.security.oidc.mysso.issuer=abcd1234
   ```

   Manual settings always take priority over those retrieved from the discovery endpoint.

3. Configure audience.

   Provide the expected value for the audience(`aud`) claim:

   ```
   dbms.security.oidc.mysso.claims.audience=myaudience
   ```

4. Configure claims.

   Provide the name of the claims that map to the database username and roles. `username` is expected to be a string claim, and `roles` is expected to be a list of strings representing a set of roles or a single string representing a single role:

   ```
   dbms.security.oidc.mysso.claims.username=sub
   dbms.security.oidc.mysso.claims.groups=roles
   ```

5. Optionally, map the groups in the OIDC groups claim to the Neo4j built-in and custom roles.

   See Map the identity provider groups to the Neo4j roles

OpenID Connect fetching claims from a provider

In this configuration, Neo4j receives a token from the identity provider and uses that token to call back to the identity provider using its *UserInfo* endpoint to retrieve claims for the database username and Neo4j roles.

1. Configure Neo4j for OpenID Connect Using JWT Claims.
2. Configure the claims to fetch from the *UserInfo* endpoint:

   ```
   dbms.security.oidc.mysso.get_username_from_user_info=true
   dbms.security.oidc.mysso.get_groups_from_user_info=true
   ```

   It is possible to fetch just the username, just the groups, or both from the userinfo endpoint.

Configure SSO at the user level using auth providers

User auth providers can be used to determine which users can authenticate and authorize using the configured providers.

Introduced in 5.24

You must change the `dbms.security.require_local_user` configuration setting to `true` to use auth providers. This means that a user with a matching auth provider **must** exist in order to be able to authenticate and authorize. This applies to all providers.

Conversely, when `dbms.security.require_local_user` is set to `false`, users' auth providers have no bearing on the way that they are authenticated and authorized, instead authentication and authorization is controlled centrally (for all users) by the database configuration.

The following examples show how to configure users with auth provider using Cypher.

Example 131. Create a user with an auth provider who can authenticate and authorize using `mysso`

```
CREATE USER jake
SET AUTH 'oidc-mysso' {SET ID 'jakesUniqueMySsoId'} // the id must match the claim that you
configured via dbms.security.oidc.mysso.claims.username
```

The command creates the user `jake` who can authenticate and authorize using `mysso` provided they present a valid token with a `sub` claim of `jakesUniqueMySsoId`. The claim used for authentication is determined by the `dbms.security.oidc.mysso.claims.username` config setting (the default is the `sub` claim).

Example 132. Create a user with two auth providers allowing the user to authenticate and authorize with one of them

```
CREATE USER jake
SET HOME DATABASE anotherDb
SET AUTH 'oidc-mysso1' {SET ID 'jakesUniqueMySso1Id'} // `jakesUniqueMySso1Id` must match the value
of the claim that you configured via dbms.security.oidc.mysso1.claims.username
SET AUTH 'oidc-mysso2' {SET ID 'jakesUniqueMySso2Id'} // `jakesUniqueMySso2Id` must match the value
of the claim that you configured via dbms.security.oidc.mysso2.claims.username
```

The command creates the user `jake` who can authenticate and authorize using `mysso1` or `mysso2`. The example also illustrates that the user can have their home database set even when using only external auth providers.

Example 133. Alter a user to remove one of their auth providers

```
ALTER USER jake
REMOVE AUTH 'oidc-mysso2'
```

The command prevents the user `jake` from being able to authenticate and authorize with the `mysso2` provider.

Example 134. Alter a user to allow them to authenticate and authorize using username and password

```
ALTER USER jake
SET AUTH 'native' {SET PASSWORD 'changeme' SET PASSWORD CHANGE REQUIRED}
```

The command allows the user `jake` to authenticate and authorize using the specified username and password (in addition to what they are already configured to use).

Example 135. Configure the database to allow authentication via `mysso` and authorization via the `native` provider

1. Set the following database config:

```
dbms.security.authentication_providers=oidc-mysso
```

```
dbms.security.authorization_providers=native
```

2. Create a user with a `mysso` auth provider:

```
CREATE USER jake
SET AUTH 'oidc-mysso' {SET ID 'jakesUniqueMySsoId'} // `jakesUniqueMySsoId` must match the value
of the claim that you configured via dbms.security.oidc.mysso.claims.username
```

3. Natively grant the `READER` role to the user:

```
GRANT ROLE READER TO jake
```

   The command allows the user `jake` to authenticate using `mysso` and receive the `READER` role from the `native` provider.

4. You can also give the user the union of roles from `mysso` **and** `native` by setting `mysso` as an authorization provider too:

```
dbms.security.authentication_providers=oidc-mysso
dbms.security.authorization_providers=native,oidc-mysso
```

Example 136. Suspend a user

```
ALTER USER jake
SET STATUS SUSPENDED
```

The command completely prevents the user from being able to authenticate/authorize by any means.

Use a self-signed certificate (SSL) in a test environment

Production environments should always use an SSL certificate issued by a Certificate Authority for secure access to the identity provider. However, there are scenarios, for example in test environments, where you may want to use a self-signed SSL certificate on the identity provider server.

To configure a self-signed SSL certificate used on an identity provider server, enter the details of a Java keystore containing the relevant certificates using `server.jvm.additional` in *neo4j.conf*. The path to the certificate file `MyCert.jks` is an absolute path to the Neo4j server.

```
server.jvm.additional=-Djavax.net.ssl.keyStore=/path/to/MyCert.jks
server.jvm.additional=-Djavax.net.ssl.keyStorePassword=mypasword
server.jvm.additional=-Djavax.net.ssl.trustStore=/path/to/MyCert.jks
server.jvm.additional=-Djavax.net.ssl.trustStorePassword=mypasword
```

Debug logging of JWT claims

While setting up an OIDC integration, it is sometimes necessary to perform troubleshooting. In these cases, it can be useful to view the claims contained in the JWT supplied by the identity provider.

To enable the logging of these claims at `DEBUG` level in the security log, set dbms.security.logs.oidc.jwt_claims_at_debug_level_enabled to be `true` and the security log level to `DEBUG`. You can do this in *<NEO4J_HOME>/conf/server-logs.xml*.

If you need more information on how to set up and manage the security log, see Configure the security log.

Warning:

Make sure to set dbms.security.logs.oidc.jwt_claims_at_debug_level_enabled back to `false` for production environments to avoid unwanted logging of potentially sensitive information. Also, bear in mind that the set of claims provided by an identity provider in the JWT can change over time.

Security

This section describes how to ensure physical data security according to industry best practices with regards to server and network security:

- Securing extensions
- SSL framework
- Credentials handling in Neo4j Browser
- Security checklist

Additionally, logs can be useful for continuous analysis, or for specific investigations. Facilities are available for producing security event logs as well as query logs as described in Monitoring.

Refer to Authentication and authorization for information on how to manage users and their authentication and authorization.

Securing extensions

Neo4j can be extended by writing custom code which can be invoked directly from Cypher, as described in Java Reference → User-defined functions. This page describes how to ensure the security of these additions.

Allow listing

Allow listing can be used to allow the loading of only a few extensions from a larger library. It is recommended to load extensions using the principle of least privilege. This principle dictates that you only load the procedures and functions necessary to execute your queries.

The configuration setting `dbms.security.procedures.allowlist` is used to name certain procedures and functions that should be available from a library. It defines a comma-separated list of procedures and functions that are to be loaded. The list may contain both fully qualified procedure names, and partial names with the wildcard `*`.

Example 137. Allow listing

In this example, you need to allow the use of the method `apoc.load.json` as well as all the methods under `apoc.coll`. You do not want any additional extensions from the `apoc` library to become available, other than the ones matching these criteria.

```
# Example allow listing
dbms.security.procedures.allowlist=apoc.coll.*,apoc.load.json
```

There are a few things that should be noted about `dbms.security.procedures.allowlist`:

- If using this setting, no extensions other than those listed will be loaded. In particular, if it is set to the

empty string, no extensions will be loaded.

Warning:

- The default of the setting is `*`. This means that if you do not explicitly give it a value (or no value), all libraries in the *plugins* directory will be loaded.

Unrestricting

For security reasons, procedures and functions that use internal APIs are disabled by default. In this case, it is also recommended to use the principle of least privilege and only unrestrict those procedures and functions which you are certain to use.

Procedures and functions can be unrestricted using the configuration setting `dbms.security.procedures.unrestricted`. It defines a comma-separated list of procedures and functions that are to be unrestricted. The list may contain both fully qualified procedure and function names, and partial names with the wildcard (`*`) expression.

Example 138. Unrestricting

In this example, you need to unrestict the use of the procedures `apoc.cypher.runFirstColumn` and `apoc.cypher.doIt`.

```
# Example unrestricting
dbms.security.procedures.unrestricted=apoc.cypher.runFirstColumn,apoc.cypher.doIt
```

SSL framework

The SSL framework provides support for securing the following Neo4j communication channels using standard SSL/TLS technology:

- `bolt` (port - `7687`)
- `https` (port - `7473`)
- `cluster` (ports - `5000`, `6000`, `7000`, and `7688`)
- `backups` (port - `6362`)

This page describes how to set up SSL within your environment, how to view, validate, and test the certificates.

Note:

As of Neo4j 5.23, the use of port `5000` for discovery management and discovery service v1 is deprecated. Neo4j 5.23 introduces the discovery service v2, which now utilizes the port `6000`. For more details, refer to Clustering → Cluster server discovery.

SSL Providers

The secure networking in Neo4j is provided through the Netty library, which supports both the native JDK SSL provider as well as Netty-supported OpenSSL derivatives. Each version of Neo4j ships with a version of Netty, and Netty requires a specific version of the `netty-tcnative` library for compatibility.

Follow these steps to use OpenSSL:

- Install a suitable `netty-tcnative` dependency into the `plugins/` directory of Neo4j.
  - Dependencies can be downloaded from https://netty.io/wiki/forked-tomcat-native.html.
  - Which `netty-tcnative` version you need depends upon the Neo4j version. For versioning details, see the Netty support per Neo4j version table. Make sure to install a build variant that matches your OS and architecture.
- Set `dbms.netty.ssl.provider =OPENSSL`.
- Restart Neo4j.

Note:

Using dynamic versions of tcnative will require installation of platform-specific dependency libraries as described in https://netty.io/wiki/forked-tomcat-native.html.

In most use cases, the statically linked `boringssl` variant of `netty-tcnative` will be sufficient to enable SSL encryption.

The following table shows information about supported Neo4j versions.

The following table lists when the `netty-tcnative` dependency was updated in Neo4j.

Tip:

If a Neo4j version is not listed, use the table entry for the next earliest Neo4j version listed.

For example: for Neo4j 5.15.0 the next earliest version listed in the table is 5.10, so the required `netty-tcnative` version is `2.0.61.Final`.

Starting from Neo4j 5.23, you can find the dynamically linked version of `netty-tcnative` under the */lib* directory.

Table 576. Netty-TCNative support per Neo4j version

Neo4j version	tcnative version	Direct link
5.23	2.0.65.Final. Only netty-tcnative-boringssl-static is required.	netty-tcnative-boringssl-static-2.0.65.Final
5.20	2.0.65.Final. Both netty-tcnative-boringssl-static and netty-tcnative-classes are required.	netty-tcnative-boringssl-static-2.0.65.Final netty-tcnative-classes-2.0.65.Final

Neo4j version	tcnative version	Direct link
5.10	2.0.61.Final. Both netty-tcnative-boringssl-static and netty-tcnative-classes are required.	netty-tcnative-boringssl-static-2.0.61.Final netty-tcnative-classes-2.0.61.Final
5.8	2.0.60.Final. Both netty-tcnative-boringssl-static and netty-tcnative-classes are required.	netty-tcnative-boringssl-static-2.0.60.Final netty-tcnative-classes-2.0.60.Final
5.5	2.0.56.Final. Both netty-tcnative-boringssl-static and netty-tcnative-classes are required.	netty-tcnative-boringssl-static-2.0.56.Final netty-tcnative-classes-2.0.56.Final
5.1	2.0.54.Final. Both netty-tcnative-boringssl-static and netty-tcnative-classes are required.	netty-tcnative-boringssl-static-2.0.54.Final netty-tcnative-classes-2.0.54.Final

Note:

Using OpenSSL can significantly improve performance, especially for AES-GCM-cryptos, e.g. TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256.

Certificates and private keys

Certificates

The SSL configuration requires SSL certificates to be issued by a Certificate Authority (CA). All certificates must follow the X.509 standard and be saved in a `PEM-encoded` file.

Tip:

Valid trusted certificates can be generated for free using non-profit CAs such as Let's Encrypt.

Example public.crt file

```
-----BEGIN CERTIFICATE-----
MIIDojCCAoqgAwIBAgIBATANBgkqhkiG9w0BAQsFADBhMQswCQYDVQQGEwJTRTEQ
     ...
xsUBvcQuyxewlvWRS18YB51J+yu0Xg==
-----END CERTIFICATE-----
```

The instructions on this page assume that you have already obtained the required certificates from the CA and added them to the *public.crt* file. To achieve this, you should concatenate each PEM-encoded certificate, starting from the leaf certificate and moving up the chain toward the root.

Tip:

If the same certificates are used across all instances of the cluster, make sure that when generating the certificates to include the DNS names of all the cluster instances in the

certificates. Multi-host and wildcard certificates are also supported.

Tip:

If setting up intra-cluster encryption as part of a cluster configuration, ensure that the certificates used on the cluster endpoint support server and client usage. This is because when connecting between the Neo4j servers for clustering, each server uses its own certificate to authenticate as a client on the connection to another server.

This could be verified from within the certificate details:

```
openssl x509 -in public.crt -noout -text
```

We should see that the X509v3 Extended Key Usage section shows both the usages listed:

```
X509v3 Extended Key Usage:
    TLS Web Server Authentication, TLS Web Client Authentication
```

Transformations

Neo4j requires all SSL certificates to be in the `PEM` format. If your certificate is in the binary `DER` format, you must transform it into `PEM` format.

Transform `DER` format certificate to `PEM` format

```
openssl x509 -in cert.der -inform der -outform pem -out cert.crt
```

Private keys

Private keys must be in the standard format PKCS #8 and saved as a `PEM-encoded` file.

Example private.key file

```
-----BEGIN PRIVATE KEY-----
MIICdQIBADANBgkqhkiG9w0BAQEFAASCAl8wggJbAgEAAoGBAN5D0I4bgdQK4In6
     ...
oaMe91ZPQ1JI
-----END PRIVATE KEY-----
```

Private keys can also be encrypted with a passphrase according to the PKCS #5 standard.

Example private.key file with passphrase encryption

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
MIICojAcBgoqhkiG9w0BDAEEMA4ECL3eSAoRlJ18AgIIAASCAoCj7WDyjsgcawdv
     ...
lYeSjVah
-----END ENCRYPTED PRIVATE KEY-----
```

Transformation

If the private key is encoded with the old PKCS #1 format, the file will typically start with the line:

```
-----BEGIN RSA PRIVATE KEY-----
```

You can convert it to PKCS #8 format with the following command:

Convert a PKCS #1 key to a PKCS #8 key

```
openssl pkcs8 -topk8 -in pkcs1.key -out pkcs8.key
```

An unencrypted private key could be PKCS #1 or PKCS #8. It can be encrypted with the following command:

Convert an unencrypted key to an encrypted PKCS #8 key using 256bit AES in cipher-block-chaining (CBC) mode

```
openssl pkcs8 -topk8 -v2 aes-256-cbc -v2prf hmacWithSHA512 -in pkcs1or8.key -out pkcs8.encrypted.key
```

Supported encryption arguments to openssl are:

- `-v2 aes-128-cbc -v2prf hmacWithSHA1`
- `-v2 aes-128-cbc -v2prf hmacWithSHA224`
- `-v2 aes-128-cbc -v2prf hmacWithSHA256`
- `-v2 aes-128-cbc -v2prf hmacWithSHA384`
- `-v2 aes-128-cbc -v2prf hmacWithSHA512`
- `-v2 aes-256-cbc -v2prf hmacWithSHA224`
- `-v2 aes-256-cbc -v2prf hmacWithSHA256`
- `-v2 aes-256-cbc -v2prf hmacWithSHA384`
- `-v2 aes-256-cbc -v2prf hmacWithSHA512`

Note:

Versions before Neo4j 5.0 allow keys to be stored with the old PKCS #1 standard. You can identify them by the line `-----BEGIN RSA PRIVATE KEY-----` at the beginning of the file. While Neo4j 5.0 can load and use those keys, they are considered deprecated and will be removed in a future version.

Validate the key and the certificate

If you need, you can validate the key file and the certificate as follows:

Validate the key

```
openssl rsa -in private.key -check
```

Validate certificate in the PEM format

```
openssl x509 -in public.crt -text -noout
```

Connectors

Before enabling SSL support, you must ensure the following connector configurations to avoid errors:

- Set `server.https.enabled` to `true` when using HTTPS.
- Set `server.bolt.tls_level` to `REQUIRED` or `OPTIONAL` when using Bolt.

For more information on configuring connectors, see Configure connectors.

Configuration

The SSL policies are configured by assigning values to parameters of the following format:

`dbms.ssl.policy.<scope>.<setting-suffix>`

- `scope` is the name of the communication channel, such as `bolt`, `https`, `cluster`, and `backup`.
- `setting-suffix` can be any of the following:

> **Note:**
>
> Neo4j does not validate the `setting-suffix` and if it is misspelled or incorrectly set, it will be ignored.

Setting suffix	Description	Default value
	Basic	
`enabled`	Setting this to `true` enables this policy.	`false`
`base_directory`	The base directory under which cryptographic objects are searched for by default.	`certificates/<scope>`
`private_key`	The private key used for authenticating and securing this instance.	`private.key`
`private_key_password`	The passphrase to decode the private key. Only applicable for encrypted private keys.	
`public_certificate`	A public certificate matching the private key signed by a CA.	`public.crt`
`trusted_dir`	A directory populated with certificates of trusted parties.	`trusted/`

Setting suffix	Description	Default value
`revoked_dir`	A directory populated with certificate revocation lists (CRLs).	`revoked/`
	Advanced	
`verify_hostname`	Enabling this setting turns on client-side hostname verification. After receiving the server's public certificate, the client compares the address it uses against the certificate Common Name (CN) and Subject Alternative Names (SAN) fields. If the address does not match those fields, the client disconnects.	`false`
`ciphers`	A comma-separated list of ciphers suites allowed during cipher negotiation. Valid values depend on the current JRE, SSL provider, and TLS version. For Ciphers supported by the Oracle JRE, see the Oracle official documentation.	Java platform default allowed cipher suites.
`tls_versions`	A comma-separated list of allowed TLS versions. By default only TLSv1.2 and TLSv1.3 are allowed. To use both TLSv1.2 and TLSv1.3 versions, you must specify which ciphers to be enforced for each version. Otherwise, Neo4j could use every possible cipher in the JVM for those versions, leading to a less secure configuration.	`TLSv1.2` `TLSv1.3` Supported from Neo4j 5.24
`client_auth`	Whether or not clients must be authenticated. Setting this to `REQUIRE` enables mutual authentication for servers. Other possible values are `NONE` and `OPTIONAL`.	`OPTIONAL` for `bolt` and `https`; `REQUIRE` for `cluster` and `backup`.
`trust_all`	Setting this to `true` results in all clients and servers to be trusted and the content of the `trusted_dir` directory to be ignored. Use this only as a mean of debugging, since it does not offer security.	`false`

> **Note:**
>
> For security reasons, Neo4j does not automatically create any of these directories. Therefore, the creation of an SSL policy requires the appropriate file system structure to be set up manually. Note that the existence of the directories, the certificate file, and the private key are mandatory. Ensure that only the Neo4j user can read the private key.

Each policy needs to be explicitly enabled by setting:

```
dbms.ssl.policy.<scope>.enabled=true
```

Configure SSL over Bolt

Bolt protocol is based on the PackStream serialization and supports the Cypher type system, protocol versioning, authentication, and TLS via certificates. For Neo4j clusters, Bolt provides smart client routing with load balancing and failover. When server side routing is enabled, an additional Bolt port is open on `7688`. It can be used only within the cluster and with all the same settings as the external Bolt port.

Bolt connector is used by Cypher Shell, Neo4j Browser, and by the officially supported language drivers. Bolt connector is enabled by default but its encryption is disabled. To enable the encryption over Bolt, create the folder structure and place the key file and the certificates under those. Then, you need to configure the SSL Bolt policies in the *neo4j.conf* file.

1. Enable the Bolt connector to enable SSL over Bolt:

   ```
   server.bolt.enabled=true (default is true)
   ```

2. Set up the *bolt* folder under *certificates*.

   a. Create a directory *bolt* under *<NEO4J_HOME>/certificates* folder:

   ```
   mkdir certificates/bolt
   ```

   b. Create a directory *trusted* and *revoked* under *<NEO4J_HOME>/certificates/bolt* folder:

   ```
   mkdir certificates/bolt/trusted
   mkdir certificates/bolt/revoked
   ```

3. Place the certificates *private.key* and the *public.crt* files under *<NEO4J_HOME>/certificates/bolt* folder:

   ```
   cp /path/to/certs/private.key certificates/bolt
   cp /path/to/certs/public.crt certificates/bolt
   ```

4. Place the *public.crt* file under the *<NEO4J_HOME>/certificates/bolt/trusted* folder.

   ```
   cp /path/to/certs/public.crt certificates/bolt/trusted
   ```

5. (Optional) If a particular certificate is revoked, then place it under

<*NEO4J_HOME*>/*certificates/bolt/revoked* folder.

```
cp /path/to/certs/public.crt certificates/bolt/revoked
```

The folder structure should look like this with the right file permissions and the groups and ownerships:

Path	Directory/File	Owner	Group	Permission	Unix/Linux View
/data/neo4j/certificates/bolt	Directory	neo4j	neo4j	0755	drwxr-xr-x
/data/neo4j/certificates/bolt/public.crt	File	neo4j	neo4j	0644	-rw-r--r--
/data/neo4j/certificates/bolt/private.key	File	neo4j	neo4j	0400	-r--------
/data/neo4j/certificates/bolt/trusted	Directory	neo4j	neo4j	0755	drwxr-xr-x
/data/neo4j/certificates/bolt/trusted/public.crt	File	neo4j	neo4j	0644	-rw-r--r--
/data/neo4j/certificates/bolt/revoked	Directory	neo4j	neo4j	0755	drwxr-xr-x

Tip:

The owner/group should be configured to the user/group that will be running the `neo4j` service. Default user/group is neo4j/neo4j.

6. Set the Bolt SSL configuration in *neo4j.conf*.
   a. Set the SSL Bolt policy to `true`:

```
dbms.ssl.policy.bolt.enabled=true
```

   b. Set the appropriate certificates path and the right key and cert files:

```
dbms.ssl.policy.bolt.base_directory=certificates/bolt
dbms.ssl.policy.bolt.private_key=private.key
dbms.ssl.policy.bolt.public_certificate=public.crt
```

Tip:

If the certificate is located outside of `NEO4J_HOME`, then set the absolute path for the certificates directory.

   c. Set the Bolt client authentication to `NONE` to disable the mutual authentication:

```
dbms.ssl.policy.bolt.client_auth=NONE
```

d. Set the Bolt TLS level to allow the connector to accept encrypted and/or unencrypted connections:

```
server.bolt.tls_level=REQUIRED (default is DISABLED)
```

> **Tip:**
>
> In Neo4j version 3.5, the default value is `OPTIONAL`. In the Neo4j 4.x versions, the default value is `DISABLED`, where only unencrypted client connections are to be accepted by this connector, and all encrypted connections will be rejected. Use `REQUIRED` when only encrypted client connections are to be accepted by this connector, and all unencrypted connections will be rejected. Use `OPTIONAL` where either encrypted or unencrypted client connections are accepted by this connector.

7. Test the SSL connection to the specified host and Bolt port and view the certificate:

```
openssl s_client -connect my_domain.com:7687
```

Connect with SSL over Bolt

Each of the `neo4j` and `bolt` URI schemes permit variants that contain extra encryption and trust information. The `+s` variants enable encryption with a full certificate check. The `+ssc` variants enable encryption with no certificate check. This latter variant is designed specifically for use with self-signed certificates.

URI Scheme	Routing	Description
`neo4j`	Yes	Unsecured
`neo4j+s`	Yes	Secured with full certificate
`neo4j+ssc`	Yes	Secured with self-signed certificate
`bolt`	No	Unsecured
`bolt+s`	No	Secured with full certificate
`bolt+ssc`	No	Secured with self-signed certificate

Once SSL is enabled over Bolt, you can connect to the Neo4j DBMS using `neo4j+s` or `bolt+s`:

Cypher Shell

```
cypher-shell -a neo4j+s://<Server DNS or IP>:<Bolt port>

or

cypher-shell -a bolt+s://<Server DNS or IP>:<Bolt port>
```

Neo4j Browser

From the **Connect URL** dropdown menu, select the URI scheme you want to use (`neo4j+s` or `bolt+s`).

> **Note:**
>
> URI schemes ending `+ssc` are not supported by Neo4j Browser since the browser's OS handles certificate trust. If it is necessary to connect to a Neo4j instance using a self-signed certificate from Neo4j Browser, first visit a web page that uses the self-signed certificate in order to prompt the browser to request that certificate trust be granted. Once that trust has been granted, you can connect with URI schemes ending `+s`.

Configure SSL over HTTPS

HTTP(s) is used by the Neo4j Browser and the HTTP API. HTTPS (secure HTTP) is set to encrypt network communications. To enable the encryption over HTTPS, create the folder structure and place the key file and the certificates under those. Then, you need to configure the SSL HTTPS policies in the *neo4j.conf* file and disable the HTTP connector.

> **Note:**
>
> The HTTPS configuration requires that Bolt is also set. Refer to Configure SSL over Bolt for more instructions.

1. Enable the HTTPS connector to enable SSL over HTTPS:

```
server.https.enabled=true (default is false)
```

2. Set up the *https* folder under *certificates*.
   a. Create a directory *https* under *<NEO4J_HOME>/certificates* folder:

```
mkdir certificates/https
```

   b. Create a directory *trusted* and *revoked* under *<NEO4J_HOME>/certificates/https* folder:

```
mkdir certificates/https/trusted
mkdir certificates/https/revoked
```

3. Place the certificates *private.key* and the *public.crt* files under *<NEO4J_HOME>/certificates/https* folder:

```
cp /path/to/certs/private.key certificates/https
cp /path/to/certs/public.crt certificates/https
```

4. Place the *public.crt* file under the *<NEO4J_HOME>/certificates/https/trusted* folder.

```
cp /path/to/certs/public.crt certificates/https/trusted
```

5. (Optional) If a particular certificate is revoked, then place it under *<NEO4J_HOME>/certificates/https/revoked* folder.

```
cp /path/to/certs/public.crt certificates/https/revoked
```

The folder structure should look like this with the right file permissions and the groups and ownerships:

Path	Directory/File	Owner	Group	Permission	Unix/Linux View
/data/neo4j/certificates/https	Directory	neo4j	neo4j	0755	drwxr-xr-x
/data/neo4j/certificates/https/public.crt	File	neo4j	neo4j	0644	-rw-r--r--
/data/neo4j/certificates/https/private.key	File	neo4j	neo4j	0400	-r--------
/data/neo4j/certificates/https/trusted	Directory	neo4j	neo4j	0755	drwxr-xr-x
/data/neo4j/certificates/https/trusted/public.crt	File	neo4j	neo4j	0644	-rw-r--r--
/data/neo4j/certificates/https/revoked	Directory	neo4j	neo4j	0755	drwxr-xr-x

Tip:

The owner/group should be configured to the user/group that will be running the `neo4j` service. Default user/group is neo4j/neo4j.

6. Set the HTTPS SSL configuration in *neo4j.conf*.
   a. Set the SSL HTTPS policy to `true`:

```
dbms.ssl.policy.https.enabled=true
```

   b. Set the appropriate certificates path and the right key and cert files:

```
dbms.ssl.policy.https.base_directory=certificates/https
dbms.ssl.policy.https.private_key=private.key
dbms.ssl.policy.https.public_certificate=public.crt
```

Tip:

If the certificate is located outside of `NEO4J_HOME`, then set the absolute path for the certificates directory.

   c. Set the HTTPS client authentication to `NONE` to disable the mutual authentication:

```
dbms.ssl.policy.https.client_auth=NONE
```

   d. Disable HTTP connector:

```
server.http.enabled=false
```

7. Test the SSL connection to the specified host and HTTPS port and view the certificate:

```
openssl s_client -connect my_domain.com:7473
```

Configure SSL for intra-cluster communications

Intra-cluster encryption is the security solution for the cluster communication. The Neo4j cluster communicates on 4 ports:

- `5000` - Discovery management
- `6000` - Transactions
- `7000` - Raft communications
- `7688` - Server side routing

To configure SSL for intra-cluster communication, create the folder structure and place the key and certificate files under it. Then, configure the SSL cluster policies in the *neo4j.conf* file and test that the intra-cluster communication is encrypted.

Set up SSL certificates for intra-cluster communications

1. Create a directory *cluster* under *<NEO4J_HOME>/certificates* folder:

```
mkdir certificates/cluster
```

2. Create a directory *trusted* and *revoked* under *<NEO4J_HOME>/certificates/cluster* folder:

```
mkdir certificates/cluster/trusted
mkdir certificates/cluster/revoked
```

3. Place the certificates *private.key* and the *public.crt* files under *<NEO4J_HOME>/certificates/cluster* folder:

```
cp /path/to/certs/private.key certificates/cluster
cp /path/to/certs/public.crt certificates/cluster
```

4. Place the *public.crt* file under the *<NEO4J_HOME>/certificates/cluster/trusted* folder.

```
cp /path/to/certs/public.crt certificates/cluster/trusted
```

> **Tip:**
>
> If each server has a certificate of its own, signed by a CA, then each server's public certificate has to be put in the *trusted* folder on each instance of the cluster. Thus, the

servers are able to establish trust relationships with each other.

5. (Optional) If a particular certificate is revoked, then place it under *<NEO4J_HOME>/certificates/cluster/revoked* folder.

```
cp /path/to/certs/public.crt certificates/cluster/revoked
```

Verify the folder structure and permissions

The folder structure should look like this with the right file permissions and the groups and ownerships:

Path	Directory/File	Owner	Group	Permission	Unix/Linux View
/data/neo4j/certificates/cluster	Directory	neo4j	neo4j	0755	drwxr-xr-x
/data/neo4j/certificates/cluster/public.crt	File	neo4j	neo4j	0644	-rw-r--r--
/data/neo4j/certificates/cluster/private.key	File	neo4j	neo4j	0400	-r--------
/data/neo4j/certificates/cluster/trusted	Directory	neo4j	neo4j	0755	drwxr-xr-x
/data/neo4j/certificates/cluster/trusted/public.crt	File	neo4j	neo4j	0644	-rw-r--r--
/data/neo4j/certificates/cluster/revoked	Directory	neo4j	neo4j	0755	drwxr-xr-x

Tip:

The owner/group should be configured to the user/group that will be running the `neo4j` service. Default user/group is neo4j/neo4j.

Set the cluster SSL configuration in *neo4j.conf*

1. Set the cluster SSL policy to `true`:

```
dbms.ssl.policy.cluster.enabled=true
```

2. Set the appropriate certificates path and the right key and cert files:

```
dbms.ssl.policy.cluster.base_directory=certificates/cluster
dbms.ssl.policy.cluster.private_key=private.key
dbms.ssl.policy.cluster.public_certificate=public.crt
```

Tip:

If the certificate is located outside of `NEO4J_HOME`, then set the absolute path for the certificates directory.

3. Set the cluster client authentication to `REQUIRE` to enable the mutual authentication, which means that both ends of a channel have to authenticate:

```
dbms.ssl.policy.cluster.client_auth=REQUIRE
```

> **Note:**
>
> The policy must be configured on every server with the same settings. The actual cryptographic objects installed will be mostly different since they do not share the same private keys and corresponding certificates. The trusted CA certificate will be shared however.

Verify that the intra-cluster communication is encrypted

Once the intra-cluster encryption is enabled, you can verify that the communication is encrypted. You may use an external tooling, such as Nmap (https://nmap.org/download.html):

```
nmap --script ssl-enum-ciphers -p <port> <hostname>
```

> **Note:**
>
> The hostname and port have to be adjusted according to your configuration. This can prove that TLS is in fact enabled and that only the intended cipher suites are enabled. All servers and all applicable ports should be tested. If the intra-cluster encryption is enabled, the output should indicate the port is open and it is using TLS with the ciphers used.

> **Tip:**
>
> For more details on securing the comunication between the cluster servers, see Intra-cluster encryption.

Configure SSL for backup communication

In a single instance, the backup communication happens on port `6362` by default.
In a cluster topology, it is possible to take a backup from any server, and each server has two configurable ports capable of serving a backup:

- `dbms.backup.listen.address` (port `6362`)
- `server.cluster.listen_address` (port `6000`)

If the intra-cluster encryption is enabled and the backup communication uses port `6000`, then the communication channels are already encrypted.

However, if your backup communication uses a different port, you need to enable SSL for it by creating a separate SSL policy.

When setting up SSL for backup communication, you can choose between two options for certificates:

- self-signed certificates — This is typically the case where you control both ends of the connection, and the distribution of certificates can be automated and secured. However, self-signed certificates do not verify the identity of the server, so they cannot be trusted by clients and are vulnerable to man-in-the-middle attacks.
- certificates signed by a certificate authority (CA) — more secure option, because a known trusted authority verifies the identity of the server, ensuring authenticity and preventing impersonation.

You can configure SSL for backup communication in one of the following ways:

- Use the same certificates in the *trusted_dir* on both the backup server and client.
  This approach is simpler to configure but less secure, as both ends share the same identity and trust the same certificate.
- Mirror the certificates on the backup server and client to achieve mutual authentication.
  Add the server certificate to the client's *trusted_dir* and the client certificate to the server's *trusted_dir*.
  This way, the server validates the client's certificate in addition to the typical normal TLS where only the client validates the server's certificate.
- Use a certificate authority (CA) to sign both the client's and server's certificates. In this case, the *trusted_dir* must contain only the CA or intermediate certificates.

Set up SSL certificates for backup

Create the folder structure and place the key and certificate files under it.

1. Create a directory *backup* under *<NEO4J_HOME>/certificates* folder:

```
mkdir certificates/backup
```

2. Create a directory *trusted* and *revoked* under *<NEO4J_HOME>/certificates/backup* folder:

```
mkdir certificates/backup/trusted
mkdir certificates/backup/revoked
```

3. Place the certificates *private.key* and the *public.crt* files under *<NEO4J_HOME>/certificates/backup* folder:

```
cp /path/to/certs/private.key certificates/backup
cp /path/to/certs/public.crt certificates/backup
```

4. Place the *public.crt* file under the *<NEO4J_HOME>/certificates/backup/trusted* folder.

```
cp /path/to/certs/public.crt certificates/backup/trusted
```

5. (Optional) If a particular certificate is revoked, then place it under *<NEO4J_HOME>/certificates/backup/revoked* folder.

```
cp /path/to/certs/public.crt certificates/backup/revoked
```

Verify the backup folder structure and permissions

The folder structure should look like this with the right file permissions and the groups and ownerships, assuming the `neo4j` user runs the service:

Path	Directory/File	Owner	Group	Permission	Unix/Linux View
/data/neo4j/certificates/backup	Directory	neo4j	neo4j	0755	drwxr-xr-x
/data/neo4j/certificates/backup/public.crt	File	neo4j	neo4j	0644	-rw-r--r--
/data/neo4j/certificates/backup/private.key	File	neo4j	neo4j	0400	-r--------
/data/neo4j/certificates/backup/trusted	Directory	neo4j	neo4j	0755	drwxr-xr-x
/data/neo4j/certificates/backup/trusted/public.crt	File	neo4j	neo4j	0644	-rw-r--r--
/data/neo4j/certificates/backup/revoked	Directory	neo4j	neo4j	0755	drwxr-xr-x

Tip:

The owner/group should be configured to the user/group that will be running the `neo4j` service. Default user/group is neo4j/neo4j.

Set the backup SSL configuration in the *neo4j.conf* file.

1. Set the backup SSL policy to `true`:

```
dbms.ssl.policy.backup.enabled=true
```

2. Set the appropriate certificates path and the right key and cert files:

```
dbms.ssl.policy.backup.base_directory=certificates/backup
dbms.ssl.policy.backup.private_key=private.key
dbms.ssl.policy.backup.public_certificate=public.crt
```

Tip:

If the certificate is located outside of `NEO4J_HOME`, then set the absolute path for the certificates directory.

3. Set the backup client authentication to `REQUIRE` to enable the mutual authentication, which means that both ends of a channel have to authenticate:

```
dbms.ssl.policy.backup.client_auth=REQUIRE
```

Note:

Neo4j does not validate the `setting-suffix` and if it is misspelled or incorrect, it will be ignored. See Configuration for more details.

Configure the backup client for SSL

When using `neo4j-admin backup` command, the backup client needs to be configured to use SSL as well. Regardless of which backup port you are targeting (see Configure SSL for backup communication), the backup client uses the SSL policy specified in `dbms.ssl.policy.backup.*`, given the same SSL policy name matches between server and client.

If the backup client is on a different machine from the backup server, you must install SSL certificates and keys on the backup client machine as well, so that the backup client can authenticate the server and vice versa.

The following steps assume that you have already set up the SSL certificates and keys on the backup server machine and you are using the self-signed certificates.

For example, if you have set up the backup SSL policy described in section Configure SSL for backup communication, then you need to set the following in the *neo4j-admin.conf* file on the backup client machine:

```
dbms.ssl.policy.backup.enabled=true
dbms.ssl.policy.backup.base_directory=certificates/backup
dbms.ssl.policy.backup.private_key=private.key
dbms.ssl.policy.backup.public_certificate=public.crt
dbms.ssl.policy.backup.client_auth=REQUIRE
dbms.ssl.policy.backup.tls_versions=TLSv1.2,TLSv1.3
dbms.ssl.policy.backup.ciphers=TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA3
84
```

Other configurations for SSL

Using encrypted private key

To use an encrypted private key, configure the following settings. The private key password must be clear text format without any quotes.

Bolt

```
dbms.ssl.policy.bolt.private_key_password=<clear text password>
```

HTTPS

```
dbms.ssl.policy.https.private_key_password=<password>
```

Intra-cluster encryption

```
dbms.ssl.policy.cluster.private_key_password=<password>
```

Backup

```
dbms.ssl.policy.backup.private_key_password=<password>
```

If hardcoding of clear text private key password is not feasible due to security constraints, it can be set up to use dynamic password pickup by following these steps:

1. Create a file containing the `cleartext` password for the private key password and encrypt it with the certificate (assuming private key for cert has password set and certificate is in `pwd`):

   ```
   echo "password123" > passwordfile

   base64 -w 0 certificate.crt | openssl aes-256-cbc -a -salt -in passwordfile -out password.enc -pass
   stdin
   ```

   > **Note:**
   >
   > Delete the password file and set file permissions for `password.enc` to `400` (e.g. `chmod 400 password.enc`).

2. Verify that encrypted password can be read from password.enc:

   ```
   base64 -w 0 certificate.crt | openssl aes-256-cbc -a -d -in password.enc -pass stdin
   ```

3. Set the neo4j.conf `dbms.ssl.policy.<type>.private_key_password` to be able to read out encrypted password. To adjust paths to cert and encrypted password file, use full paths:

   ```
   dbms.ssl.policy.bolt.private_key_password=$(base64 -w 0 certificate.crt | openssl aes-256-cbc -a -d
   -in password.enc -pass stdin)
   ```

> **Note:**
>
> Using a dynamic command requires Neo4j to be started with the `--expand-commands` option. For more information, see Command expansion.

Using specific cipher

There are cases where Neo4j Enterprise requires the use of specific ciphers for encryptions. One can set up a Neo4j configuration by specifying the list of cipher suites that will be allowed during cipher negotiation. Valid values depend on the current JRE and SSL provider. For Oracle JRE here is the list of supported ones - https://docs.oracle.com/en/java/javase/17/docs/specs/security/standard-names.html#jsse-cipher-suite-names.

> **Important:**

CBC (cipher block chaining) ciphers, as used in TLS v1.2 network encryption, have several security vulnerabilities that make them less secure than alternative methods. The Internet Engineering Task Force (IETF) does not recommend using CBC-based ciphers (RFC 8447), and these ciphers were removed from the TLS standard with the development of TLS v1.3.

Note that the use of the following CBC-based ciphers are deprecated from Neo4j 5.26:

- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA256

Bolt

```
dbms.ssl.policy.bolt.ciphers=TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
4
```

HTTPS

```
dbms.ssl.policy.https.ciphers=TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA38
44
```

Intra-cluster encryption

```
dbms.ssl.policy.cluster.ciphers=TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA
3844
```

Backup

```
dbms.ssl.policy.backup.ciphers=TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA3
844
```

Using OCSP stapling

From Neo4j 4.2, Neo4j supports OCSP stapling, which is implemented on the server side, and can be configured in the *neo4j.config* file. OCSP stapling is also available for Java Bolt driver and HTTP API.

On the server side in the *neo4j.conf* file, configure the following settings:

1. Set the SSL Bolt policy to `true`:

   ```
   dbms.ssl.policy.bolt.enabled=true
   ```

2. Enable the OCSP stapling for Bolt:

   ```
   server.bolt.ocsp_stapling_enabled=true (default = false)
   ```

SSL logs

All information related to SSL can be found in the *debug.log* file. You can also enable additional debug logging for SSL by adding the following configuration to the *neo4j.conf* file and restarting Neo4j.

```
server.jvm.additional=-Djavax.net.debug=ssl:handshake
```

This will log additional information in the *neo4j.log* file. In some installations done using `rpm` based installs, *neo4j.log* is not created. To get the contents of this, since *neo4j.log* just contains `STDOUT` content, look for the `neo4j` service log contents using `journalctl`:

```
neo4j@ubuntu:/var/log/neo4j$ journalctl -u neo4j -b > neo4j.log
neo4j@ubuntu:/var/log/neo4j$ vi neo4j.log
```

Warning:

Beware that the SSL debug option logs a new statement every time a client connects over SSL, which can make *neo4j.log* grow large reasonably quickly. To avoid that scenario, make sure this setting is only enabled for a short term duration.

Terminology

The following terms are relevant to SSL support within Neo4j:

Certificate Authority (CA)

A trusted entity that issues electronic documents that can verify the identity of a digital entity. The term commonly refers to globally recognized CAs, but can also include internal CAs that are trusted inside of an organization. The electronic documents are digital certificates. They are an essential part of secure communication, and play an important part in the Public Key Infrastructure.

Certificate Revocation List (CRL)

In the event of a certificate being compromised, that certificate can be revoked. This is done by means of a list (located in one or several files) spelling out which certificates are revoked. The CRL is always issued by the CA which issues the corresponding certificates.

cipher

An algorithm for performing encryption or decryption. In the most general implementation of encrypted communications, Neo4j makes implicit use of ciphers that are included as part of the Java platform. The configuration of the SSL framework also allows for the explicit declaration of allowed ciphers.

communication channel

A means for communicating with the Neo4j database. Available channels are:

- Bolt client traffic
- HTTPS client traffic
- intra-cluster communication
- backup traffic

cryptographic objects

A term denoting the artifacts private keys, certificates and CRLs.

configuration parameters

These are the parameters defined for a certain ssl policy in *neo4j.conf*.

certificate

SSL certificates are issued by a trusted certificate authority (CA). The public key can be obtained and used by anyone to encrypt messages intended for a particular recipient. The certificate is commonly stored in a file named *<file name>.crt*. This is also referred to as the public key.

SAN

SAN is an acronym for *Subject Alternative Names*. It is an extension to certificates that one can include optionally. When presented with a certificate that includes SAN entries, it is recommended that the address of the host is checked against this field. Verifying that the hostname matches the certificate SAN helps prevent attacks where a rogue machine has access to a valid key pair.

SSL

SSL is an acronym for *Secure Sockets Layer*, and is the predecessor of TLS. It is common to refer to SSL/TLS as just SSL. However, the modern and secure version is TLS, which is also the default in Neo4j.

SSL policy

An SSL policy in Neo4j consists of a a digital certificate and a set of configuration parameters defined in *neo4j.conf*.

PKCS #1

PKCS #1 is the first family of standards called Public-Key Cryptography Standards (PKCS). It provides the basic definitions and recommendations for implementing the RSA algorithm for public-key cryptography. It defines the mathematical properties of public and private keys, primitive operations for encryption and signatures, secure cryptographic schemes, and related ASN.1 syntax representations.

PKCS #5

PKCS #5 contains recommendations for implementating password-based cryptography, covering key derivation functions, encryption schemes, message authentication schemes, and *ASN.1* syntax, identifying the techniques.

PKCS #8

PKCS #8 is a standard syntax for storing private key information. The PKCS #8 private key may be encrypted with a passphrase using the PKCS #5 standards, which support multiple ciphers. The main difference from PKCS #1 is that it allows more algorithms than RSA and supports stronger encryption of the private key.

private key

The private key ensures that encrypted messages can be deciphered only by the intended recipient. The private key is commonly stored in a file named *<file name>.key*. It is important to protect the private key to ensure the integrity of encrypted communication.

Public Key Infrastructure (PKI)

A set of roles, policies, and procedures needed to create, manage, distribute, use, store, and revoke digital certificates and manage public-key encryption.

public key

The public key can be obtained and used by anyone to encrypt messages intended for a particular recipient. This is also referred to as the certificate.

TLS protocol

The cryptographic protocol that provides communications security over a computer network. The Transport Layer Security (TLS) protocol and its predecessor, the Secure Sockets Layer (SSL) protocol, are both frequently referred to as "SSL".

TLS version

A version of the TLS protocol.

X.509

X.509 is an International Telecommunication Union (ITU) standard defining the format of public key certificates.

Configuring SSL for FIPS 140-2 compatibility

Enterprise Edition | Introduced in 5.24

Federal Information Processing Standards (FIPS) 140 is a U.S. government standard established by the National Institute of Standards and Technology (NIST) which is used to accredit cryptographic modules such as those used in TLS network encryption. While FIPS 140 compliance is primarily required for federal agencies and their contractors, it also is used in the healthcare sector under regulations like the Health Insurance Portability and Accountability Act (HIPAA) to protect patient data.

This guide helps configure Neo4j to use TLS/SSL encryption in a FIPS-compliant way. It is supplementary to the SSL framework documentation, as many of the configuration processes and requirements are the same.

Prerequisites

- Verify that the machine running Neo4j has FIPS-compatible hardware and operating system. Only Linux operating systems are supported for Neo4j FIPS compatibility at this time.
- Use Neo4j Enterprise 5.23.0 or later.
- Install and configure a non-native authentication provider, for example LDAP or SSO. See Authentication and authorization.

Enable FIPS SSL provider

The secure networking in Neo4j is provided through the Netty library, which supports both the native JDK SSL provider and Netty-supported OpenSSL derivatives. Specifically Netty's *Forked Tomcat Native* library called netty-tcnative.

The `netty-tcnative` library is provided in several variants. However, to achieve FIPS compliance, you must use the dynamically linked version of `netty-tcnative` alongside a FIPS-compatible installation of OpenSSL.

The dynamically linked library requires the following dependencies to be installed[1]:

- Apache Portable Runtime Library
- A FIPS certified version of OpenSSL, with a FIPS provider installed and set as default.

Refer to Forked Tomcat Native for more information.

Note:

Netty provides a convenient pre-build, statically linked version of `netty-tcnative` using BoringSSL, but this is not FIPS certified[2].

By using the dynamic `netty-tcnative` library variant combined with a FIPS certified OpenSSL installation, Neo4j's cryptographic operations are delegated by `netty-tcnative` to OpenSSL, transitively giving FIPS compatibility.

Install Apache portable runtime library

To install Apache Portable Runtime Library, use the operating system's package manager.

In Debian/Ubuntu this package is usually called `libapr1`

Install Apache Portable Runtime Library in Debian or Ubuntu

```
apt install -y libapr1
```

In RedHat Enterprise Linux, the package is usually called `apr`:

Install Apache Portable Runtime Library in RedHat

```
dnf install -y apr
```

Install OpenSSL

Instructions on how to build and install a FIPS-compatible OpenSSL are out of scope for this document. Installation steps can differ depending on operating system, and other security requirements you might have for OpenSSL.

In general:

- For a list of FIPS certified OpenSSL versions, see https://openssl-library.org/source/.
- A FIPS provider must be installed into OpenSSL.
- OpenSSL must be configured to use the FIPS provider by default.

Install the correct `netty-tcnative` library

Since Neo4j 5.23.0, builds of `netty-tcnative` dynamic library are provided in the Neo4j `lib` directory under their own subfolder called `netty-tcnative`.

To install the `netty-tcnative` dynamic library:

1. Locate the Neo4j `lib` directory.

   The location of the `lib` directory is different depending on the method used to install Neo4j. Check the file locations documentation for the correct location.

   This location will be referred to as *<NEO4J_LIB>*.

2. Make sure there are no `netty-tcnative-boringssl` libraries present in the *<NEO4J_LIB>* folder.

   ```
   find <NEO4J_LIB> -name "netty-tcnative-boringssl*.jar" -delete
   ```

3. Check which netty-tcnative libraries are available:

   ```
   ls -l <NEO4J_LIB>/netty-tcnative
   ```

   There are Linux and Fedora Linux variants available, compiled for both x86_64 and ARM 64 architectures. Select the one matching the local machine's operating system and architecture.

4. Verify the dependencies are correctly installed using `ldd`:

   Verify netty-tcnative dependencies are installed

   ```
   unzip -d /tmp <NEO4J_LIB>/netty-tcnative/netty-tcnative-*-linux-$(arch).jar
   ldd /tmp/META-INF/native/libnetty_tcnative_linux_*.so
   rm -rf /tmp/META-INF
   ```

   Verify Fedora variant of netty-tcnative dependencies are installed

   ```
   unzip -d /tmp <NEO4J_LIB>/netty-tcnative/netty-tcnative-*-linux-$(arch)-fedora.jar
   ldd /tmp/META-INF/native/libnetty_tcnative_linux_$(arch).so
   rm -rf /tmp/META-INF
   ```

   The `ldd` command shows a list of library dependencies and where they are loaded from on the local machine.

   - If any dependencies are missing, they must be installed, or Neo4j will fail to run.
   - The `libssl.so` and `libcrypto.so` libraries listed must be the ones installed with OpenSSL in the previous steps.

5. Copy the verified JAR file to *<NEO4J_LIB>*.

   > **Note:**
   >
   > Only copy **one** of the JAR files. Otherwise Neo4j will not be able to resolve dependencies at runtime. In case of this error, you will get a message like:

```
"Failed to load any of the given libraries: [netty_tcnative_linux_x86_64,
netty_tcnative_linux_x86_64_fedora, netty_tcnative_x86_64, netty_tcnative]".
```

Generate SSL certificate and private key

Neo4j SSL encryption requires a certificate in the X.509 standard and a private key in PKCS #8 format, both encoded in PEM format.

Important:

For FIPS compatibility, the private key must be secured with a password.

Refer to the SSL certificate and key instructions for more information.

Configure Neo4j to use SSL encryption

SSL configuration is described in detail in SSL framework configuration.

This section describes configuration that must be done **in addition to** standard non-FIPS compliant SSL configuration.

Note:

- The following group of FIPS-compatible cipher suites is for use with TLSv1.2:
  - `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`
  - `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`
  - `TLS_DHE_RSA_WITH_AES_256_GCM_SHA384`
  - `TLS_DHE_RSA_WITH_AES_128_GCM_SHA256`

  They require additional configuration in the application or OpenSSL settings.
- The following cipher suites are supported by default in OpenSSL when using TLSv1.3:
  - `TLS_AES_256_GCM_SHA384`
  - `TLS_AES_128_GCM_SHA256`

  These suites do not require additional configuration when OpenSSL is built with FIPS support.

Bolt

1. Set `dbms.netty.ssl.provider =OPENSSL`
2. Set `server.bolt.tls_level =REQUIRED`
3. Follow instructions on how to Configure SSL over Bolt.
4. Set additional Bolt configurations:

```
dbms.ssl.policy.bolt.trust_all=false
dbms.ssl.policy.bolt.tls_level=REQUIRED
dbms.ssl.policy.bolt.tls_versions=TLSv1.2,TLSv1.3
dbms.ssl.policy.bolt.ciphers=TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA2
56,TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,TLS_AES_256_GCM_SHA384,TLS_
AES_128_GCM_SHA256
```

5. Follow the instructions in SSL Framework → Using encrypted private key to configure `dbms.ssl.policy.bolt.private_key_password` to dynamically read the password from an encrypted password file. The password must **not** be set in plain text.

HTTPS

This section is only applicable if HTTPS is enabled.

1. Follow instructions on how to Configure SSL over HTTPS.
2. Set additional HTTPS configurations:

```
dbms.ssl.policy.https.trust_all=false
dbms.ssl.policy.https.tls_level=REQUIRED
dbms.ssl.policy.https.tls_versions=TLSv1.2,TLSv1.3
dbms.ssl.policy.https.ciphers=TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA
256,TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,TLS_AES_256_GCM_SHA384,TLS
_AES_128_GCM_SHA256
```

3. Follow the instructions in SSL Framework → Using encrypted private key to configure `dbms.ssl.policy.https.private_key_password` to dynamically read the password from an encrypted password file. The password must NOT be set in plain text.

Intra-cluster encryption

For FIPS compatbility, intra-cluster encryption must be enabled if you are running a Neo4j cluster.

1. Follow instructions to configure SSL for intra-cluster communication.
2. Set additional cluster configurations:

```
dbms.ssl.policy.cluster.enabled=true
dbms.ssl.policy.cluster.tls_level=REQUIRED
dbms.ssl.policy.cluster.client_auth=REQUIRED
dbms.ssl.policy.cluster.tls_versions=TLSv1.2,TLSv1.3
dbms.ssl.policy.cluster.ciphers=TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_128_GCM_S
HA256,TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,TLS_AES_256_GCM_SHA384,T
LS_AES_128_GCM_SHA256
```

3. Follow the instructions in SSL Framework → Using encrypted private key to configure `dbms.ssl.policy.cluster.private_key_password` to dynamically read the password from an encrypted password file. The password must **not** be set in plain text.

Backup

This section is applicable on instances or cluster members used for taking backups.

1. Follow instructions on how to Configure SSL for backup communication.

2. Set additional backup configurations:

```
dbms.ssl.policy.backup.enabled=true
dbms.ssl.policy.backup.client_auth=REQUIRED
dbms.ssl.policy.backup.trust_all=false
dbms.ssl.policy.backup.tls_versions=TLSv1.2,TLSv1.3
dbms.ssl.policy.backup.ciphers=TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,TLS_AES_256_GCM_SHA384,TLS_AES_128_GCM_SHA256
```

3. Follow the instructions in SSL Framework → Using encrypted private key to configure `dbms.ssl.policy.backup.private_key_password` to dynamically read the password from an encrypted password file. The password must **not** be set in plain text.

Browser credentials handling

Neo4j Browser has two mechanisms for avoiding users having to repeatedly enter their Neo4j credentials. This page explains how to control how credentials are handled.

First, while the Browser is open in a web browser tab, it ensures that the existing database session is kept alive. This is subject to a timeout. The timeout is configured in the setting `browser.credential_timeout`. The timeout is reset whenever there is user interaction with the Browser.

Second, the Browser can also cache the user's Neo4j credentials locally. When credentials are cached, they are stored unencrypted in the web browser's local storage. If the web browser tab is closed and then re-opened, the session is automatically re-established using the cached credentials. This local storage is also subject to the timeout configured in the setting `browser.credential_timeout`. In addition, caching credentials in browser local storage can be disabled altogether. To disable credentials caching, set `browser.retain_connection_credentials=false` in the server configuration.

If the user issues a `:server disconnect` command then any existing session is terminated and the credentials are cleared from local storage.

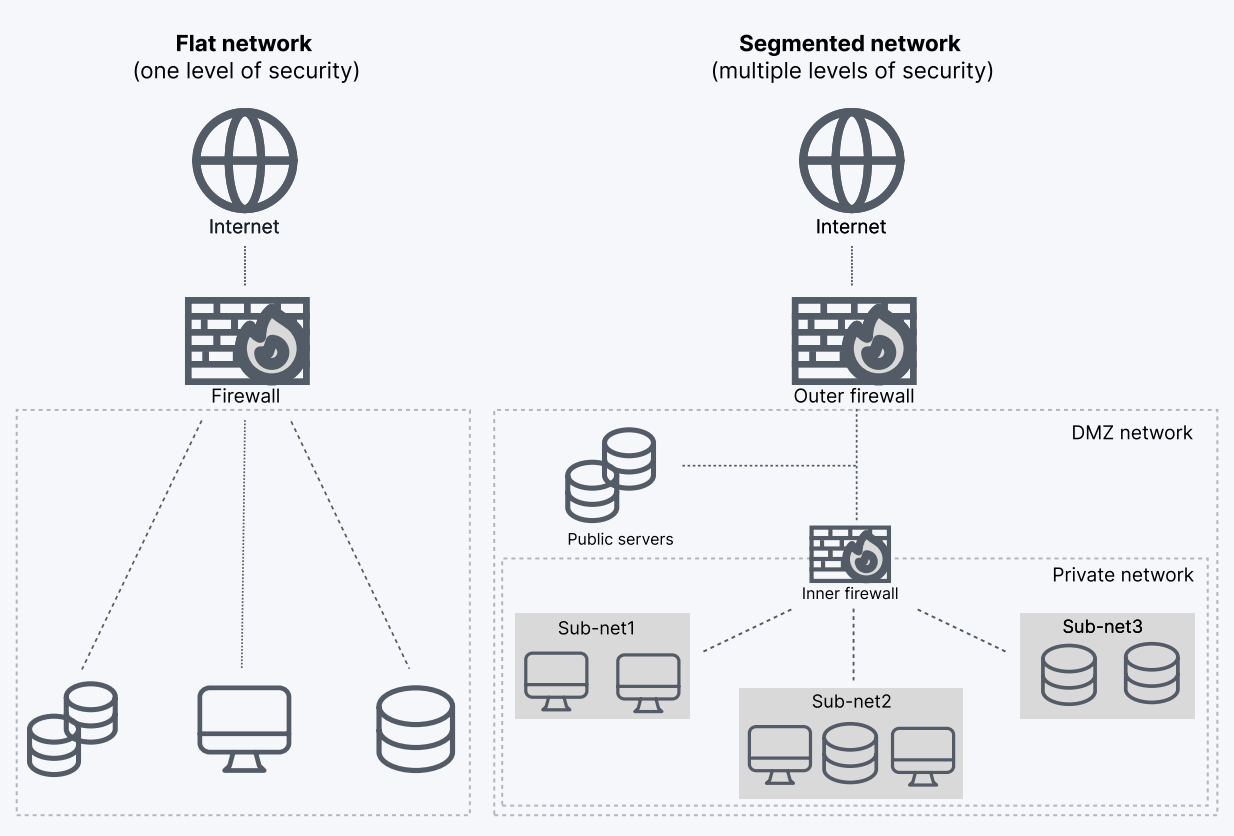
Note:

For more information on how to administer and use Neo4j Browser, see the Neo4j Browser manual → Browser operations.

Security checklist

The following checklist highlights the specific areas within Neo4j that may need extra attention to ensure the appropriate level of security for your application after Neo4j is deployed.

1. Deploy Neo4j on safe servers in secure networks:
    a. Use subnets and firewalls to segment the network.



b. Open only the ports that you need. For a list of relevant ports, see Ports.

   In particular, ensure that there is no external access to the port specified by the setting `server.backup.listen_address`. Failing to protect this port may open a security hole by which an unauthorized user can make a copy of the database onto a different machine.

2. Protect data-at-rest:

   a. Use volume encryption (e.g., Bitlocker).

   b. Manage access to database dumps and backups. Refer to Back up an offline database and backups Back up an online database for more information.

   c. Manage access to configuration files, data files, and transaction logs by ensuring the correct file permissions on the Neo4j files. Refer to Default file locations for instructions on permission levels.

   d. If a directory stores sensitive data, it must never be world-readable or world-executable. Even if files inside are restricted, a world-executable directory can leak structure.

      - On Linux, follow the recommendations:

Scenario	Recommended mode
Only owner can access	`700`
Owner and specific group may have access	`750`

      Ensure the process creating the directory uses a restrictive umask (e.g., `077`) to prevent overly permissive defaults.

      - On Windows, allow access only to the owning account (or service account) and

`Administrators`. Remove access for `Users`, `Authenticated Users`, and `Everyone`. Before removing broad permissions disable inheritance on the directory, otherwise they may be reintroduced from the parent directory.

Consider security recommendations above when creating a directory for backup files.

3. Protect data-in-transit:
    a. For remote access to the Neo4j database, only use encrypted Bolt or HTTPS.
    b. Use SSL certificates issued from a trusted Certificate Authority.
    c. For configuring your Neo4j installation to use encrypted communication, refer to SSL framework.
    d. If using clustering, configure and use encryption for intra-cluster communication. For details, see Intra-cluster encryption.
    e. If using clustering, configure and use encryption for backups. This ensures that only servers with the specified SSL policy and SSL certificates can access the server and perform the backup.
    f. For configuring your Bolt and HTTPS connectors, refer to Configure connectors.
    g. If using LDAP, configure your LDAP system with encryption via StartTLS. For more information, see Use LDAP with encryption via StartTLS.
4. Be on top of the security for custom extensions:
    a. Validate any custom code you deploy (procedures and unmanaged extensions) and ensure that they do not unintentionally expose any parts of the product or data.
    b. Survey the settings `dbms.security.procedures.unrestricted` and `dbms.security.procedures.allowlist` to ensure that they exclusively contain intentionally exposed extensions.
5. Make sure you have the Default file locations on the Neo4j files.
6. Protect against the execution of unauthorized extensions by restricting access to the *bin*, *lib*, and *plugins* directories. Only the operating system user that Neo4j runs as should have permissions to those files. Refer to Default file locations for instructions on permission levels.
7. With `LOAD CSV` enabled, ensure that it does not allow unauthorized users to import data. How to configure `LOAD CSV` is described in Cypher Manual → `LOAD CSV`.
8. Use Neo4j authentication. The setting `dbms.security.auth_enabled` controls native authentication. The default value is `true`.
9. Survey your JVM-specific configuration settings in the *neo4j.conf* file for ports relating to deprecated functions, such as remote JMX (controlled by the parameter setting `server.jvm.additional=-Dcom.sun.management.jmxremote.port=3637`).
10. Review Browser credentials handling to determine whether the default credentials handling in Neo4j Browser complies with your security regulations. Follow the instructions to configure it if necessary.
11. Use the latest patch version of Neo4j and set up a process to update it when security advisories are published.

[1] https://netty.io/wiki/forked-tomcat-native.html
[2] https://boringssl.googlesource.com/boringssl/+/master/crypto/fipsmodule/FIPS.md

Performance

This section describes factors that affect operational performance and how to tune Neo4j for optimal throughput. The following topics are covered:

- Memory configuration — How to configure memory settings for efficient operations.
- Index configuration — How to configure indexes.
- Garbage collector — How to configure the Java Virtual Machine's garbage collector.
- Bolt thread pool configuration — How to configure the Bolt thread pool.
- Linux file system tuning — How to configure the Linux file system.
- Disks, RAM and other tips — Disks, RAM and other tips.
- Statistics and execution plans — How schema statistics and execution plans affect Cypher query performance.
- Space reuse — Data deletion and storage space reuse.

Memory configuration

This page describes the different aspects of Neo4j memory configuration and use. The RAM of the Neo4j server has a number of usage areas, with some sub-areas:

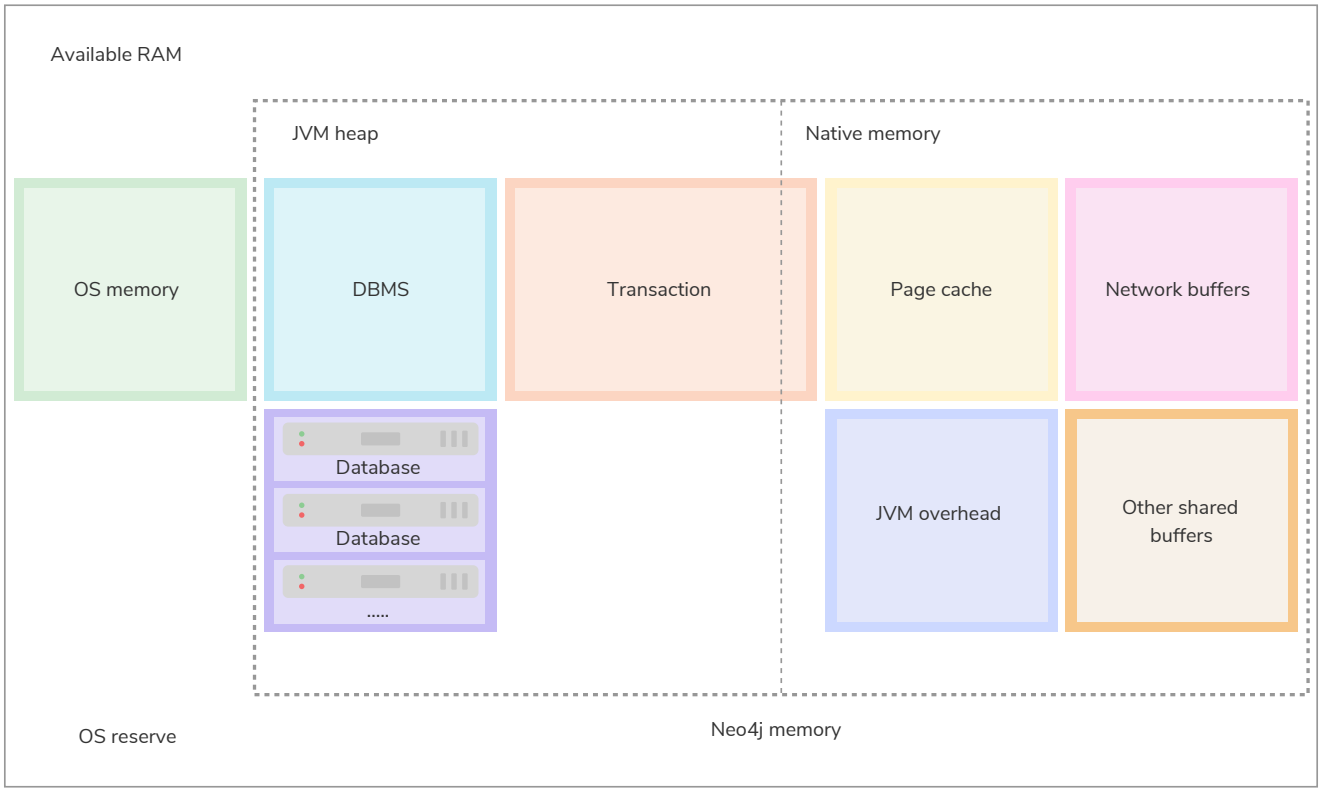


Figure 36. Neo4j memory management

OS memory

Some memory must be reserved for running the processes of the operating system itself. It is not possible to explicitly configure the amount of RAM that should be reserved for the operating system, as

this is what RAM remains available after configuring Neo4j.

1GB is a good starting point for a server that is dedicated to running Neo4j. However, there are cases where the amount reserved for the OS is significantly larger than 1GB, such as servers with exceptionally large RAM.

> **Note:**
>
> If you do not leave enough space for the OS, it will start to swap memory to disk, which will heavily affect performance. Therefore, it is generally recommended to have swap turned off for a Neo4j dedicated server.

JVM Heap

The JVM heap is a separate dynamic memory allocation that Neo4j uses to store instantiated Java objects. The memory for the Java objects are managed automatically by a garbage collector. Particularly important is that a garbage collector automatically handles the deletion of unused objects. For more information on how the garbage collector works and how to tune it, see Tuning of the garbage collector.

The heap memory size is determined by the parameters `server.memory.heap.initial_size` and `server.memory.heap.max_size`. It is recommended to set these two parameters to the same value to avoid unwanted full garbage collection pauses.

Generally, to aid performance, you should configure a large enough heap to sustain concurrent operations.

Native memory

Native memory, sometimes referred to as off-heap memory, is memory directly allocated by Neo4j from the OS. This memory will grow dynamically as needed and is not subject to the garbage collector.

DBMS

The database management system, or DBMS, contains the global components of the Neo4j instance. For example, the bolt server, logging service, monitoring service, etc.

Database

Each database in the system comes with an overhead. In deployments with multiple databases, this overhead needs to be accounted for.

Transaction

When executing a transaction, Neo4j holds not yet committed data, the result, and intermediate states of the queries in memory. The size needed for this is very dependent on the nature of the usage of Neo4j. For example, long-running queries, or very complicated queries, are likely to require more memory. Some parts of the transactions can optionally be placed off-heap, but for the best performance, it is recommended to keep the default with everything on-heap.

This memory group can be limited with the setting `dbms.memory.transaction.total.max`.

Page cache

The page cache is used to cache the Neo4j data stored on disk. The caching of graph data and indexes into memory helps avoid costly disk access and result in optimal performance.

The parameter for specifying how much memory Neo4j is allowed to use for the page cache is: `server.memory.pagecache.size`.

Network buffers

Direct buffers are used by Neo4j to send and receive data. Direct byte buffers are important for improving performance because they allow native code and Java code to share data without copying it. However, they are expensive to create, which means byte buffers are usually reused once they are created.

Other shared buffers

This includes unspecified shared direct buffers.

JVM overhead

The JVM will require some memory to function correctly. For example, this can be:

- **Thread stacks** – Each thread has its own call stack. The stack stores primitive local variables and object references along with the call stack (list of method invocations) itself. The stack is cleaned up as stack frames move out of context, so there is no GC performed here.
- **Metaspace** – Metaspace stores the java class definitions and some other metadata.
- **Code cache** – The JIT compiler stores the native code it generates in the code cache to improve performance by reusing it.

For more details and means of limiting the memory used by the JVM please consult your JVM documentation.

Considerations

Always use explicit configuration

To have good control of the system behavior, it is recommended to always define the page cache and heap size parameters explicitly in *neo4j.conf*. Otherwise, Neo4j computes some heuristic values at startup based on the available system resources.

Initial memory recommendation

Use the `neo4j-admin server memory-recommendation` command to get an initial recommendation for how to distribute a certain amount of memory. The values may need to be adjusted to cater for each specific use case.

Inspect the memory settings of all databases in a DBMS

The `neo4j-admin server memory-recommendation` command is useful for inspecting the current distribution of data and indexes.

Example 139. Use `neo4j-admin server memory-recommendation` to inspect the memory settings of all your databases

Estimate the total size of the database files.

```
bin/neo4j-admin server memory-recommendation
...
...
...
# Total size of lucene indexes in all databases: 6690m
# Total size of data and native indexes in all databases: 17050m
```

You can see that the Lucene indexes take up approximately 6.7GB of data, and that the data volume and native indexes combined take up approximately 17GB.

Using this information, you can do a sanity check of your memory configuration:

- Compare the value for data volume and native indexes to the value of `server.memory.pagecache.size`.
- For cases when *off-heap* transaction state is used, estimate transactional workload and how much memory is left to the value of `dbms.tx_state.max_off_heap_memory`.
- Compare the value for Lucene indexes to how much memory is left after assigning `server.memory.pagecache.size` and `server.memory.heap.initial_size`.

In some production systems the access to memory is limited and must be negotiated between different areas. Therefore, it is recommended to perform a certain amount of testing and tuning of these settings to figure out the optimal division of the available memory.

Limit transaction memory usage recommendation

The measured heap usage of all transactions is only an estimate and the actual heap utilization may be slightly larger or slightly smaller than the estimated value. In some cases, limitations of the estimation algorithm to detect shared objects at a deeper level of the memory graph could lead to overestimations. This is because a conservative estimate is given based on aggregated estimations of memory usage, where the identities of all contributing objects are not known, and cannot be assumed to be shared. For example, when you use `UNWIND` on a very large list, or expand a variable length or shortest path pattern, where many relationships are shared between the computed result paths.

In these cases, if you experience problems with a query that gets terminated, you can execute the same query with the transaction memory limit disabled. If the actual heap usage is not too large, it might succeed without triggering an out-of-memory error.

Capacity planning

In many use cases, it is advantageous to try to cache as much of the data and indexes as possible. The following examples illustrate methods for estimating the page cache size, depending on whether you are already running in production or planning for a future deployment:

Example 140. Estimate page cache for the existing Neo4j databases

First, estimate the total size of data and indexes, and then multiply with some factor, for example 20%, to allow for growth.

```
bin/neo4j-admin server memory-recommendation
...
...
...
# Total size of lucene indexes in all databases: 6690m
# Total size of data and native indexes in all databases: 35050m
```

You can see that the data volume and native indexes combined take up approximately 35GB. In your specific use case, you estimate that 20% will provide sufficient head room for growth.

`server.memory.pagecache.size` = 1.2 * (35GB) = 42GB

You configure the page cache by adding the following to *neo4j.conf*:

```
server.memory.pagecache.size=42GB
```

Example 141. Estimate page cache for a new Neo4j database

When planning for a future database, it is useful to run an import with a fraction of the data, and then multiply the resulting store size delta by that fraction plus some percentage for growth.

1. Run the `memory-recommendation` command to see the total size of the data and indexes in all current databases.

   ```
   bin/neo4j-admin server memory-recommendation
   ...
   ...
   ...
   # Total size of lucene indexes in all databases: 6690m
   # Total size of data and native indexes in all databases: 35050m
   ```

2. Import 1/100th of the data and again measure the data volume and native indexes of all databases.

   ```
   bin/neo4j-admin server memory-recommendation
   ...
   ...
   ...
   # Total size of lucene indexes in all databases: 6690m
   # Total size of data and native indexes in all databases: 35400m
   ```

   You can see that the data volume and native indexes combined take up approximately 35.4GB.

3. Multiply the resulting store size delta by that fraction.

   35.4GB - 35GB = 0.4GB * 100 = 40GB

4. Multiply that number by 1.2 to size up the result, and allow for 20% growth.

`server.memory.pagecache.size` = 1.2 * (40GB) = 48GB

5. Configure the page cache by adding the following to *neo4j.conf*:

```
server.memory.pagecache.size=48G
```

Limit transaction memory usage

By using the `dbms.memory.transaction.total.max` setting you can configure a global maximum memory usage for all of the transactions running on the server. This setting must be configured low enough so that you do not run out of memory. If you are experiencing `OutOfMemory` messages during high transaction load, try to lower this limit.

Neo4j also offers the following settings to provide fairness, which can help improve stability in multi-tenant deployments.

- The setting `db.memory.transaction.total.max` limits the transaction memory usage per database.
- The setting `db.memory.transaction.max` constrains each transaction.

When any of the limits are reached, the transaction is terminated without affecting the overall health of the database.

To help configure these settings you can use the following commands to list the current usage:

```
CALL dbms.listPools()
SHOW TRANSACTIONS
```

Or alternatively, you can monitor the memory usage of each query in the query.log.

Index configuration

This page describes how to configure Neo4j indexes to enhance search performance and enable full-text search. The supported index types are:

- Range
- Point
- Text
- Full-text
- Token lookup

All types of indexes can be created and dropped using Cypher and they can also all be used to index both nodes and relationships. The token lookup index is the only index present by default in the database.

Range, point, text, and full-text indexes provide a mapping from a property value to an entity (node or relationship). Token lookup indexes are different and provide a mapping from labels to nodes, or relationship types to relationships, instead of between properties and entities.

When you write a Cypher query, you do not need to specify which indexes to use. Cypher's query planner decides which of the available indexes to use.

The rest of this page provides information on the available indexes and their configuration aspects. For further details on creating, querying, and dropping indexes, see Cypher Manual → Indexes to support full-text search.

The type of an index can be identified according to the table below:

Index type	Cypher command	Core API
Range index	`SHOW RANGE INDEXES`	`org.neo4j.graphdb.schema.IndexType#RANGE`
Point index	`SHOW POINT INDEXES`	`org.neo4j.graphdb.schema.IndexType#POINT`
Text index	`SHOW TEXT INDEXES`	`org.neo4j.graphdb.schema.IndexType#TEXT`
Full-text index	`SHOW FULLTEXT INDEXES`	`org.neo4j.graphdb.schema.IndexType#FULLTEXT`
Token lookup index	`SHOW LOOKUP INDEXES`	`org.neo4j.graphdb.schema.IndexType#LOOKUP`
Vector index	`SHOW VECTOR INDEXES`	`org.neo4j.graphdb.schema.IndexType#VECTOR`

Note:

You cannot have indexes of the same type over the same properties.

Range indexes

Range indexes can be used for exact lookups on all types of values, range scans, full scans, and prefix searches.

Range indexes are the most general-purpose of the property indexes, as they support all value types and a wide range of operations.

Limitations on key size

The range index has a key size limit of around 8kB.

If a transaction reaches the key size limit for one or more of its changes, that transaction fails before committing any changes. If the limit is reached during index population, the resulting index is in a failed state, and as such is not usable for any queries.

Workarounds to address limitations

Since the text index has a key size limit of around 32kB, the key size limit of the range index can be worked around by using a text index instead. However, the text index is not a general-purpose index like the range index, so this workaround cannot be applied to all cases. For more information, see Text indexes.

Point indexes

Point indexes are a type of highly-specialized, single-property index and they only index properties with Point values, unlike range indexes.

Point indexes are designed to speed up spatial queries, specifically the `distance` and `bounding box` queries. Exact lookups are the only non-spatial query that this index type supports.

For more information on the queries a point index can be used for, refer to Cypher Manual → Query Tuning → The use of indexes.

Point indexes optionally accept configuration properties for tuning the behavior of spatial search. For more information on configuring point index, refer to Cypher Manual → Indexes for search performance.

Text indexes

Text indexes are a type of single-property index. Unlike range indexes, text indexes index only properties with string values.

Text indexes are specifically designed to deal with `ENDS WITH` or `CONTAINS` queries efficiently. They are used through Cypher and they support a smaller set of string queries. Even though text indexes do support other text queries, `ENDS WITH` or `CONTAINS` queries are the only ones for which this index type provides an advantage over a range index.

For more information on the queries a text index can be used for, refer to Cypher Manual → Query Tuning → The use of indexes.

For more information on the different index types, refer to Cypher Manual → Indexes for search performance.

Note:

Neo4j 5.1 introduces an improved index provider, `text-2.0`, that performs significantly better. New Text indexes will use the `text-2.0` provider by default. Pre-existing indexes will continue to use `text-1.0`. You must re-create these indexes to benefit from this improvement.

Limitations

Text indexes only index single property strings.

The index has a key size limit for single property strings of around 32kB. If a transaction reaches the key size limit for one or more of its changes, that transaction fails before committing any changes. If the limit is reached during index population, the resulting index is in a failed state, and as such is not usable for any queries.

Full-text indexes

Full-text indexes are optimized for indexing and searching texts.

Even though text and full-text indexes might seem to solve very similar problems, there are essential

differences. Unlike text indexes, which index only single property strings, full-text indexes can index any kind of string data. Text indexes solve substring matches and exact string matches according to the semantics defined by the Cypher language. While, full-text indexes use pluggable analyzers, many of which provide language-specific processing of the text that allows for more sophisticated queries than a simple substring match. Depending on which analyzer is used, the full-text index can be used for different text search types, such as exact matches, relevance matches, phrase queries, autocompletion, and many others. Additionally, the results are ordered by relevance.

An example of a use case for full-text indexes is parsing a book for a certain term and taking advantage of the knowledge that the book is written in a certain language. The use of an *analyzer* for that language enables the exclusion of stop words, such as "*if*" and "*and*", and the inclusion of word forms.

Another use case example is indexing the various address fields and text data in a corpus of emails. Using the `email` analyzer, you can find all emails that are sent from/to or mention a specific email account.

In contrast to range and text indexes, full-text indexes are queried using built-in procedures. They are however created and dropped using Cypher. The use of full-text indexes does require familiarity with how those indexes operate.

Full-text indexes are powered by the Apache Lucene indexing and search library. A full description of how to create and use full-text indexes is provided in the Cypher Manual → Indexes to support full-text search.

Configuring a full-text index

The following options are available for configuring full-text indexes. For a complete list of Neo4j procedures, see Procedures.

`db.index.fulltext.default_analyzer`

The name of the default analyzer when creating a new Full-text index. Once created, the index's analyzer is not affected by this setting.

`db.index.fulltext.eventually_consistent`

The default consistency model when creating a new full-text index. Once created, the index's consistency model is not affected by this setting.

Indexes are normally fully consistent, and the committing of a transaction does not return until both the store and indexes are updated. Eventually consistent full-text indexes, on the other hand, are not updated as part of a commit but instead have their updates queued up and applied in a background thread. This means that there can be a short delay between committing a change and that change becoming visible via any eventually consistent full-text indexes. This delay is just an artifact of the queueing and is usually relatively small since eventually consistent indexes are updated "as soon as possible".

By default, this is turned off, and full-text indexes are fully consistent.

`db.index.fulltext.eventually_consistent_index_update_queue_max_length`

Eventually, consistent full-text indexes have their updates queued up and applied in a background thread, and this setting determines the maximum size of that update queue. If the maximum queue size is reached, then committing transactions block and wait until there is more room in the queue before

adding more updates to it.

This setting applies to all eventually consistent full-text indexes, and they all use the same queue. The maximum queue length must be at least 1 index update and no more than 50 million due to heap space usage considerations.

The default maximum queue length is 10.000 index updates.

Selecting an analyzer

By default, the full-text index uses the `standard-no-stop-words` analyzer, specified in `db.index.fulltext.default_analyzer` configuration setting. This analyzer is the same as Lucene's `StandardAnalyzer`, except no stop-words are filtered out.

To specify another analyzer, use the `OPTIONS` clause of the full-text index creation command. The list of all possible analyzers is available via the `db.index.fulltext.listAvailableAnalyzers()` Cypher procedure.

By default, the analyzer analyzes both the indexed values and query string. In some cases, however, using different analyzers for the indexed values and query string is more appropriate. You can do that by specifying an analyzer for the query string when using the full-text search procedures.

For detailed information on how to create and use full-text indexes, see the Cypher Manual → Indexes to support full-text search.

Per-property analyzer

A full-text index can be created over multiple properties. If different analyzers for different properties are required, the standard approach in Lucene is to create a custom Composite analyzer. The Lucene project provides `PerFieldAnalyzerWrapper` that can associate analyzers with specific fields. For more information, see the Lucene official documentation.

Token lookup indexes

Token lookup indexes are used to look up nodes with a specific label or relationships of a specific type. They are always created over all labels or relationship types. Therefore, databases can have a maximum of two token lookup indexes - one for nodes and one for relationships.

Use and significance

Token lookup indexes are the most important indexes as they significantly speed up the population of other indexes. They are also essential for the Cypher queries execution and Core API operations. Therefore, dropping them should be carefully considered.

The node label lookup index is important for queries that match a node by one or more labels. It can also be used for matching labels and properties of a node when there are no suitable indexes available. Likewise, the relationship type lookup index is important for queries that match relationships by their types.

Most queries are executed by matching nodes and expanding their relationships. Hence, the node label

lookup index is slightly more significant than the relationship type lookup index.

Both node and relationship type lookup indexes are present by default in all databases created in 4.3 and onwards.

Databases created before 4.3

Databases created before 4.3 do not get relationship lookup index automatically, in order to preserve the backward compatibility and performance characteristics of such databases.

If needed, such databases can get a relationship type lookup index by creating it explicitly through Cypher.

Warning:

Creating a relationship type lookup index on a large database can take a significant amount of time, as all relationships need to be scanned when populating such an index.

Tuning of the garbage collector

This page discusses the effects of the Java Virtual Machine's garbage collector with regards to Neo4j performance. In this setting, the heap is separated into an *old generation* and a *young generation*, while new objects are allocated in the young generation, and then later moved to the old generation, if they stay live (in use) for long enough.

When a generation fills up, the garbage collector performs a collection, during which all other threads in the process are paused. The young generation is quick to collect since the pause time correlates with the *live* set of objects. In the old generation, pause times roughly correlates with the size of the heap. For this reason, the heap should ideally be sized and tuned such that transaction and query state never makes it to the old generation.

The heap size is configured with the `server.memory.heap.initial_size` (in MBs) setting in the *neo4j.conf* file. The initial size of the heap is specified by the `server.memory.heap.initial_size` setting, or with the `-Xms???m` flag, or chosen heuristically by the JVM itself if left unspecified. The JVM will automatically grow the heap as needed, up to the maximum size. The growing of the heap requires a full garbage collection cycle. It is recommended to set the initial heap size and the maximum heap size to the same value. This way the pause that happens when the garbage collector grows the heap can be avoided.

If the new generation is too small, short-lived objects may be moved to the old generation too soon. This is called premature promotion and will slow the database down by increasing the frequency of old generation garbage collection cycles. If the new generation is too big, the garbage collector may decide that the old generation does not have enough space to fit all the objects it expects to promote from the new to the old generation. This turns new generation garbage collection cycles into old generation garbage collection cycles, again slowing the database down. Running more concurrent threads means that more allocations can take place in a given span of time, in turn increasing the pressure on the new generation in particular.

The *Compressed OOPs* feature in the JVM allows object references to be compressed to use only 32 bits. The feature saves a lot of memory but is only available for heaps up to 32

GB. The maximum applicable size varies from platform and JVM version. The `-XX:+UseCompressedOops` option can be used to verify whether the system can use the *Compressed OOPs* feature. If it cannot, this will will be logged in the default process output stream.

How to tune the specific garbage collection algorithm depends on both the JVM version and the workload. It is recommended to test the garbage collection settings under realistic load for days or weeks. Problems like heap fragmentation can take a long time to surface.

To gain good performance, these are the things to look into first:

- Make sure the JVM is not spending too much time performing garbage collection. The goal is to have a large enough heap to make sure that heavy/peak load will not result in so called GC-trashing. Performance can drop as much as two orders of magnitude when GC-trashing happens. Having too large heap may also hurt performance so you may have to try some different heap sizes.
- Neo4j needs enough heap memory for the transaction state and query processing, plus some head-room for the garbage collector. As heap memory requirements are so workload-dependent, it is common to see heap memory configurations from 1 GB, up to 32 GB.

Edit the following properties:

Table 577. neo4j.conf JVM tuning properties

Property Name	Meaning
`server.memory.heap.initial_size`	initial heap size (in MB)
`server.memory.heap.max_size`	maximum heap size (in MB)
`server.jvm.additional`	additional literal JVM parameter

Bolt thread pool configuration

The Bolt connector is backed by a thread pool on the server side, whereas the thread pool is constructed as part of the server startup process. This page discusses the thread pool infrastructure and how it can be configured.

How thread pooling works

The Bolt thread pool has a minimum and a maximum capacity. It starts with a minimum number of threads available, and grows up to the maximum count depending on the workload. Threads that sit idle for longer than a specified time period are stopped and removed from the pool in order to free up resources. However, the size of the pool will never go below the minimum.

Each connection being established is assigned to the connector's thread pool. Idle connections do not consume any resources on the server side, and they are monitored against messages arriving from the client. Each message arriving on a connection triggers the scheduling of a connection on an available thread in the thread pool. If all the available threads are busy, and there is still space to grow, a new thread is created and the connection is handed over to it for processing. If the pool capacity is filled up, and no

threads are available to process, the job submission is rejected and a failure message is generated to notify the client of the problem.

The default values assigned to the Bolt thread pool will fit most workloads, so it is generally not necessary to configure the connection pool explicitly. If the maximum pool size is set too low, an exception will be thrown with an error message indicating that there are no available threads to serve. The message will also be written to neo4j.log.

Any connection with an active explicit, or implicit, transaction will stick to the thread that starts the transaction, and will not return that thread to the pool until the transaction is closed. Therefore, in applications that are making use of explicit transactions, it is important to close the transactions appropriately. To learn more about transactions, refer to the Neo4j Driver manuals.

Configuration options

The following configuration options are available for configuring the Bolt connector:

Table 578. Thread pool options

Option name	Default	Description
`server.bolt.thread_pool_min_size`	`5`	The minimum number of threads that will always be up even if they are idle.
`server.bolt.thread_pool_max_size`	`400`	The maximum number of threads that will be created by the thread pool.
`server.bolt.thread_pool_keep_alive`	`5m`	The duration that the thread pool will wait before killing an idle thread from the pool. However, the number of threads will never go below `server.bolt.thread_pool_min_size`.

How to size your Bolt thread pool

Select values for thread pool sizing based on your workload. Since each active transaction will borrow a thread from the pool until the transaction is closed, it is basically the minimum and maximum active transaction at any given time that determine the values for pool configuration options. You can use the monitoring capabilities (see Monitoring) of the database to discover more about your workload.

Configure `server.bolt.thread_pool_min_size` based on your minimum or average workload. Since there will always be this many amount of threads in the thread pool, sticking with lower values may be more resource-friendly than having too many idle threads waiting for job submissions.

Configure `server.bolt.thread_pool_max_size` based on your maximum workload. This should basically be set after the maximum number of active transactions that is expected on the server. You should also account for non-transaction operations that will take place on the thread pool, such as connection and

disconnection of clients.

Example 142. Configure the thread pool for a Bolt connector

In this example, you configure the Bolt thread pool to be of minimum size `5`, maximum size `100`, and have a keep-alive time of `10 minutes`.

```
server.bolt.thread_pool_min_size=5
server.bolt.thread_pool_max_size=100
server.bolt.thread_pool_keep_alive=10m
```

Linux file system tuning

Databases often produce many small and random reads when querying data, and few sequential writes when committing changes. For maximum performance, it is recommended to store database and transaction logs on separate physical devices. This page covers Neo4j I/O behavior and how to optimize for operations on disk.

It is recommended to disable file and directory access time updates by setting the `noatime,nodiratime` mount options in *fstab*, or when issuing the disk mount command. This way, the file system will not have to issue writes that update this meta-data, thus improving write performance.

Since databases can put a high and consistent load on a storage system for a long time, it is recommended to use a file system that has good aging characteristics. The EXT4 and XFS file systems are both supported.

Tip:

While EXT4 and XFS file systems are both supported, XFS can provide marginal performance benefits (up to 10%) in some workloads, but uses more disk space (up to 2x) compared to EXT4. Therefore, EXT4 is generally recommended.

A high read and write I/O load can also degrade SSD performance over time. The first line of defense against SSD wear is to ensure that the working dataset fits in RAM. A database with a high write workload will, however, still cause wear on SSDs. The simplest way to combat this is to over-provision; use SSDs that are at least 20% larger than you strictly need them to be.

Warning:

Neo4j does not recommend and support the usage of NFS or NAS as database storage.

Disks, RAM and other tips

As with any persistence solution, performance depends a lot on the persistence media used. In general, the faster storage you have, and the more of your data you can fit in RAM, the better performance you will get. This page provides an overview of performance considerations for disk and RAM when running Neo4j.

Storage

There are many performance characteristics to consider for your storage solutions. The performance can vary hugely in orders of magnitude. Generally, having all your data in RAM achieves maximum performance.

If you have multiple disks or persistence media available, it may be a good idea to divide the store files and transaction logs across those disks. Keeping the store files on disks with low seek time can do wonders for read operations.

Use tools like `dstat` or `vmstat` to gather information when your application is running. If the swap or paging numbers are high, that is a sign that the database does not quite fit in memory. In this case, database access can have high latencies.

Note:

To achieve maximum performance, it is recommended to provide Neo4j with as much RAM as possible to avoid hitting the disk.

Page cache

When Neo4j starts up, its page cache is empty and needs to warm up. The pages, and their graph data contents, are loaded into memory on demand as queries need them. This can take a while, especially for large stores. It is not uncommon to see a long period with many blocks being read from the drive, and high IO wait times. This will show up in the page cache metrics as an initial spike in page faults. The page fault spike is then followed by a gradual decline of page fault activity, as the probability of queries needing a page that is not yet in memory drops.

Active page cache warmup

Neo4j Enterprise Edition has a feature called *active page cache warmup*, which is enabled by default via the `db.memory.pagecache.warmup.enable` configuration setting.

How it works

It shortens the page fault spike and makes the page cache warm up faster. This is done by periodically recording *cache profiles* of the store files while the database is running. These profiles contain information about what data is and is not in memory and are stored in the *data/databases/mydatabase/profiles* directory. When Neo4j is restarted next time, it looks for these cache profiles and loads the same data that was in memory when the profile was created. The profiles are also copied as part of the online backup and cluster store-copy operations and help warm up new databases that join a cluster.

The setting should remain enabled for most scenarios. However, when the workload changes after the database restarts, the setting can be disabled to avoid spending time fetching data that will be directly evicted.

Configuration options

Load the entire database into memory

It is also possible to configure `db.memory.pagecache.warmup.preload` to load the entire database data into memory. This is useful when the size of the database store is smaller than the available memory for the page cache. When enabled, it disables warmup by profile and prefetches data into the page cache as part of the startup.

Load specified files into memory

The files that you want to prefetched can be filtered using the `db.memory.pagecache.warmup.preload.allowlist` setting. It takes a regular expression as a value to match the files.

Example 143. Load only the nodes and relationships

For example, if you want to load only the nodes and relationships, you can use the regex `.*(node|relationship).*` to match the name of the store files. The *active page cache warmup* will prefetch the content of the following files:

```
neostore.nodestore.db
neostore.nodestore.db.id
neostore.nodestore.db.labels
neostore.nodestore.db.labels.id
neostore.relationshipgroupstore.db
neostore.relationshipgroupstore.db.id
neostore.relationshipstore.db
neostore.relationshipstore.db.id
neostore.relationshiptypestore.db
neostore.relationshiptypestore.db.id
neostore.relationshiptypestore.db.names
Neostore.relationshiptypestore.db.names.id
```

And can be verified using unix `grep`:

```
ls neo4j/ | grep -E '.*(node|relationship).*'
```

Configure the profile frequency for the page cache

The profile frequency is the rate at which the profiles are re-generated. More frequent means more accurate. A profile contains information about those parts of the files that are currently loaded into memory. By default, it is set to `db.memory.pagecache.warmup.profile.interval=1m`. It takes some time to generate these profiles, and therefore `1m` is a good interval. If the workload is very stable, then the profile will not change much. Accordingly, if the workload changes often, the profile will thus often become outdated.

Checkpoint IOPS limit

Neo4j flushes its page cache in the background as part of its checkpoint process. This will show up as a period of elevated write IO activity. If the database is serving a write-heavy workload, the checkpoint can slow the database down by reducing the IO bandwidth that is available to query processing. Running the database on a fast SSD, which can service a lot of random IOs, significantly reduces this problem. If a fast SSD is not available in your environment, or if it is insufficient, then an artificial IOPS limit can be placed on the checkpoint process. The `db.checkpoint.iops.limit` restricts the IO bandwidth that the checkpoint

process is allowed to use. Each IO is, in the case of the checkpoint process, an 8 KiB write. An IOPS limit of 600, for instance, would thus only allow the checkpoint process to write at a rate of roughly 5 MiB per second. This will, on the other hand, make checkpoints take longer to complete. A longer time between checkpoints can cause more transaction log data to accumulate, and can lengthen recovery times. See the Checkpointing and log pruning section for more details on the relationship between checkpoints and log pruning. The IOPS limit can be changed at runtime, making it possible to tune it until you have the right balance between IO usage and checkpoint time.

Statistics and execution plans

When a Cypher query is issued, it gets compiled to an execution plan that can run and answer the query. The Cypher query engine uses the available information about the database, such as schema information about which indexes and constraints exist in the database. This page describes how to configure the Neo4j statistics collection and the query replanning in the Cypher query engine.

Note:

Neo4j also uses statistical information about the database to optimize the execution plan. For more information, see Cypher Manual → Execution plans.

Configure statistics collection

The Cypher query planner depends on accurate statistics to create efficient plans. Therefore, these statistics are kept up-to-date as the database evolves.

For each database in the DBMS, Neo4j collects the following statistical information and keeps it up-to-date:

For graph entities

- The number of nodes with a certain label.
- The number of relationships by type.
- The number of relationships by type between nodes with a specific label.

These numbers are updated whenever you set or remove a label from a node.

For database schema

- Selectivity per index.

To produce a selectivity number, Neo4j runs a full index scan in the background. Because this could potentially be a very time-consuming operation, a full index scan is triggered only when the changed data reaches a specified threshold.

Automatic statistics collection

You can control whether and how often statistics are collected automatically by configuring the following settings:

Parameter name	Default value	Description
`db.index_sampling.background_enabled`	`true`	Enable the automatic (background) index sampling.
`db.index_sampling.update_percentage`	`5`	Percentage of index updates of total index size required before sampling of a given index is triggered.

Manual statistics collection

You can manually trigger index resampling by using the built-in procedures `db.resampleIndex()` and `db.resampleOutdatedIndexes()`.

`db.resampleIndex()`

Trigger resampling of a specified index.

```
CALL db.resampleIndex("indexName")
```

`db.resampleOutdatedIndexes()`

Trigger resampling of all outdated indexes.

```
CALL db.resampleOutdatedIndexes()
```

Configure the replanning of execution plans

Execution plans are cached and are not replanned until the statistical information used to produce the plan changes.

Automatic replanning

You can control how sensitive the replanning should be to database updates by configuring the following settings:

Parameter name	Default value	Description
`dbms.cypher.statistics_divergence_threshold`	`0.75`	The threshold for statistics above which a plan is considered stale. When the changes to the underlying statistics of an execution plan meet the specified threshold, the plan is considered stale and is replanned. Change is calculated as `abs(a-b)/max(a,b)`. This means that a value of `0.75` requires the database to approximately quadruple in size before replanning occurs. A value of `0` means that the query is replanned as soon as there is a change in the statistics and the replan interval elapses.

Parameter name	Default value	Description
`dbms.cypher.min_replan_interval`	`10s`	The minimum amount of time between two query replanning executions. After this time, the graph statistics are evaluated, and if they have changed more than the value set in `dbms.cypher.statistics_divergence_threshold`, the query is replanned. Each time the statistics are evaluated, the divergence threshold is reduced until it reaches 10% after about 7h. This ensures that even moderately changing databases see query replanning after a sufficiently long time interval.

Manual replanning

You can manually force the database to replan the execution plans that are already in the cache by using the following built-in procedures:

`db.clearQueryCaches()`

Clear all query caches. Does not change the database statistics.

```
CALL db.clearQueryCaches()
```

`db.prepareForReplanning()`

Completely recalculates all database statistics to be used for any subsequent query planning.

The procedure triggers an index resampling, waits for it to complete, and clears all query caches. Afterwards, queries are planned based on the latest database statistics.

```
CALL db.prepareForReplanning()
```

You can use Cypher replanning to specify whether you want to force a replan, even if the plan is valid according to the planning rules, or skip replanning entirely should you wish to use a valid plan that already exists.

For more information, see:

- Cypher manual → Cypher replanning
- Cypher manual → Execution plans
- Procedures

Space reuse

Neo4j uses logical deletes to remove data from the database to achieve maximum performance and scalability. A logical delete means that all relevant records are marked as deleted, but the space they occupy is not immediately returned to the operating system. Instead, it is subsequently reused by the

transactions *creating* data.

Marking a record as deleted requires writing a record update command to the transaction log files, as when something is created or updated. Therefore, when deleting large amounts of data, this leads to a storage usage growth of that particular database, because Neo4j writes records for all deleted nodes, their properties, and relationships to the transaction log.

> **Note:**
>
> Keep in mind that when doing `DETACH DELETE` on many nodes, those deletes can take up more space in the in-memory transaction state and the transaction log than you might expect.

Transactions are eventually pruned out of the transaction log files, bringing the storage usage of the log back down to the expected level. The store files, on the other hand, do not shrink when data is deleted. The space that the deleted records take up is kept in the store files. Until the space is reused, the store files are sparse and fragmented, but the performance impact of this is usually minimal.

ID files

Neo4j uses *.id* files for managing the space that can be reused. These files contain the set of IDs for all the deleted records in their respective files. The ID of the record uniquely identifies it within the store file. For instance, depending on the store format, the IDs of all deleted nodes are contained in `neostore.nodestore.db.id` or `block.x1.db.id`.

These *.id* files are maintained as part of the write transactions that interact with them. When a write transaction commits a deletion, the record's ID is buffered in memory. The buffer keeps track of all overlapping unfinished transactions. When they complete, the ID becomes available for reuse.

The buffered IDs are flushed to the *.id* files as part of the checkpointing. Concurrently, the *.id* file changes (the ID additions and removals) are inferred from the transaction commands. This way, the recovery process ensures that the *.id* files are always in-sync with their store files. The same process also ensures that clustered databases have precise and transactional space reuse.

> **Warning:**
>
> If you want to shrink the size of your database, do not delete the *.id* files. The store files must *only* be modified by the Neo4j database and the `neo4j-admin` tools.

Reclaim unused space

You can use the `neo4j-admin database copy` command to create a defragmented copy of your database. The `copy` command creates and entirely new and independent database. If you want to run that database in a cluster, you have to re-seed the existing cluster, or seed a new cluster from that copy.

Example 144. Example of database compaction using `neo4j-admin database copy`

The following is a detailed example on how to check your database store usage and how to reclaim space.

Let's use the Cypher Shell command-line tool to add 100k nodes and then see how much store they occupy.

1. In a running Neo4j standalone instance, log in to the Cypher Shell command-line tool with your credentials.

```
bin/cypher-shell -u neo4j -p <password>
```

```
Connected to Neo4j at neo4j://localhost:7687 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
```

2. Add 100k nodes to the `neo4j` database using the following command:

```
neo4j@neo4j> foreach (x in range (1,100000) | create (n:testnode1 {id:x}));
```

```
0 rows available after 1071 ms, consumed after another 0 ms
Added 100000 nodes, Set 100000 properties, Added 100000 labels
```

3. Check the allocated ID range:

```
neo4j@neo4j> MATCH (n:testnode1) RETURN ID(n) as ID order by ID limit 5;
```

```
+----+
| ID |
+----+
| 0  |
| 1  |
| 2  |
| 3  |
| 4  |
+----+

5 rows available after 171 ms, consumed after another 84 ms
```

4. Run `call db.checkpoint()` procedure to force a checkpoint.

```
neo4j@neo4j> call db.checkpoint();
```

```
+----------------------------------+
| success | message                |
+----------------------------------+
| TRUE    | "Checkpoint completed." |
+----------------------------------+

1 row available after 18 ms, consumed after another 407 ms
```

5. In Neo4j Browser, run `:sysinfo` to check the total store size of `neo4j`.

   The reported output for the store size is 791.92 KiB, ID Allocation: Node ID 100000, Property ID 100000.

6. Delete the above created nodes.

```
neo4j@neo4j> Match (n) detach delete n;
```

7. Run `call db.checkpoint()` procedure again.

```
neo4j@neo4j> call db.checkpoint();
```

```
+-----------------------------------+
| success | message                 |
+-----------------------------------+
| TRUE    | "Checkpoint completed." |
+-----------------------------------+

1 row available after 18 ms, consumed after another 407 ms
```

8. In Neo4j Browser, run `:sysinfo` to check the total store size of `neo4j`.

   The reported output for the store size is 31.01 MiB, ID Allocation: Node ID 100000, Property ID 100000.

> By default, a checkpoint flushes any cached updates in pagecache to store files. Thus, the allocated IDs remain unchanged, and the store size increases or does not alter (if the instance restarts) despite the deletion. In a production database, where numerous load/deletes are frequently performed, the result is a significant unused space occupied by store files.

To reclaim that unused space, you can use the neo4j-admin database copy command to create a defragmented copy of your database. Use the `system` database and stop the `neo4j` database before running the command.

1. Invoke the `neo4j-admin database copy` command to create a copy of your `neo4j` database.

```
bin/neo4j-admin database copy neo4j neo4jcopy1 --compact-node-store --verbose
```

```
Starting to copy store, output will be saved to: $neo4j_home/logs/neo4j-admin-copy-2020-11-
04.11.30.57.log
2020-10-23 11:40:00.749+0000 INFO [StoreCopy] ### Copy Data ###
2020-10-23 11:40:00.750+0000 INFO [StoreCopy] Source: $neo4j_home/data/databases/neo4j (page
cache 8m) (page cache 8m)
2020-10-23 11:40:00.750+0000 INFO [StoreCopy] Target: $neo4j_home/data/databases/neo4jcopy1 (page
cache 8m)
2020-10-23 11:40:00.750+0000 INFO [StoreCopy] Empty database created, will start importing
readable data from the source.
2020-10-23 11:40:02.397+0000 INFO [o.n.i.b.ImportLogic] Import starting
Nodes, started 2020-11-04 11:31:00.088+0000
[*Nodes:?? 7.969MiB-------------------------------------------------------------------------------]
100K Δ 100K
Done in 632ms
```

```
Prepare node index, started 2020-11-04 11:31:00.735+0000
[*DETECT:7.969MiB-------------------------------------------------------------------------------]
0 Δ    0
Done in 79ms
Relationships, started 2020-11-04 11:31:00.819+0000
[*Relationships:?? 7.969MiB---------------------------------------------------------------------]
0 Δ    0
Done in 37ms
Node Degrees, started 2020-11-04 11:31:01.162+0000
[*>:??------------------------------------------------------------------------------------------]
0 Δ    0
Done in 12ms
Relationship --> Relationship 1/1, started 2020-11-04 11:31:01.207+0000
[*>:??------------------------------------------------------------------------------------------]
0 Δ    0
Done in 0ms
RelationshipGroup 1/1, started 2020-11-04 11:31:01.232+0000
[*>:??------------------------------------------------------------------------------------------]
0 Δ    0
Done in 10ms
Node --> Relationship, started 2020-11-04 11:31:01.245+0000
[*>:??------------------------------------------------------------------------------------------]
0 Δ    0
Done in 10ms
Relationship <-- Relationship 1/1, started 2020-11-04 11:31:01.287+0000
[*>:??------------------------------------------------------------------------------------------]
0 Δ    0
Done in 0ms
Count groups, started 2020-11-04 11:31:01.549+0000
[*>:??------------------------------------------------------------------------------------------]
0 Δ    0
Done in 0ms
Node --> Group, started 2020-11-04 11:31:01.579+0000
[*>:??------------------------------------------------------------------------------------------]
0 Δ    0
Done in 1ms
Node counts and label index build, started 2020-11-04 11:31:01.986+0000
[*>:??------------------------------------------------------------------------------------------]
0 Δ    0
Done in 11ms
Relationship counts, started 2020-11-04 11:31:02.034+0000
[*>:??------------------------------------------------------------------------------------------]
0 Δ    0
Done in 0ms

IMPORT DONE in 3s 345ms.
Imported:
  0 nodes
  0 relationships
  0 properties
Peak memory usage: 7.969MiB
2020-11-04 11:31:02.835+0000 INFO [o.n.i.b.ImportLogic] Import completed successfully, took 3s
345ms. Imported:
  0 nodes
  0 relationships
  0 properties
2020-11-04 11:31:03.330+0000 INFO [StoreCopy] Import summary: Copying of 100704 records took 5
seconds (20140 rec/s). Unused Records 100704 (100%) Removed Records 0 (0%)
2020-11-04 11:31:03.330+0000 INFO [StoreCopy] ### Extracting schema ###
2020-11-04 11:31:03.330+0000 INFO [StoreCopy] Trying to extract schema...
2020-11-04 11:31:03.338+0000 INFO [StoreCopy] ... found 0 schema definitions.
```

The example resulted in a compact and consistent store (any inconsistent nodes, properties, relationships are not copied over to the newly created store).

2. Use the `system` database and create the `neo4jcopy1` database.

```
neo4j@system> create database neo4jcopy1;
```

```
0 rows available after 60 ms, consumed after another 0 ms
```

3. Verify that the `neo4jcopy1` database is online.

```
neo4j@system> show databases;
```

```
+-----------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
| name         | type       | aliases | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage | default | home  | constituents |
+-----------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
| "neo4j"      | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   |
"offline"       | "offline"     | ""            | TRUE    | TRUE  | []           |
| "neo4jcopy1" | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   |
"online"        | "online"      | ""            | FALSE   | FALSE | []           |
| "system"     | "system"   | []      | "read-write" | "localhost:7687" | "primary" | TRUE   |
"online"        | "online"      | ""            | FALSE   | FALSE | []           |
+-----------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+

3 rows available after 2 ms, consumed after another 1 ms
```

4. In Neo4j Browser, run `:sysinfo` to check the total store size of `neo4jcopy1`.

   The reported output for the store size after the compaction is 800.68 KiB, ID Allocation: Node ID 0, Property ID 0.

Monitoring

Neo4j provides mechanisms for continuous analysis through the output of metrics as well as the inspection and management of currently-executing queries.

Logs can be harvested for continuous analysis, or for specific investigations. Facilities are available for producing security event logs as well as query logs. The query management functionality is provided for specific investigations into query performance. Monitoring features are also provided for ad-hoc analysis of a Causal Cluster.

This chapter describes the following:

Logging

Neo4j provides logs for monitoring purposes. The logs are stored in the *logs* directory. If you want to use a custom directory, set the path to that directory in the *neo4j.conf* file by using the parameter `server.directories.logs`.

Log files

The following table describes the log files that are available in Neo4j and their default configuration. Which logs are enabled or disabled is configured in the *neo4j.conf* file.

Table 579. Neo4j log files

Filename	Description	Configuration	Default
neo4j.log	Logs general information about Neo4j. For Debian and RPM packages run `journalctl --unit=neo4j`.		`true`
debug.log	Log information required by Neo4j Customer Support to investigate problems. It is highly recommended to keep it enabled and not to alter the *debug.log* format. If you require the *debug.log* messages in a different format, create an additional Appender.	`server.logs.debug.enabled`	`true`
http.log	Logs information about the HTTP API.	`dbms.logs.http.enabled`	`false`
gc.log	Logs information provided by the JVM. For details, see Configure the garbage collection log.	`server.logs.gc.enabled`	`false`
query.log	[.label-baltic]# Enterprise #Logs information about executed queries that run longer than a specified threshold. Must be one of `INFO`, `VERBOSE`, or `OFF`. • `INFO` logs queries longer than a given threshold. • `VERBOSE` logs the start and end of all queries. • `OFF` disables the *query.log*. For details, see Configure the query log.	`db.logs.query.enabled` and `db.logs.query.threshold`	`VERBOSE`
security.log	Enterprise Logs information about security events.	`dbms.security.auth_enabled`	`true`
service-out.log	[.label-earth]# Windows #Log of the console output when installing or running the Windows service.		
service-error.log	[.label-earth]# Windows #Logs information about errors encountered when installing or running the Windows service.		

Default logging configuration

As of Neo4j 5.0, Neo4j uses Log4j 2 for logging. The logging configuration is located in the *conf* directory and consists of two files:

- *user-log.xml* — provides configuration for *neo4j.log*.
- *server-logs.xml* — provides configuration for *debug.log*, *http.log*, *query.log*, and *security.log*.

The *gc.log* is handled by the Java Virtual Machine(JVM) and is configured using the JVM parameters. For details, see Configure the garbage collection log.

> **Tip:**
>
> If you want to use a custom directory for your Log4j configuration files, set the new paths to your XML files in the *neo4j.conf* file by using the parameters `server.logs.user.config` and `server.logs.config`.

Each configuration file comprises two main elements: *Appenders* and *Loggers*:

Within the Appenders element, you can define

- The output locations, for example, a file, the console, the network socket, etc.
- How the output is formatted, for example, plain text, JSON, CSV, etc.
- When the files are rolled and how many files to keep as history before deleting them.
- If and what log events are to be published and how.

Within the Loggers element, you can define

- Which log events are to be captured and sent to which *Appenders*.
- The log level of the log events to be captured.
- If the log events are to be forwarded to other *Loggers*.

For more details, see Log4j 2 official documentation on configuration.

The following example shows the default configuration of the *user-logs.xml* file.

Default user-log.xml configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<!--

    Copyright (c) "Neo4j"
    Neo4j Sweden AB [http://neo4j.com]
    This file is a commercial add-on to Neo4j Enterprise Edition.

-->
<!--
    This is a log4j 2 configuration file that provides maximum flexibility.

    All configuration values can be queried with the lookup prefix "config:". You can, for example, resolve
    the path to your neo4j home directory with ${config:dbms.directories.neo4j_home}.

    Please consult https://logging.apache.org/log4j/2.x/manual/configuration.html for instructions and
    available configuration options.
-->
<Configuration status="ERROR" monitorInterval="30" packages="org.neo4j.logging.log4j"> \

    <Appenders> \
        <RollingRandomAccessFile name="Neo4jLog"
fileName="${config:server.directories.logs}/neo4j.log"

filePattern="$${config:server.directories.logs}/neo4j.log.%02i">
            <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSSZ}{GMT+0} %-5p %m%n"/>
            <Policies> \
                <SizeBasedTriggeringPolicy size="20 MB"/> \
            </Policies>
            <DefaultRolloverStrategy fileIndex="min" max="7"/> \
        </RollingRandomAccessFile>

```

```
        <!-- Only used by "neo4j console", will be ignored otherwise -->
        <Console name="ConsoleAppender" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSSZ}{GMT+0} %-5p %m%n"/>
        </Console>
    </Appenders>

    <Loggers> \
        <!-- Log level for the neo4j log. One of DEBUG, INFO, WARN, ERROR or OFF -->
        <Root level="INFO"> \
            <AppenderRef ref="Neo4jLog"/>
            <AppenderRef ref="ConsoleAppender"/>
        </Root>
    </Loggers>

</Configuration>
```

The following example shows the default configuration of the *server-logs.xml* file.

Default server-logs.xml configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<!--

    Copyright (c) "Neo4j"
    Neo4j Sweden AB [http://neo4j.com]
    This file is a commercial add-on to Neo4j Enterprise Edition.

-->
<!--
    This is a log4j 2 configuration file.

    Please do not alter the original debug.log.
    Changing the debug.log format can interfere with Neo4j's ability to offer customer support and invalidate your support warranty.

    If you require the debug.log messages in a different format, you can copy the Appender and change the filename.
    See Neo4j documentation for details.

    All configuration values can be queried with the lookup prefix "config:". You can, for example, resolve
    the path to your neo4j home directory with ${config:dbms.directories.neo4j_home}.

    Please consult https://logging.apache.org/log4j/2.x/manual/configuration.html for instructions and
    available configuration options.
-->
<Configuration status="ERROR" monitorInterval="30" packages="org.neo4j.logging.log4j"> \
    <Appenders> \
        <!-- Neo4j debug.log,do not change. Required by Neo4j customer support. -->
        <RollingRandomAccessFile name="DebugLog" fileName="${config:server.directories.logs}/debug.log" \

filePattern="$${config:server.directories.logs}/debug.log.%02i"> \
            <Neo4jDebugLogLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSSZ}{GMT+0} %-5p [%c{1.}] %m%n"/> \
            <Policies> \
                <SizeBasedTriggeringPolicy size="20 MB"/> \
            </Policies>
            <DefaultRolloverStrategy fileIndex="min" max="7"/> \
        </RollingRandomAccessFile>

        <RollingRandomAccessFile name="HttpLog" fileName="${config:server.directories.logs}/http.log"

filePattern="$${config:server.directories.logs}/http.log.%02i">
            <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSSZ}{GMT+0} %-5p %m%n"/>
            <Policies>
                <SizeBasedTriggeringPolicy size="20 MB"/>
            </Policies>
            <DefaultRolloverStrategy fileIndex="min" max="5"/>
        </RollingRandomAccessFile>
```

```
44
45          <RollingRandomAccessFile name="QueryLog"
    fileName="${config:server.directories.logs}/query.log"
46
    filePattern="$${config:server.directories.logs}/query.log.%02i">
47              <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSSZ}{GMT+0} %-5p %m%n"/>
48              <Policies>
49                  <SizeBasedTriggeringPolicy size="20 MB"/>
50              </Policies>
51              <DefaultRolloverStrategy fileIndex="min" max="7"/>
52          </RollingRandomAccessFile>
53
54          <RollingRandomAccessFile name="SecurityLog"
    fileName="${config:server.directories.logs}/security.log"
55
    filePattern="$${config:server.directories.logs}/security.log.%02i">
56              <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSSZ}{GMT+0} %-5p %m%n"/>
57              <Policies>
58                  <SizeBasedTriggeringPolicy size="20 MB"/>
59              </Policies>
60              <DefaultRolloverStrategy fileIndex="min" max="7"/>
61          </RollingRandomAccessFile>
62      </Appenders>
63
64      <Loggers> \
65          <!-- Log levels. One of DEBUG, INFO, WARN, ERROR or OFF -->
66
67          <!-- The debug log is used as the root logger to catch everything -->
68          <Root level="INFO"> \
69              <AppenderRef ref="DebugLog"/> <!-- Keep this -->
70          </Root>
71
72          <!-- The query log, must be named "QueryLogger" -->
73          <Logger name="QueryLogger" level="INFO" additivity="false"> \
74              <AppenderRef ref="QueryLog"/>
75          </Logger>
76
77          <!-- The http request log, must be named "HttpLogger" -->
78          <Logger name="HttpLogger" level="INFO" additivity="false">
79              <AppenderRef ref="HttpLog"/>
80          </Logger>
81
82          <!-- The security log, must be named "SecurityLogger" -->
83          <Logger name="SecurityLogger" level="INFO" additivity="false">
84              <AppenderRef ref="SecurityLog"/>
85          </Logger>
86      </Loggers>
87  </Configuration>
```

Configuration tag with a `monitorInterval` of 30 seconds and a package namespace of `org.neo4j.logging.log4j`.
The monitor interval tells Log4j to periodically check the XML file for changes and reload the file if a change is detected.
The package namespace gives access to the Neo4j configuration lookup with `${config:<setting>}`.

Appenders are used to define where the log messages are written. The `name` of the appender must be unique as it is used by the logger when referencing the appender. The Neo4j default appenders write to *debug.log*, *http.log*, *query.log*, and *security.log*.

By default, Neo4j uses the `<RollingRandomAccessFile>` appender as it is very performant because it always writes to a buffer. However, if the server crashes, the last log messages might be lost. If that is not acceptable for you, use the `<RollingFile>` appender instead. See Appenders for more information.

`filePattern` specifies a file pattern to be used when the file is rolled. The pattern renames the files to *debug.log.01* and *http.log.01* when they reach the defined trigger.

`PatternLayout` defines the layout for the appender, in this case with the `GMT+2` timezone. See Log layouts for more information.

The `Policies` element defines when the files are rolled and how many files to keep as history before they are deleted.

The `SizeBasedTriggeringPolicy` defines when the files are rolled. In this case, when the size of the files reaches 20 MB, the files are renamed according to the `filePattern`, and the log files start over. In Neo4j 4.0, this was configured with the parameter `dbms.logs.user.rotation.size`.

The `DefaultRolloverStrategy` defines how many files to keep as history.
The `fileIndex=min` implies that the minimum/the lowest number is the most recent one.
The `max` attribute defines the number of files to keep as history before they are deleted, in this case, 7 files. In Neo4j 4.0, this was configured with the parameter `dbms.logs.user.rotation.keep_number`.

Loggers are used to define the log level and which appender to use for the log messages. The loggers are referenced by the `name` attribute. See Loggers for more information.

The root logger is a "catch-all" logger that catches everything that is not caught by the other loggers and sends it to the appender(s) specified in the `AppenderRef` element(s). The root logger is referenced by the `Root` element. It can be set to `DEBUG`, `INFO`, `WARN`, `ERROR`, or `OFF`. The default log level is `INFO`.

You can also define custom loggers to catch specific log events and send them to the appender(s) specified in the `AppenderRef` element(s). For example, the `QueryLogger` logger (configured in *server-logs.xml*) is used to catch log events with a log level of `INFO` or above and send them to the `QueryLog` appender.
The `additivity="false"` is set to fully consume the log event and not send it to the root logger.
If `additivity="true"` is set, which is the default, the log event is also sent to the root logger.

Advanced logging configuration

The default logging configuration is a good starting point, but you might want to customize it to your needs. The following sections describe some Log4j configuration elements and how to use them to customize the logging configuration. For additional information and more advanced customizations, such as filtering and extensions, see the Log4j official documentation on configuration.

> **Note:**
>
> If Neo4j is embedded in your Java applications, you must provide an implementation of `org.neo4j.logging.LogProvider`. Depending on how you set up your log provider, it uses or ignores the *user-logs.xml* file. For details, refer to Java Reference → Using Neo4j embedded in Java applications.

Appenders

All Log4j standard appenders are available in Neo4j. For more details, see the Log4j official documentation on appenders.

A few of the most common appenders are `<RollingRandomAccessFile>`, `<RollingFile>`, and `<Console>`.

`<RollingRandomAccessFile>` appender

The `<RollingRandomAccessFile>` is the default appender in Neo4j. It is very performant and has a low impact on the system because it always writes to a buffer. However, the log events might **not** be visible immediately, and if the server crashes, the last log messages might be lost. This appender is configured with the `filePattern` attribute, which specifies a file pattern to be used when the file is rolled. The pattern renames the files to *debug.log.01* and *http.log.01* when they reach the defined trigger.

The possible triggers are `SizeBasedTriggeringPolicy` and `TimeBasedTriggeringPolicy`. The `SizeBasedTriggeringPolicy` defines when the files are rolled, in this case, when the size of the files reaches 20 MB. The `TimeBasedTriggeringPolicy` defines when the files are rolled based on time, in this case, daily.

The `DefaultRolloverStrategy` defines how many files to keep as history and which file to use as the most recent one. The `fileIndex=min` implies that the minimum/the lowest number is the most recent one. The `max` attribute defines the number of files to keep as history before they are deleted, in this case, 7 files.

For more information, see Log4j official documentation on RollingRandomAccessFile Appender.

`<RollingFile>` appender

A `<RollingFile>` appender is very similar to `<RollingRandomAccessFile>` but it writes log events to a file. It rolls when certain criteria are met. A standard scheme is to keep one log file daily or roll a log file once a specific size is reached.

An example of a rolling file appender with one new log file each day

```
<RollingFile name="myLog" fileName="${config:server.directories.logs}/my.log"
                     filePattern="${config:server.directories.logs}/my-%d{yyyy-MM-dd}.log">
  <!-- Layout -->
  <Policies>
      <TimeBasedTriggeringPolicy />
  </Policies>
</RollingFile>
```

The rolling also supports the compression of rolled-out files. Adding one of `.gz`, `.zip`, `.bz2`, `.deflate`, or `.pack200` as a suffix to the `filePattern` attribute causes the file to be compressed with the appropriate compression scheme.

An example of a rolling file appender with zip compression

```
<RollingFile name="myLog" fileName="${config:server.directories.logs}/my.log"
                     filePattern="${config:server.directories.logs}/my.%i.log.zip">
  <!-- Layout -->
  <Policies>
      <SizeBasedTriggeringPolicy size="20 MB"/>
  </Policies>
</RollingFile>
```

`<Console>` appender

The console appender outputs log events to *stdout* or *stderr*. It is only used by the "neo4j console".

An example of a console appender

```
<Console name="console" target="SYSTEM_OUT"> <!-- or SYSTEM_ERR -->
  <PatternLayout pattern="%m%n"/>
```

```
</Console>
```

Log layouts

The log files can be written in a lot of different ways, referred to as layouts. Neo4j comes bundled with all the default layouts of Log4j 2, as well as a few Neo4j-specific ones. For more details on the default Log4j 2 layouts, see the Log4j official documentation.

`<PatternLayout>`

`<PatternLayout>` is the most common layout. It is a flexible layout configurable with a pattern string, which is specified in the `pattern` attribute. For example:

```
<PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSSZ}{GMT+0} %-5p [%c{1.}] %m%n"/>
```

The `pattern` consists of different converters that are prefixed with `%`. The converters are replaced with the corresponding value from the log event.

Table 580. Example pattern layout converters

Converter	Description
`%d{ date-pattern }{ timezone }`	Date of the log event. The time zone is optional. If omitted, the system time is used.
`%p`	The log level of the event. Can be `DEBUG`, `INFO`, `WARN`, or `ERROR`. Adding `-5` between the `%` symbol and the `p` pads the level to be exactly 5 characters long.
`%c`	The class where the log event originated from. Adding `{1.}` after compacts the package names, e.g. `org.apache.commons.Foo` will become `o.a.c.Foo`.
`%m`	The log message of the log event.
`%n`	System-specific new line.

For all available converters, consult the Log4j 2 Pattern Layout documentation.

`<Neo4jDebugLogLayout>`

The `<Neo4jDebugLogLayout>` layout is essentially the same as the `PatternLayout`. The main difference is that a header is injected at the start of the log file with diagnostic information useful for Neo4j developers. This layout should typically only be used for the *debug.log* file.

An example usage of the Neo4j debug log layout

```
<Neo4jDebugLogLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSSZ}{GMT+0} %-5p [%c{1.}] %m%n"/>
```

`<JsonTemplateLayout>`

The `<JsonTemplateLayout>` is equivalent to the pattern layout. For more information, see the Log4j official

documentation.

There are two ways of configuring the JSON template layout.

- You can specify a JSON event template file and the layout will use that. The JSON template file can be on the file system.

```
<JsonTemplateLayout eventTemplateUri="file://path/to/template.json"/>
```

- The JSON event template file can be embedded in the XML configuration:

```
<JsonTemplateLayout>
  <eventTemplate>
    <![CDATA[
      {
        "time": { "$resolver": "timestamp",
          "pattern": { "format": "yyyy-MM-dd HH:mm:ss.SSSZ", "timeZone": "UTC" }
        },
        "level": { "$resolver": "level", "field": "name" },
        "message": { "$resolver": "message" },
        "includeFullMap": { "$resolver": "map", "flatten": true },
        "stacktrace": { "$resolver": "exception", "field": "message" }
      }
    ]]>
  </eventTemplate>
</JsonTemplateLayout>
```

There are also a couple of built-in templates available from the classpath, for example:

```
<JsonTemplateLayout eventTemplateUri="classpath:org/neo4j/logging/StructuredJsonLayout.json"/>
```

Table 581. Available built-in templates

eventTemplateUri	Description
`classpath:org/neo4j/logging/StructuredJsonLayout.json`	Layout for structured log messages. Only applicable to the *query.log* and *security.log*.
`classpath:org/neo4j/logging/StructuredLayoutWithMessage.json`	Generic layout for logging JSON messages. Can be used for any log file.
`classpath:org/neo4j/logging/QueryLogJsonLayout.json`	Backward-compatible JSON layout that will match the Neo4j 4.x query log.
`classpath:LogstashJsonEventLayoutV1.json`	Logstash `json_event` pattern for Log4j
`classpath:GelfLayout.json`	Graylog Extended Log Format (GELF) payload specification with additional `_thread` and `_logger` fields.
`classpath:GcpLayout.json`	Google Cloud Platform structured logging with additional `_thread`, `_logger`, and `_exception` fields.
`classpath:JsonLayout.json`	Same layout as the less flexible `<JsonLayout>`.

Filters

You can also configure filters to determine if and what log events are published and how. For details, see

the Log4j official documentation.

Plugins

You can also add plugins to Log4j by dropping them in the *plugin* directory. For details, see the Log4j official documentation on plugins.

Loggers

Loggers forward log events to appenders. There can be an arbitrary number of `<Logger>` elements but only one `<Root>` logger element. Loggers have the possibility of being additive. An additive logger forwards a log event to its appender(s) and then passes the log event to the next matching logger. A non-additive logger forwards a log event to its appender(s) and then drops the event. The root logger is a special logger that matches everything, so if another logger does not pick up a log event, the root logger will. Therefore, it is best practice always to include a root logger so that no log events are missed.

Configuration of loggers

```
<Configuration>
    <!-- Appenders -->
    <Loggers>
        <Root level="WARN">
            <AppenderRef ref="DebugLog"/>
        </Root>

        <Logger name="HttpLogger" level="INFO" additivity="false">
            <AppenderRef ref="HttpLog"/>
        </Logger>
    </Loggers>
</Configuration>
```

A logger has a `level` that filters log events. A level can also include levels of different severity. For example, a logger with `level="INFO"` forwards log events with `INFO`, `WARN`, and `ERROR`. A logger with `level="WARN"` only logs `WARN` and `ERROR` events.

The following table lists all log levels raised by Neo4j and their severity level:

Table 582. Log levels

Message type	Severity level	Description
`DEBUG`	Low severity	Report details on the raised errors and possible solutions.
`INFO`	Low severity	Report status information and errors that are not severe.
`WARN`	Low severity	Report errors that need attention but are not severe.
`ERROR`	High severity	Reports errors that prevent the Neo4j server from running and must be addressed immediately.

For more details on loggers, see the Log4j official documentation → Configuring Loggers.

Configure the garbage collection log

The garbage collection log, or GC log for short, is special and cannot be configured with Log4j 2. The GC log is handled by the Java Virtual Machine(JVM) and must be passed directly to the command line. To simplify this process, Neo4j exposes the following settings in *neo4j.conf*:

Table 583. Garbage collection log configurations

The *garbage collection log* configuration	Default value	Description
`server.logs.gc.enabled`	`false`	Enable garbage collection logging.
`server.logs.gc.options`	`-Xlog:gc*,safepoint,age*=trace`	Garbage collection logging options. For available options, consult the documentation of the JVM distribution used.
`server.logs.gc.rotation.keep_number`	`5`	The maximum number of history files for the garbage collection log.
`server.logs.gc.rotation.size`	`20MB`	The threshold size for rotation of the garbage collection log.

Configure the security log

Neo4j provides security event logging that records all security events. The security log is enabled automatically when the configuration `dbms.security.auth_enabled` is set to `true` (which is the default). It ensures that all requests to Neo4j are authenticated. For additional configuration of the security log, see *<NEO4J_HOME>/conf/server-logs.xml*.

For native user management, the following actions are recorded:

- Login attempts — by default, both successful and unsuccessful logins are recorded.
- All administration commands run against the `system` database.
- Authorization failures from role-based access control.

If using LDAP as the authentication method, some cases of LDAP misconfiguration will also be logged, as well as the LDAP server communication events and failures.

If many programmatic interactions are expected, it is advised to disable the logging of successful logins by setting the `dbms.security.log_successful_authentication` parameter in the *neo4j.conf* file:

```
dbms.security.log_successful_authentication=false
```

The security log can use a JSON layout. To change the format, the layout for the `SecurityLogger` must be changed from using the `PatternLayout`:

```
<RollingRandomAccessFile name="SecurityLog" fileName="${config:server.directories.logs}/security.log"
                         filePattern="$${config:server.directories.logs}/security.log.%02i">
            <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSSZ}{GMT+0} %-5p %m%n"/>
            <Policies>
                <SizeBasedTriggeringPolicy size="20 MB"/>
            </Policies>
            <DefaultRolloverStrategy fileIndex="min" max="7"/>
```

```
</RollingRandomAccessFile>
```

to using the `JsonTemplateLayout`:

```
<RollingRandomAccessFile name="SecurityLog" fileName="${config:server.directories.logs}/security.log"
                         filePattern="$${config:server.directories.logs}/security.log.%02i">
            <JsonTemplateLayout eventTemplateUri=
"classpath:org/neo4j/logging/StructuredJsonLayout.json"/>
            <Policies>
                <SizeBasedTriggeringPolicy size="20 MB"/>
            </Policies>
            <DefaultRolloverStrategy fileIndex="min" max="7"/>
 </RollingRandomAccessFile>
```

See also `<JsonTemplateLayout>` and Default logging configuration.

The following information is available in the JSON format:

Table 584. JSON format log entries

Name	Description
`time`	The timestamp of the log message.
`level`	The log level.
`type`	It is always `security`.
`source`	Connection details.
`database`	The database name the command is executed on. This field is optional and thus will not be populated for all security events.
`username`	The user connected to the security event. This field is deprecated by `executingUser`.
`executingUser`	The name of the user triggering the security event. Either same as `authenticatedUser` or an impersonated user.
`authenticatedUser`	The name of the user who authenticated and is connected to the security event.
`message`	The log message.
`stacktrace`	Included if there is a stacktrace associated with the log message.

An example of the security log in a plain format:

```
2019-12-09 13:45:00.796+0000 INFO  [johnsmith]: logged in
2019-12-09 13:47:53.443+0000 ERROR [johndoe]: failed to log in: invalid principal or credentials
2019-12-09 13:48:28.566+0000 INFO  [johnsmith]: CREATE USER janedoe SET PASSWORD '********' CHANGE
REQUIRED
2019-12-09 13:48:32.753+0000 INFO  [johnsmith]: CREATE ROLE custom
2019-12-09 13:49:11.880+0000 INFO  [johnsmith]: GRANT ROLE custom TO janedoe
2019-12-09 13:49:34.979+0000 INFO  [johnsmith]: GRANT TRAVERSE ON GRAPH * NODES A, B (*) TO custom
2019-12-09 13:49:37.053+0000 INFO  [johnsmith]: DROP USER janedoe
2019-12-09 13:52:24.685+0000 INFO  [johnsmith:alice]: impersonating user alice logged in
```

Configure the query log

Query logging is enabled by default and is controlled by the setting `db.logs.query.enabled`. It helps you analyze long-running queries and does not impact system performance. The default is to log all queries,

but it is recommended to log for queries exceeding a certain threshold.

Configuration settings

The following values are available for the parameter `db.logs.query.enabled`:

Table 585. `db.logs.query.enabled` values

Option	Description
`OFF`	Completely disable logging.
`INFO`	Log at the end of queries that have either succeeded or failed. The `db.logs.query.threshold` parameter is used to determine the threshold for logging a query. If the execution of a query takes longer than this threshold, the query is logged. Setting the threshold to `0s` results in all queries being logged.
`VERBOSE`	[.label-baltic]# Default #Log all queries at both start and finish, regardless of `db.logs.query.threshold`.

The following configuration settings are available for the query logging:

Table 586. Query log configurations

The *query log* configuration	Default value	Description
`db.logs.query.early_raw_logging_enabled`	`false`	Log query text and parameters without obfuscating passwords. This allows queries to be logged earlier before parsing starts.
`db.logs.query.enabled`	`VERBOSE`	Log executed queries.
`db.logs.query.max_parameter_length`	`2147483647`	This configuration option allows you to set a maximum parameter length to include in the log. Parameters exceeding this length will be truncated and appended with `...`. This applies to each parameter in the query.

The *query log* configuration	Default value	Description
`db.logs.query.obfuscate_literals`	`false`	If `true`, obfuscates all query literals before writing the query to the log. This is useful when Cypher queries expose sensitive information. It is recommended to set this setting to `true` in production. **Note:** Node labels, relationship types, and map property keys are still shown. Changing the setting does not affect cached queries. Therefore, if you want the switch to have an immediate effect, you must also clear the query cache; `CALL db.clearQueryCaches()`. Also, keep in mind that if Neo4j receives a malformed query that cannot be parsed, it cannot obfuscate its literals (because it does not know which parts are literals) and, therefore, the query text will not be included in any logging. **Warning:** The setting does not obfuscate literals in parameters. If parameter values are not required in the log, set `db.logs.query.parameter_logging_enabled=false`. The setting does not obfuscate any sensitive information when logging errors. To get that capability, set `db.logs.query.obfuscate_errors=true`.

The *query log* configuration	Default value	Description
Introduced in 5.26.21 `db.logs.query.obfuscate_errors`	`false`	If `true`, obfuscates all error information that can contain sensitive data before writing it to the log. This applies to error messages when the query log uses pattern layout or JSON layout with fields such as `failureReason` and `statusDescription`, where `statusDescription` is an element of `errorInfo`. It is recommended to set this setting to `true` in production. **Warning:** The setting does not obfuscate literals when logging Cypher queries. To get that capability, set `db.logs.query.obfuscate_literals=true`.
`db.logs.query.parameter_logging_enabled`	`true`	Log parameters for the executed queries being logged. You can disable this configuration setting if you do not want to display sensitive information.
`db.logs.query.plan_description_enabled`	`false`	This configuration option allows you to log the query plan for each query. The query plan shows up as a description table and is useful for debugging purposes. Every time a Cypher query is run, it generates and uses a plan for the execution of the code. The plan generated can be affected by changes in the database, such as adding a new index. As a result, it is not possible to historically see what plan was used for the original query execution. **Note:** Enabling this option has a performance impact on the database due to the cost of preparing and including the plan in the *query log*. It is not recommended for everyday use.

The *query log* configuration	Default value	Description
`db.logs.query.threshold`	`0s`	If the query execution takes longer than this threshold, the query is logged once completed (provided query logging is set to `INFO`). A threshold of `0` seconds logs all queries.
`db.logs.query.transaction.enabled`	`OFF`	Track the start and end of a transaction within the query log. Log entries are written to the *query log*. They include the transaction ID for a specific query and the start and end of a transaction. You can also choose a level of logging (`OFF`, `INFO`, or `VERBOSE`). If `INFO` is selected, you must exceed the time before the log is written (`db.logs.query.transaction.threshold`).
`db.logs.query.transaction.threshold`	`0s`	If the transaction is open for longer than this threshold (duration of time), the transaction is logged once completed, provided transaction logging is set to `INFO`. Defaults to `0` seconds, which means all transactions are logged. This can be useful when identifying where there is a significant time lapse after query execution and transaction commits, especially in performance analysis around locking.

Configure for simple query logging

In this example, the query logging is set to `INFO`, and all other query log parameters are at their defaults.

```
db.logs.query.enabled=INFO
```

The following is an example of the query log with this basic configuration:

```
2017-11-22 14:31 ... INFO  9 ms: bolt-session   bolt    johndoe neo4j-javascript/1.4.1
client/127.0.0.1:59167  ...
2017-11-22 14:31 ... INFO  0 ms: bolt-session   bolt    johndoe neo4j-javascript/1.4.1
client/127.0.0.1:59167  ...
2017-11-22 14:32 ... INFO  3 ms: server-session http    127.0.0.1   /db/data/cypher neo4j - CALL
dbms.procedures() - {}
2017-11-22 14:32 ... INFO  1 ms: server-session http    127.0.0.1   /db/data/cypher neo4j - CALL
dbms.showCurrentUs...
2017-11-22 14:32 ... INFO  0 ms: bolt-session   bolt    johndoe neo4j-javascript/1.4.1
client/127.0.0.1:59167  ...
2017-11-22 14:32 ... INFO  0 ms: bolt-session   bolt    johndoe neo4j-javascript/1.4.1
client/127.0.0.1:59167  ...
2017-11-22 14:32 ... INFO  2 ms: bolt-session   bolt    johndoe neo4j-javascript/1.4.1
client/127.0.0.1:59261  ...
```

Configure for query logging with more details

In this example, the query log is enabled, as well as some additional logging:

```
db.logs.query.enabled=INFO
db.logs.query.parameter_logging_enabled=true
db.logs.query.threshold=<appropriate value>
```

The following sample query is run on the Movies database:

```
MATCH (n:Person {name:'Tom Hanks'})-[:ACTED_IN]->(n1:Movie)<-[:DIRECTED]-(n2:Person {name:"Tom Hanks"})
RETURN n1.title
```

The corresponding query log in <*.file*>*query.log* is:

```
2017-11-23 12:44:56.973+0000 INFO  1550 ms: (planning: 20, cpu: 920, waiting: 10) - 13792 B - 15 page
hits, 0 page faults - bolt-session    bolt    neo4j   neo4j-javascript/1.4.1      client/127.0.0.1:58189
server/127.0.0.1:7687>  neo4j - match (n:Person {name:'Tom Hanks'})-[:ACTED_IN]->(n1:Movie)<-[:DIRECTED]-
(n2:Person {name:"Tom Hanks"}) return n1.title; - {} - {}
```

An obvious but essential point of note when examining the parameters of a particular query is to ensure you analyze only the entries relevant to that specific query plan, as opposed to, e.g., CPU, time, bytes, and so on for each log entry in sequence.

Following is a breakdown of resource usage parameters with descriptions corresponding to the above query:

`2017-11-23 12:44:56.973+0000`

Log timestamp.

`INFO`

Log category.

`1550 ms`

Total elapsed cumulative wall time spent in query execution. It is the total of planning time + CPU + waiting + any other processing time, e.g., taken to acquire execution threads. This figure is cumulative for every time a CPU thread works on executing the query.

`Planning`

Refers to the time the Cypher engine takes to create a query plan. Plans may be cached for repetitive queries, and therefore, planning times for such queries will be shorter than those for previously unplanned ones. In the example, this contributed 20ms to the total execution time of 1550ms.

`CPU time`

Refers to the time taken by the individual threads executing the query, e.g., a query is submitted at 08:00. It uses CPU for 720ms, but then the CPU swaps out to another query, so the first query is no longer using the CPU. Then, after 100ms, it gets/uses the CPU again for 200ms (more results to be loaded, requested by the client via the Driver), then the query completes at 08:01:30, so the total duration is 1550ms (includes some round-trip time for 2 round-trips), and CPU is 720+200=920ms.

`Waiting`

Time a query spent waiting before execution (in ms), for example, if an existing query has a lock which the new query must wait to release. In the example, this contributed 10ms to the total execution time of

1550ms.
It is important to note that the client requests data from the server only when its record buffer is empty (one round-trip from the server may end up with several records), and the server stops pushing data into outgoing buffers if the client does not read them in a timely fashion. Therefore, it depends on the size of the result set. If it is relatively small and fits in a single round-trip, the client receives all the results at once, and the server finishes processing without any client-side effect. Meanwhile, if the result set is large, the client-side processing time will affect the overall time, as it is directly connected to when new data is requested from the server.

`13792 B`

The logged allocated bytes for the executed queries. This is the amount of HEAP memory used during the life of the query. The logged number is cumulative over the duration of the query, i.e., for memory-intense or long-running queries, the value may be larger than the current memory allocation.

`15 page hits`

Page hit means the result was returned from the page cache as opposed to the disk. In this case, the page cache was hit 15 times.

`0 page faults`

Page fault means that the query result data was not in the `dbms.memory.pagecache` and, therefore, had to be fetched from the file system. In this case, query results were returned entirely from the 8 page cache hits mentioned above, so there were 0 hits on the disk required.

`bolt-session`

The session type.

`bolt`

The Browser ↔ database communication protocol used by the query.

`neo4j`

The process ID.

`neo4j-javascript/1.4.1`

The Driver version.

`client/127.0.0.1:52935`

The query client outbound `IP:port` used.

`server/127.0.0.1:7687>`

The server listening `IP:port` used.

`neo4j`

username of the query executioner

`match (n:Person {name:'Tom Hanks'})-[:ACTED_IN]→(n1:Movie)←[:DIRECTED]-(n2:Person {name:"Tom Hanks"}) return n1.title`

The executed query.

The last two parenthesis `{}` `{}` are for the query parameters and `txMetaData`.

Attach metadata to a transaction

You can attach metadata to a transaction and have it printed in the query log using the built-in procedure `tx.setMetaData`.

> **Note:**
>
> Neo4j Drivers also support attaching metadata to a transaction. For more information, see the respective Driver's manual.

Every graph app should follow a convention for passing metadata with the queries that it sends to Neo4j:

```
{
  app: "neo4j-browser_v4.4.0",
  type: "system"
}
```

`app` can be a user-agent styled-name plus version.

`type` can be one of:

- `system` — a query automatically run by the app.
- `user-direct` — a query the user directly submitted to/through the app.
- `user-action` — a query resulting from an action the user performed.
- `user-transpiled` — a query that has been derived from the user input.

This is typically done programmatically but can also be used with the Neo4j dev tools.
In general, you start a transaction on a user database and attach a list of metadata to it by calling `tx.setMetaData`. You can also use the procedure `CALL tx.getMetaData()` to show the metadata of the current transaction. These examples use the Movie Graph dataset from the pre-installed Neo4j Browser guide.

Example 145. Using `cypher-shell`, attach metadata to a transaction

> Cypher Shell always adds metadata that follows the convention by default. In this example, the defaults are overridden.

```
neo4j@neo4j> :begin
neo4j@neo4j# CALL tx.setMetaData({app: 'neo4j-cypher-shell_v.4.4.0', type: 'user-direct', user:
'jsmith'});
0 rows
ready to start consuming query after 2 ms, results consumed after another 0 ms
neo4j@neo4j# CALL tx.getMetaData();
+--------------------------------------------------------------------------+
| metadata                                                                 |
+--------------------------------------------------------------------------+
| {app: "neo4j-cypher-shell_v.4.4.0", type: "user-direct", user: "jsmith"} |
+--------------------------------------------------------------------------+

1 row
ready to start consuming query after 37 ms, results consumed after another 2 ms
neo4j@neo4j# MATCH (n:Person) RETURN n  LIMIT 5;
```

```
+--------------------------------------------------+
| n                                                |
+--------------------------------------------------+
| (:Person {name: "Keanu Reeves", born: 1964})      |
| (:Person {name: "Carrie-Anne Moss", born: 1967})  |
| (:Person {name: "Laurence Fishburne", born: 1961}) |
| (:Person {name: "Hugo Weaving", born: 1960})      |
| (:Person {name: "Lilly Wachowski", born: 1967})   |
+--------------------------------------------------+

5 rows
ready to start consuming query after 2 ms, results consumed after another 1 ms
neo4j@neo4j# :commit
```

Example result in the query.log file

```
2021-07-30 14:43:17.176+0000 INFO  id:225 - 2 ms: 136 B - bolt-session  bolt    neo4j-cypher-
shell/v4.4.0       client/127.0.0.1:54026  server/127.0.0.1:7687>  neo4j - neo4j -
MATCH (n:Person) RETURN n  LIMIT 5; - {} - runtime=pipelined - {app: 'neo4j-cypher-shell_v.4.4.0',
type: 'user-direct', user: 'jsmith'}
```

Example 146. Using Neo4j Browser, attach metadata to a transaction

```
CALL tx.setMetaData({app: 'neo4j-browser_v.4.4.0', type: 'user-direct', user: 'jsmith'})
MATCH (n:Person) RETURN n LIMIT 5
```

Example result in the query.log file

```
2021-07-30 14:51:39.457+0000 INFO  Query started: id:328 - 0 ms: 0 B - bolt-session bolt    neo4j-
browser/v4.4.0        client/127.0.0.1:53666  server/127.0.0.1:7687>  neo4j - neo4j - MATCH
(n:Person) RETURN n  LIMIT 5 - {} - runtime=null - {type: 'system', app: 'neo4j-browser_v4.4.0'}
```

Example 147. Using Neo4j Bloom, attach metadata to a transaction

```
CALL tx.setMetaData({app: 'neo4j-browser_v.1.7.0', type: 'user-direct', user: 'jsmith'})
MATCH (n:Person) RETURN n LIMIT 5
```

Example result in the query.log file

```
2021-07-30 15:09:54.048+0000 INFO  id:95 - 1 ms: 72 B - bolt-session    bolt    neo4j-bloom/v1.7.0
client/127.0.0.1:54693  server/127.0.0.1:11003> neo4j - neo4j - RETURN TRUE - {} - runtime=pipelined
- {app: 'neo4j-bloom_v1.7.0', type: 'system'}
```

Note:

In Neo4j Browser and Bloom, the user-provided metadata is always replaced by the system metadata.

Use JSON format for the query log

The query log can use a JSON layout. To change the format, the layout for the `QueryLogger` must be changed from using the `PatternLayout`:

```
<RollingRandomAccessFile name="QueryLog" fileName="${config:server.directories.logs}/query.log"
                     filePattern="${config:server.directories.logs}/query.log.%02i">
    <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSSZ}{GMT+0} %-5p %m%n"/>
    <Policies>
        <SizeBasedTriggeringPolicy size="20 MB"/>
    </Policies>
    <DefaultRolloverStrategy fileIndex="min" max="7"/>
</RollingRandomAccessFile>
```

to using the `JsonTemplateLayout`:

```
<RollingRandomAccessFile name="QueryLog" fileName="${config:server.directories.logs}/query.log"
                     filePattern="${config:server.directories.logs}/query.log.%02i">
    <JsonTemplateLayout eventTemplateUri="classpath:org/neo4j/logging/QueryLogJsonLayout.json"/>
    <Policies>
        <SizeBasedTriggeringPolicy size="20 MB"/>
    </Policies>
    <DefaultRolloverStrategy fileIndex="min" max="7"/>
</RollingRandomAccessFile>
```

See also `<JsonTemplateLayout>` and Default logging configuration.

JSON format log entries

The `QueryLogJsonLayout.json` template mimics the 4.x layout and contains the following information:

Table 587. JSON format log entries

Name	Description
`time`	The timestamp of the log message.
`level`	The log level.
`type`	Valid options are `query` and `transaction`.
`stacktrace`	Included when there is a stacktrace associated with the log message.

If the type of the log entry is `query`, these additional fields are available:

Table 588. JSON format log entries for log type `query`

Name	Description
`event`	Valid options are `start`, `fail`, and `success`.
`id`	The query ID. The ID is incremental, starting from 1 and resetting upon every Neo4j restart. Included when `db.logs.query.enabled` is `VERBOSE`.
`elapsedTimeMs`	The elapsed time in milliseconds.
`planning`	Milliseconds spent on planning.
`cpu`	Milliseconds spent actively executing on the CPU.
`waiting`	Milliseconds spent waiting on locks or other queries, as opposed to actively running this query.
`allocatedBytes`	Number of bytes allocated by the query.

Name	Description
`pageHits`	Number of page hits.
`pageFaults`	Number of page faults.
`source`	Connection details.
`database`	The database name on which the query is run. This field will be `<none>` if the query cannot be parsed and routed to a database.
`executingUser`	The name of the user executing the query. Either same as `authenticatedUser` or an impersonated user.
`authenticatedUser`	The name of the user who authenticated and is executing the query.
`query`	The query text.
`queryParameters`	The query parameters. Included when `db.logs.query.parameter_logging_enabled` is `true`.
`runtime`	The runtime used to run the query.
`annotationData`	Metadata attached to the transaction.
`failureReason`	Reason for failure. Included when applicable.
`errorInfo`	Introduced in 5.25 GQL error information as a JSON object. See GQL error information for details on the contents of the `errorInfo` JSON object.
`transactionId`	The transaction ID of the running query.
`queryPlan`	The query plan. Included when `db.logs.query.plan_description_enabled` is `true`.
`executionPlanCacheKeyHash`	The 32-bit hash of the cache key used to cache the execution plan. It is a fixed-length 8-character string of a hex-encoded integer value. Enabled by default only in the JSON format. If multiple query executions use the same cached execution plan, their hashes should match. For `event=start`, the value is `00000000`.

If the type of the log entry is `transaction`, the following additional fields are available:

Table 589. JSON format log entries for log type `transaction`

Name	Description
`event`	Valid options are `start`, `rollback`, and `commit`.
`database`	The database name on which the transaction is run.
`executingUser`	The name of the user connected to the transaction. Either same as `authenticatedUser` or an impersonated user.
`authenticatedUser`	The name of the user who authenticated and is connected to the transaction.
`transactionId`	ID of the transaction.

GQL error information

Introduced in 5.25

The query log includes the GQL error information under the JSON object `errorInfo`. `errorInfo` can contain the following elements:

- `GQLSTATUS` — A 5-character long alpha-numeric code identifying the error.
- `statusDescription` — A message describing the error.
- `classification` — The type of error representing a division of client, transient, and database errors.
- `position` — The position (a JSON object containing a field for `column`, `offset`, and `line`) in the query where this error occurred.
- `cause` — A JSON object containing the `errorInfo` JSON object of the cause of the current `errorInfo` JSON object.

> **Note:**
>
> The default GQLSTATUS code 50N42 is returned when an exception does not have a GQL-status object. Starting from Neo4j 5.25, we started adding GQL objects to exceptions. Therefore, you can expect many 50N42 codes during this transition period. However, it is important not to rely on this default code, as future Neo4j versions might change it by adding an appropriate GQL object to the exception. Additionally, GQL codes for external procedures are not yet stable.

The following are examples of the `errorInfo` JSON object:

`errorInfo` *JSON object of a database error*

```
...
"errorInfo": {
    "GQLSTATUS": "51N66",
    "statusDescription": "error: system configuration or operation exception - resource exhaustion. Insufficient resources to complete the request.",
    "cause": {
      "GQLSTATUS": "51N55",
      "statusDescription": "error: system configuration or operation exception - cannot create additional database. Failed to create the database `db10`. The limit of databases is reached. Either increase the limit using the config setting dbms.max_databases or drop a database.",
      "classification": "DATABASE_ERROR"
    },
    "classification": "DATABASE_ERROR"
  },
...
```

`errorInfo` *JSON object of a client error*

```
...
"errorInfo": {
    "GQLSTATUS": "42N62",
    "statusDescription": "error: syntax error or access rule violation - variable not defined. Variable `m` not defined.",
    "position": {
      "column": 18,
      "offset": 17,
      "line": 1
    },
    "classification": "CLIENT_ERROR"
  },
"query": "MATCH (n) RETURN m",
```

```
13 ...
```

Metrics

You can configure Neo4j to log many different metrics to help you keep your applications running smoothly. By default, you can view this data via Neo4j Ops Manager (NOM), retrieved from CSV files, or exposed over JMX MBeans, but you can also export it to third-party monitoring tools, such as Graphite and Prometheus.

Tip:

Reading the Performance section is recommended to better understand the metrics.

This section describes the following:

- Essential metrics — Some of the important metrics most Neo4j administrators need to monitor.
- Enable metrics logging — How to configure Neo4j to log metrics data.
- Expose metrics — How to view data using NOM, JMX, CSV files, or other third-party monitoring tools.
- Metrics reference — Neo4j available metrics.

Essential metrics

To ensure your applications are running smoothly, you should monitor:

- The **server load** — the strain on the machine hosting Neo4j.
- The **Neo4j load** — the strain on Neo4j.
- The **cluster health** — to ensure the cluster is working as expected.
- The **workload** of a Neo4j instance.

Tip:

Reading the Performance section is recommended to better understand the metrics.

Server load metrics

Monitoring the hardware resources shows the strain on the server running Neo4j.

You can use utilities, such as the collectd daemon or `systemd` on Linux, to gather information about the system. These metrics can help with capacity planning as your workload grows.

Metric name	Description
CPU usage	If this is reaching 100%, you may need additional CPU capacity.

Metric name	Description
Used memory	This metric tells you if you are close to using all available memory on the server. Make sure your peaks are at 95% or below to reduce the risk of running out of memory. For more information, see Memory configuration.
Free disk space	Observe the rate of your data growth so you can plan for additional storage before you run out. This applies to all disks that Neo4j is writing to. You might also choose to write the log files to a different disk. An out of disk event may disrupt system availability and cause the database going offline, thus creating the risk of database or log file corruption. To avoid that, system monitoring tools should be configured to monitor available disk space on all drives used for databases, indexes, and transactions logs. For more recommendations, see Disks, RAM and other tips, as well as Monitoring servers and Monitoring databases in a Neo4j cluster.

Neo4j load metrics

The Neo4j load metrics monitor the strain that Neo4j is being put under. They can help with capacity planning.

Metric name	Metric	Description
Heap usage	`<prefix>.dbms.vm.heap.used`	If Neo4j consistently uses 100% of the heap, increase the initial and max heap size. For more information, see Memory configuration.
Page cache	`<prefix>.dbms.page_cache.hit_ratio` and `<prefix>.dbms.page_cache.usage_ratio`	When a request misses the page cache, the data must be fetched from a much slower disk. Ideally, the hit_ratio should be above 98% most of the time. This shows how much of the allocated memory to the page cache is used. If this is at 100%, consider increasing the page cache size.
JVM garbage collection	`<prefix>.dbms.vm.gc.time.%s`	The proportion of time the JVM spends reclaiming the heap instead of doing other work. This metric can spike when the database is running low on memory. If this happens, it can halt processing and cause query execution errors. Consider increasing the size of your database if this appears to be the case.

Metric name	Metric	Description
Checkpoint time	`<prefix>.database.<db>.check_point.duration`	You should monitor the checkpoint duration to ensure it does not start to approach the interval between checkpoints. If this happens, consider the following steps to improve checkpointing performance: • Raise the `db.checkpoint.iops.limit` to make checkpoints faster, but only if there is enough IOPS budget available to avoid slowing the commit process. • `server.memory.pagecache.flush.buffer.enabled` / `server.memory.pagecache.flush.buffer.size_in_pages` make checkpoints faster by writing batches of data in a way that plays well with the underlying disk (with a nice multiple to the block size). • Change the checkpointing policy (`db.checkpoint.*`, `db.checkpoint.interval.time`) to more frequent/smaller checkpoints, continuous checkpointing, or checkpoint by volume or `tx` count. For more information, see Checkpoint IOPS limit and Checkpointing and log pruning.

Neo4j cluster health metrics

The cluster health metrics indicate the health of a cluster member at a glance. It is essential to know which instance is the leader. The leader's load pattern differs from the followers, which should exhibit similar load patterns.

Metric name	Metric	Description
Leader	`<prefix>.database.<db>.cluster.raft.is_leader`	Track this for each database primary. It reports `0` if it is not the leader and `1` if it is the leader. The sum of all of these should always be `1`. However, there are transient periods in which the sum can be more than `1` because more than one member thinks it is the leader.
Transaction workload	`<prefix>.database.<db>.transaction.last_committed_tx_id`	The ID of the last committed transaction. Track this for each Neo4j instance. It might break into separate charts. It should show one line, ever-increasing, and if one of the lines levels off or falls behind, it is clear that this instance is no longer replicating data, and action is needed to rectify the situation.

See more about how to Monitor cluster endpoints for status information.

Workload metrics

These metrics help monitor the workload of a Neo4j instance. The absolute values of these depend on the sort of workload you expect.

Metric name	Metric	Description
Bolt connections	`<prefix>.dbms.bolt.connections_running`	The number of connections that are currently executing Cypher and returning results.
Total nodes/relationships	`<prefix>.database.<db>.count.node` and `<prefix>.database.<db>.count.relationship`	(Not enabled by default) Total number of distinct relationship types. Total number of distinct property names. Total number of relationships. Total number of nodes.
Throughput	`<prefix>.database.<db>.db.query.execution.latency.millis`	This metric produces a histogram of 99th and 95th percentile transaction latencies. Useful for identifying spikes or increases in the data load.

Note:

For the complete list of all available metrics in Neo4j, see Metrics reference.

Enable metrics logging

Note:

A subset of all available metrics is enabled by default. See `server.metrics.filter`. The list was last updated in Neo4j 4.2.

You can enable/disable metrics using the configuration setting `server.metrics.enabled`. You can also use the setting `server.metrics.filter` to enable only the metrics you want. The metrics must be specified as a comma-separated list of globbing patterns. The following example enables all checkpoint metrics and the pagecache eviction metric:

```
# Setting for enabling all supported metrics. (Default is true) Setting this to false disables all
metrics.
server.metrics.enabled=true

# Setting for enabling only the metrics matching the filter.
server.metrics.filter=*check_point*,neo4j.page_cache.evictions
```

For more information on the available metrics configuration settings, see Configuration settings.

Expose metrics

Neo4j supports the following ways of exposing data for monitoring purposes:

- Neo4j Ops Manager — a UI-based tool that enables a DBA (or any administrator) to monitor, administer, and operate all of the Neo4j DBMSs in an Enterprise.
- CSV files — retrieve metrics from CSV files. Enabled by default.
- JMX MBeans — expose metrics over JMX MBeans. Enabled by default.

- Graphite — send metrics to Graphite or any monitoring tool based on the Graphite protocol. Disabled by default.
- Prometheus — publish metrics for polling as Prometheus endpoint. Disabled by default.

Neo4j Ops Manager

The Neo4j metrics data can be viewed via Neo4j Ops Manager (NOM). For more information on how to install and set up NOM to be used with your Neo4j DBMS, see Neo4j Ops Manager documentation.

CSV files

Export metrics to CSV files.

Add the following settings to *neo4j.conf* in order to enable export of metrics into local .CSV files:

```
# Enable the CSV exporter. Default is true.
server.metrics.csv.enabled=true
# Directory path for output files.
# Default is the /metrics directory under NEO4J_HOME.
server.directories.metrics=/local/file/system/path
# How often to store data. Default is 30 seconds.
server.metrics.csv.interval=30s
# The maximum number of CSV files that will be saved. Default is 7.
server.metrics.csv.rotation.keep_number=7
# The file size at which the CSV files will auto-rotate. Default is 10.00MiB.
server.metrics.csv.rotation.size=10.00MiB
# Compresses the metric archive files. Default is NONE. Possible values are NONE, ZIP, and GZ.
server.metrics.csv.rotation.compression=ZIP
```

`server.metrics.csv.rotation.compression` selects the compression scheme to use on the files after rotation. Since CSV files are highly compressible, it is recommended to enable compression of the files to save disk space.

JMX MBeans

From Neo4j 4.2.2 onwards, the JMX metrics are exposed by default over JMX MBeans.

```
# Enable the JMX MBeans integration. Default is true.
server.metrics.jmx.enabled=true
```

For more information about accessing and adjusting the metrics, see The Java Reference Guide → JMX metrics.

Graphite

Send metrics to Graphite or any monitoring tool based on the Graphite protocol.

Add the following settings to *neo4j.conf* to enable integration with Graphite:

```
# Enable the Graphite integration. Default is false.
server.metrics.graphite.enabled=true
# The hostname or IP address of the Graphite server.
# A socket address in the format <hostname>, <hostname>:<port>, or :<port>.
# If missing, the port or hostname is acquired from server.default_listen_address.
```

```
# The default port number for Graphite is 2003.
server.metrics.graphite.server=localhost:2003
# How often to send data. Default is 30 seconds.
server.metrics.graphite.interval=30s
# Prefix for Neo4j metrics on Graphite server.
server.metrics.prefix=neo4j
```

Start Neo4j and connect to Graphite via a web browser to monitor your Neo4j metrics.

If you configure the Graphite server to be a hostname or DNS entry, you should be aware that the JVM resolves hostnames to IP addresses and, by default, caches the result indefinitely for security reasons. This is controlled by the value of `networkaddress.cache.ttl` in the JVM Security properties. See https://docs.oracle.com/javase/8/docs/technotes/guides/net/properties.html for more information.

Prometheus

Publish metrics for polling as Prometheus endpoint.

Add the following settings to *neo4j.conf* to enable the Prometheus endpoint.

```
# Enable the Prometheus endpoint. Default is false.
server.metrics.prometheus.enabled=true
# The hostname and port to use as Prometheus endpoint.
# A socket address is in the format <hostname>, <hostname>:<port>, or :<port>.
# If missing, the port or hostname is acquired from server.default_listen_address.
# The default is localhost:2004.
server.metrics.prometheus.endpoint=localhost:2004
```

When Neo4j is fully started, a Prometheus endpoint will be available at the configured address.

Warning:

You should never expose the Prometheus endpoint directly to the Internet. If security is of paramount importance, you should set `server.metrics.prometheus.endpoint=localhost:2004` and configure a reverse HTTP proxy on the same machine that handles the authentication, SSL, caching, etc.

If you can afford to send unencrypted metrics within the internal network, such as `server.metrics.prometheus.endpoint=10.0.0.123:2004`, all servers within the same netmask will be able to access it.

If you specify anything more permissive, such as `server.metrics.prometheus.endpoint=0.0.0.0:2004`, you should have a firewall rule to prevent any unauthorized access. Data in transit will still not be encrypted, so it should never go over any insecure networks.

Tip:

When Neo4j metrics are exposed via Prometheus, their names are transformed to comply with Prometheus naming conventions.

The following general rules are applied:

- Dots (.) are replaced with underscores (_), since Prometheus does not support dots in metric names.
- Depending on the metric type, a postfix is added.

Original Neo4j metric names can be found in Prometheus output, see lines starting with `# HELP`.

For more information, see Prometheus Documentation.

Metrics reference

Caution:

You should use caution when interpreting unfamiliar metrics. Reading the Performance section is recommended to better understand the metrics.

Types of metrics

Neo4j has the following types of metrics:

- Global — covers the whole Neo4j DBMS.
- Per database — covers an individual database.

The metrics fall into one of the following categories:

- Gauge — shows an instantaneous reading of a particular value.
- Counter — shows an accumulated value.
- Histogram — shows the distribution of values.

Neo4j supports several ways of exposing metrics. For more details, refer to the page Expose metrics.

Global metrics

Global metrics cover the whole database management system and represent the system's status as a whole.

Global metrics have the following name format:

- `<user-configured-prefix>.dbms.<metric-name>`, where the <user-configured-prefix> can be configured with the `server.metrics.prefix` configuration setting.

Metrics of this type are reported as soon as the database management system is available. For example, all JVM-related metrics are global. In particular, the `neo4j.dbms.vm.thread.count` metric has a default user-configured-prefix `neo4j` and the global metric name is `vm.thread.count`.

By default, global metrics include:

- Bolt metrics
- Page cache metrics
- GC metrics
- Thread metrics
- Database operation metrics
- Web Server metrics
- JVM metrics

Database metrics

Each database metric is reported for a particular database only. Database metrics are only available during the lifetime of the database. When a database becomes unavailable, all of its metrics become unavailable also.

Database metrics have the following name format:

- `<user-configured-prefix>.database.<database-name>.<metric-name>`, where the <user-configured-prefix> can be configured with the `server.metrics.prefix` configuration setting.

For example, any transaction metric is a database metric. In particular, the `neo4j.database.mydb.transaction.started` metric has a default user-configured-prefix `neo4j` and is a metric for the `mydb` database.

By default, database metrics include:

- Transaction metrics
- Checkpoint metrics
- Log rotation metrics
- Database data metrics
- Cypher metrics
- Clustering metrics

General-purpose metrics

Bolt metrics

Table 590. Bolt metrics

Name	Description
`<prefix>.dbms.bolt.connections_opened`	The total number of Bolt connections opened since startup. This includes both succeeded and failed connections. Useful for monitoring load via the Bolt drivers in combination with other metrics. (counter)

Name	Description
`<prefix>.dbms.bolt.connections_closed`	The total number of Bolt connections closed since startup. This includes both properly and abnormally ended connections. Useful for monitoring load via Bolt drivers in combination with other metrics. (counter)
`<prefix>.dbms.bolt.connections_running`	The total number of Bolt connections that are currently executing Cypher and returning results. Useful to track the overall load on Bolt connections. This is limited to the number of Bolt worker threads that have been configured via `dbms.connector.bolt.thread_pool_max_size`. Reaching this maximum indicated the server is running at capacity. (gauge)
`<prefix>.dbms.bolt.connections_idle`	The total number of Bolt connections that are not currently executing Cypher or returning results. (gauge)
`<prefix>.dbms.bolt.messages_received`	The total number of messages received via Bolt since startup. Useful to track general message activity in combination with other metrics. (counter)
`<prefix>.dbms.bolt.messages_started`	The total number of messages that have started processing since being received. A received message may have begun processing until a Bolt worker thread becomes available. A large gap observed between `bolt.messages_received` and `bolt.messages_started` could indicate the server is running at capacity. (counter)
`<prefix>.dbms.bolt.messages_done`	The total number of Bolt messages that have completed processing whether successfully or unsuccessfully. Useful for tracking overall load. (counter)
`<prefix>.dbms.bolt.messages_failed`	The total number of messages that have failed while processing. A high number of failures may indicate an issue with the server and further investigation of the logs is recommended. (counter)
`<prefix>.dbms.bolt.accumulated_queue_time`	(unsupported feature) When `internal.server.bolt.thread_pool_queue_size` is enabled, the total time in milliseconds that a Bolt message waits in the processing queue before a Bolt worker thread becomes available to process it. Sharp increases in this value indicate that the server is running at capacity. If `internal.server.bolt.thread_pool_queue_size` is disabled, the value should be `0`, meaning that messages are directly handed off to worker threads. (counter)
`<prefix>.dbms.bolt.accumulated_processing_time`	The total amount of time in milliseconds that worker threads have been processing messages. Useful for monitoring load via Bolt drivers in combination with other metrics. (counter)

Name	Description
<prefix>.dbms.bolt.worker_thread_bound_time	[.label-success]# Introduced in 5.21 #The amount of time in milliseconds that worker threads spent bound to a given connection. (histogram)
<prefix>.dbms.bolt.response_success	(unsupported feature) When `internal.server.bolt.response_metrics` is enabled, number of `encounteredsuccess` responses. (counter)
<prefix>.dbms.bolt.response_ignored	(unsupported feature) When `internal.server.bolt.response_metrics` is enabled, number of `encounteredignored` responses (counter)
<prefix>.dbms.bolt.response_failed	(unsupported feature) When `internal.server.bolt.response_metrics` is enabled, number of `encounteredinstances` of a given error code. (counter)

Bolt Driver metrics

Introduced in 5.13

Table 591. Bolt Driver metrics

Name	Description
<prefix>.dbms.bolt_driver.api.managed_transaction_function_calls	The total number of managed transaction function calls. (counter)
<prefix>.dbms.bolt_driver.api.unmanaged_transaction_calls	The total number of unmanaged transaction function calls. (counter)
<prefix>.dbms.bolt_driver.api.implicit_transaction_calls	The total number of implicit transaction function calls. (counter)
<prefix>.dbms.bolt_driver.api.execute_calls	The total number of driver-level execute function calls. (counter)

Database checkpointing metrics

Table 592. Database checkpointing metrics

Name	Description
<prefix>.check_point.events	The total number of checkpoint events executed so far. (counter)
<prefix>.check_point.total_time	The total time, in milliseconds, spent in checkpointing so far. (counter)

Name	Description
<prefix>.check_point.duration	The duration, in milliseconds, of the last checkpoint event. Checkpoints should generally take several seconds to several minutes. Long checkpoints can be an issue, as these are invoked when the database stops, when a hot backup is taken, and periodically as well. Values over `30` minutes or so should be cause for some investigation. (gauge)
<prefix>.check_point.flushed_bytes	[.label-success]# Introduced in 5.10 #The accumulated number of bytes flushed during all checkpoint events combined. (counter)
<prefix>.check_point.limit_millis	Number of millisecond checkpoint was paused by io limiter. (gauge)
<prefix>.check_point.limit_times	Number of times checkpoint was paused by io limiter. (gauge)
<prefix>.check_point.pages_flushed	The number of pages that were flushed during the last checkpoint event. (gauge)
<prefix>.check_point.io_performed	The number of IOs from Neo4j perspective performed during the last check point event. (gauge)
<prefix>.check_point.io_limit	The IO limit used during the last checkpoint event. (gauge)

Cypher metrics

Table 593. Cypher metrics

Name	Description
<prefix>.cypher.replan_events	The total number of times Cypher has decided to re-plan a query. Neo4j caches 1000 plans by default. Seeing sustained replanning events or large spikes could indicate an issue that needs to be investigated. (counter)
<prefix>.cypher.replan_wait_time	The total number of seconds waited between query replans. (counter)

Database data count metrics

Table 594. Database data count metrics

Name	Description
<prefix>.neo4j.count.relationship	The total number of relationships in the database. (gauge)
<prefix>.neo4j.count.node	The total number of nodes in the database. A rough metric of how big your graph is. And if you are running a bulk insert operation you can see this tick up. (gauge)

Name	Description
`<prefix>.neo4j.count.relationship_types`	Introduced in 5.15 The total number of internally generated IDs for the different relationship types stored in the database. These IDs do not reflect changes in the actual data. Informational, not an indication of any issue. (gauge)

Database neo4j pools metrics

Table 595. Database neo4j pools metrics

Name	Description
`<prefix>.database.<database>.pool.<pool>.<database>.used_heap`	Used or reserved heap memory in bytes. (gauge)
`<prefix>.database.<database>.pool.<pool>.<database>.used_native`	Used or reserved native memory in bytes. (gauge)
`<prefix>.database.<database>.pool.<pool>.<database>.total_used`	Sum total used heap and native memory in bytes. (gauge)
`<prefix>.database.<database>.pool.<pool>.<database>.total_size`	Sum total size of capacity of the heap and/or native memory pool. (gauge)
`<prefix>.database.<database>.pool.<pool>.<database>.free`	Available unused memory in the pool, in bytes. (gauge)

Database operation count metrics

Table 596. Database operation count metrics

Name	Description
`<prefix>.db.operation.count.create`	Count of successful database create operations. (counter)
`<prefix>.db.operation.count.start`	Count of successful database start operations. (counter)
`<prefix>.db.operation.count.stop`	Count of successful database stop operations. (counter)
`<prefix>.db.operation.count.drop`	Count of successful database drop operations. (counter)
`<prefix>.db.operation.count.failed`	Count of failed database operations. (counter)
`<prefix>.db.operation.count.recovered`	Count of database operations that failed previously but have recovered. (counter)

Database state count metrics

Introduced in 5.7

Table 597. Database state count metrics

Name	Description
`<prefix>.db.state.count.hosted`	Databases hosted on this server. Databases in states `started`, `store copying`, or `draining` are considered hosted. (gauge)
`<prefix>.db.state.count.failed`	Databases in a failed state on this server. (gauge)
`<prefix>.db.state.count.desired_started`	Databases that desire to be started on this server. (gauge)

Database data metrics

Deprecated in 5.15

Table 598. Database data metrics

Name	Description
`<prefix>.ids_in_use.relationship_type`	The total number of internally generated IDs for the different relationship types stored in the database. These IDs do not reflect changes in the actual data. Informational, not an indication of any issue. (gauge)
`<prefix>.ids_in_use.property`	The total number of internally generated IDs for the different property names stored in the database. These IDs do not reflect changes in the actual data. Informational, not an indication of any issue. (gauge)
`<prefix>.ids_in_use.relationship`	The total number of internally generated reusable IDs for the relationships stored in the database. These IDs do not reflect changes in the actual data. If you want to have a rough metric of how big your graph is, use `<prefix>.neo4j.count.relationship` instead. (gauge)
`<prefix>.ids_in_use.node`	The total number of internally generated reusable IDs for the nodes stored in the database. These IDs do not reflect changes in the actual data. If you want to have a rough metric of how big your graph is, use `<prefix>.neo4j.count.node` instead. (gauge)

Note:

Database data metrics do not report values for databases using block format. Instead, the metrics always report `-1`. This is not an indication of an error of any kind.

Global neo4j pools metrics

Table 599. Global neo4j pools metrics

Name	Description
`<prefix>.dbms.pool.<pool>.used_heap`	Used or reserved heap memory in bytes. (gauge)
`<prefix>.dbms.pool.<pool>.used_native`	Used or reserved native memory in bytes. (gauge)

Name	Description
<prefix>.dbms.pool.<pool>.total_used	Sum total used heap and native memory in bytes. (gauge)
<prefix>.dbms.pool.<pool>.total_size	Sum total size of the capacity of the heap and/or native memory pool. (gauge)
<prefix>.dbms.pool.<pool>.free	Available unused memory in the pool, in bytes. (gauge)

Database page cache metrics

Table 600. Database page cache metrics

Name	Description
<prefix>.page_cache.eviction_exceptions	The total number of exceptions seen during the eviction process in the page cache. (counter)
<prefix>.page_cache.flushes	The total number of page flushes executed by the page cache. (counter)
<prefix>.page_cache.merges	The total number of page merges executed by the page cache. (counter)
<prefix>.page_cache.unpins	The total number of page unpins executed by the page cache. (counter)
<prefix>.page_cache.pins	The total number of page pins executed by the page cache. (counter)
<prefix>.page_cache.evictions	The total number of page evictions executed by the page cache. (counter)
<prefix>.page_cache.evictions.cooperative	The total number of cooperative page evictions executed by the page cache due to low available pages. (counter)
<prefix>.page_cache.eviction.flushes	[.label-success]# Introduced in 5.17 #The total number of pages flushed by page eviction. (counter)
<prefix>.page_cache.eviction.cooperative.flushes	[.label-success]# Introduced in 5.17 #The total number of pages flushed by cooperative page eviction. (counter)
<prefix>.page_cache.page_faults	The total number of page faults in the page cache. If this count keeps increasing over time, it may indicate that more page cache is required. However, note that when Neo4j Enterprise starts up, all page cache warmup activities result in page faults. Therefore, it is normal to observe a significant page fault count immediately after startup. (counter)
<prefix>.page_cache.page_fault_failures	The total number of failed page faults happened in the page cache. (counter)

Name	Description
<prefix>.page_cache.page_cancelled_faults	The total number of cancelled page faults happened in the page cache. (counter)
<prefix>.page_cache.page_vectored_faults	The total number of vectored page faults happened in the page cache. (counter)
<prefix>.page_cache.page_vectored_faults_failures	The total number of failed vectored page faults happened in the page cache. (counter)
<prefix>.page_cache.page_no_pin_page_faults	The total number of page faults that are not caused by the page pins happened in the page cache. Represent pages loaded by the vectored faults (counter)
<prefix>.page_cache.hits	The total number of page hits happened in the page cache. (counter)
<prefix>.page_cache.hit_ratio	The ratio of hits to the total number of lookups in the page cache. Performance relies on efficiently using the page cache, so this metric should be in the 98-100% range consistently. If it is much lower than that, then the database is going to disk too often. (gauge)
<prefix>.page_cache.usage_ratio	The ratio of number of used pages to total number of available pages. This metric shows what percentage of the allocated page cache is actually being used. If it is 100%, then it is likely that the hit ratio will start dropping, and you should consider allocating more RAM to page cache. (gauge)
<prefix>.page_cache.bytes_read	The total number of bytes read by the page cache. (counter)
<prefix>.page_cache.bytes_written	The total number of bytes written by the page cache. (counter)
<prefix>.page_cache.iops	The total number of IO operations performed by page cache. (counter)
<prefix>.page_cache.throttled.times	The total number of times page cache flush IO limiter was throttled during ongoing IO operations. (counter)
<prefix>.page_cache.throttled.millis	The total number of millis page cache flush IO limiter was throttled during ongoing IO operations. (counter)
<prefix>.page_cache.pages_copied	The total number of page copies happened in the page cache. (counter)

Query execution metrics

Table 601. Query execution metrics

Name	Description
<prefix>.db.query.execution.success	Count of successful queries executed. Server-side routed queries contribute to this count on the server where they eventually land and are executed, not on the intermediate, routing server. (counter)

Name	Description
<prefix>.db.query.execution.failure	Count of failed queries executed. Server-side routed queries contribute to this count on the server where they eventually land and are executed, not on the intermediate, routing server. (counter)
<prefix>.db.query.execution.latency.millis	Execution time in milliseconds of queries executed successfully. (histogram)
<prefix>.db.query.execution.parallel.success	Count of successful queries executed by the parallel runtime. Server-side routed queries contribute to this count on the server where they eventually land and are executed, not on the intermediate, routing server. (counter)
<prefix>.db.query.execution.parallel.failure	Count of failed queries executed by the parallel runtime. Server-side routed queries contribute to this count on the server where they eventually land and are executed, not on the intermediate, routing server. (counter)
<prefix>.db.query.execution.parallel.latency.millis	Execution time in milliseconds of queries executed successfully in parallel runtime. (histogram)
<prefix>.db.query.execution.pipelined.success	Count of successful queries executed by the pipelined runtime. Server-side routed queries contribute to this count on the server where they eventually land and are executed, not on the intermediate, routing server. (counter)
<prefix>.db.query.execution.pipelined.failure	Count of failed queries executed by the pipelined runtime. Server-side routed queries contribute to this count on the server where they eventually land and are executed, not on the intermediate, routing server. (counter)
<prefix>.db.query.execution.pipelined.latency.millis	Execution time in milliseconds of queries executed successfully in pipelined runtime. (histogram)
<prefix>.db.query.execution.slotted.success	Count of successful queries executed by the slotted runtime. Server-side routed queries contribute to this count on the server where they eventually land and are executed, not on the intermediate, routing server. (counter)
<prefix>.db.query.execution.slotted.failure	Count of failed queries executed by the slotted runtime. Server-side routed queries contribute to this count on the server where they eventually land and are executed, not on the intermediate, routing server. (counter)
<prefix>.db.query.execution.slotted.latency.millis	Execution time in milliseconds of queries executed successfully in slotted runtime. (histogram)

Query routing metrics

Introduced in 5.10

Table 602. Query routing metrics

Name	Description
`<prefix>.dbms.routing.query.count.local`	The total number of queries executed locally. (counter)
`<prefix>.dbms.routing.query.count.remote_internal`	The total number of queries routed over to another member of the same cluster. (counter)
`<prefix>.dbms.routing.query.count.remote_external`	The total number of queries routed over to a server outside the cluster. (counter)

Database store size metrics

Table 603. Database store size metrics

Name	Description
`<prefix>.store.size.total`	The total size of the database and transaction logs, in bytes. The total size of the database helps determine how much cache page is required. It also helps compare the total disk space used by the data store and how much is available. (gauge)
`<prefix>.store.size.database`	The size of the database, in bytes. The total size of the database helps determine how much cache page is required. It also helps compare the total disk space used by the data store and how much is available. (gauge)
`<prefix>.store.size.available_reserved`	[.label-success]# Introduced in 5.21 #An estimate of reserved but available space in the database, in bytes. At least this much space is potentially reusable when writing new data. (gauge)

Database transaction log metrics

Table 604. Database transaction log metrics

Name	Description
`<prefix>.log.rotation_events`	The total number of transaction log rotations executed so far. (counter)
`<prefix>.log.rotation_total_time`	The total time, in milliseconds, spent in rotating transaction logs so far. (counter)
`<prefix>.log.rotation_duration`	The duration, in milliseconds, of the last log rotation event. (gauge)
`<prefix>.log.appended_bytes`	The total number of bytes appended to the transaction log. (counter)
`<prefix>.log.flushes`	The total number of transaction log flushes. (counter)

Name	Description
`<prefix>.log.append_batch_size`	The size of the last transaction append batch. (gauge)

Database transaction metrics

Table 605. Database transaction metrics

Name	Description
`<prefix>.transaction.started`	The total number of started transactions. (counter)
`<prefix>.transaction.peak_concurrent`	The highest peak of concurrent transactions. This is a useful value to understand. It can help you with the design for the highest load scenarios and whether the Bolt thread settings should be altered. (counter)
`<prefix>.transaction.active`	The number of currently active transactions. Informational, not an indication of any issue. Spikes or large increases could indicate large data loads or just high read load. (gauge)
`<prefix>.transaction.active_read`	The number of currently active read transactions. (gauge)
`<prefix>.transaction.active_write`	The number of currently active write transactions. (gauge)
`<prefix>.transaction.committed`	The total number of committed transactions. Informational, not an indication of any issue. Spikes or large increases indicate large data loads or just high read load. (counter)
`<prefix>.transaction.committed_read`	The total number of committed read transactions. Informational, not an indication of any issue. Spikes or large increases indicate high read load. (counter)
`<prefix>.transaction.committed_write`	The total number of committed write transactions. Informational, not an indication of any issue. Spikes or large increases indicate large data loads, which could correspond with some behavior you are investigating. (counter)
`<prefix>.transaction.rollbacks`	The total number of rolled back transactions. (counter)
`<prefix>.transaction.rollbacks_read`	The total number of rolled back read transactions. (counter)
`<prefix>.transaction.rollbacks_write`	The total number of rolled back write transactions. Seeing a lot of writes rolled back may indicate various issues with locking, transaction timeouts, etc. (counter)
`<prefix>.transaction.terminated`	The total number of terminated transactions. (counter)
`<prefix>.transaction.terminated_read`	The total number of terminated read transactions. (counter)
`<prefix>.transaction.terminated_write`	The total number of terminated write transactions. (counter)

Name	Description
<prefix>.transaction.last_committed_tx_id	The ID of the last committed transaction. Track this for each instance. (Cluster) Track this for each primary, and each secondary. Might break into separate charts. It should show one line, ever increasing, and if one of the lines levels off or falls behind, it is clear that this member is no longer replicating data, and action is needed to rectify the situation. (counter)
<prefix>.transaction.last_closed_tx_id	The ID of the last closed transaction. (counter)
<prefix>.transaction.tx_size_heap	The transactions' size on heap in bytes. (histogram)
<prefix>.transaction.tx_size_native	The transactions' size in native memory in bytes. (histogram)

Database index metrics

Table 606. Database index metrics

Name	Description
<prefix>.index.fulltext.queried	The total number of times fulltext indexes have been queried. (counter)
<prefix>.index.fulltext.populated	The total number of fulltext index population jobs that have been completed. (counter)
<prefix>.index.lookup.queried	The total number of times lookup indexes have been queried. (counter)
<prefix>.index.lookup.populated	The total number of lookup index population jobs that have been completed. (counter)
<prefix>.index.text.queried	The total number of times text indexes have been queried. (counter)
<prefix>.index.text.populated	The total number of text index population jobs that have been completed. (counter)
<prefix>.index.range.queried	The total number of times range indexes have been queried. (counter)
<prefix>.index.range.populated	The total number of range index population jobs that have been completed. (counter)
<prefix>.index.point.queried	The total number of times point indexes have been queried. (counter)
<prefix>.index.point.populated	The total number of point index population jobs that have been completed. (counter)
<prefix>.index.vector.queried	The total number of times vector indexes have been queried. (counter)
<prefix>.index.vector.populated	The total number of vector index population jobs that have been completed. (counter)

Server metrics

Table 607. Server metrics

Name	Description
`<prefix>.server.threads.jetty.idle`	The total number of idle threads in the jetty pool. (gauge)
`<prefix>.server.threads.jetty.all`	The total number of threads (both idle and busy) in the jetty pool. (gauge)

Metrics specific to clustering

Catch-up metrics

Table 608. Catch-up metrics

Name	Description
`<prefix>.cluster.catchup.tx_pull_requests_received`	TX pull requests received from other cluster members. (counter)

Discovery metrics v1

Deprecated in 5.23

Table 609. Discovery service V1

Name	Description
`<prefix>.cluster.discovery.replicated_data`	Size of replicated data structures. (gauge)
`<prefix>.cluster.discovery.cluster.members`	Discovery cluster member size. (gauge)
`<prefix>.cluster.discovery.cluster.unreachable`	Discovery cluster unreachable size. (gauge)
`<prefix>.cluster.discovery.cluster.converged`	Discovery cluster convergence. (gauge)
`<prefix>.cluster.discovery.restart.success_count`	Discovery restart count. (gauge)
`<prefix>.cluster.discovery.restart.failed_count`	Discovery restart failed count. (gauge)

Discovery metrics v2

Introduced in 5.22

Table 610. Discovery service V2

Name	Description
`<prefix>.cluster.discovery.memberset.reachable`	Number of members in alive or suspected state. (gauge)

Name	Description
`<prefix>.cluster.discovery.memberset.unreachable`	Number of unreachable cluster members. (gauge)

Raft core metrics

Deprecated in 5.0

Table 611. Raft core metrics

Name	Description
`<prefix>.causal_clustering.core.append_index`	The append index of the Raft log. Each index represents a write transaction (possibly internal) proposed for commitment. The values mostly increase, but sometimes they can decrease as a consequence of leader changes. The append index should always be bigger than or equal to the commit index. (gauge)
`<prefix>.causal_clustering.core.commit_index`	The commit index of the Raft log. Represents the commitment of previously appended entries. Its value increases monotonically if you do not unbind the cluster state. The commit index should always be less than or equal to the append index and bigger than or equal to the applied index. (gauge)
`<prefix>.causal_clustering.core.applied_index`	The applied index of the Raft log. Represents the application of the committed Raft log entries to the database and internal state. The applied index should always be less than or equal to the commit index. The difference between this and the commit index can be used to monitor how up-to-date the follower database is. (gauge)
`<prefix>.causal_clustering.core.term`	The Raft Term of this server. It increases monotonically if you do not unbind the cluster state. (gauge)
`<prefix>.causal_clustering.core.tx_retries`	Transaction retries. (counter)
`<prefix>.causal_clustering.core.is_leader`	Is this server the leader? Track this for each Core cluster member. It will report 0 if it is not the leader and 1 if it is the leader. The sum of all of these should always be 1. However, there will be transient periods in which the sum can be more than 1 because more than one member thinks it is the leader. Action may be needed if the metric shows 0 for more than 30 seconds. (gauge)
`<prefix>.causal_clustering.core.in_flight_cache.total_bytes`	In-flight cache total bytes. (gauge)
`<prefix>.causal_clustering.core.in_flight_cache.max_bytes`	In-flight cache max bytes. (gauge)
`<prefix>.causal_clustering.core.in_flight_cache.element_count`	In-flight cache element count. (gauge)

Name	Description
`<prefix>.causal_clustering.core.in_flight_cache.max_elements`	In-flight cache maximum elements. (gauge)
`<prefix>.causal_clustering.core.in_flight_cache.hits`	In-flight cache hits. (counter)
`<prefix>.causal_clustering.core.in_flight_cache.misses`	In-flight cache misses. (counter)
`<prefix>.causal_clustering.core.raft_log_entry_prefetch_buffer.lag`	Raft Log Entry Prefetch Lag. (gauge)
`<prefix>.causal_clustering.core.raft_log_entry_prefetch_buffer.bytes`	Raft Log Entry Prefetch total bytes. (gauge)
`<prefix>.causal_clustering.core.raft_log_entry_prefetch_buffer.size`	Raft Log Entry Prefetch buffer size. (gauge)
`<prefix>.causal_clustering.core.raft_log_entry_prefetch_buffer.async_put`	Raft Log Entry Prefetch buffer async puts. (gauge)
`<prefix>.causal_clustering.core.raft_log_entry_prefetch_buffer.sync_put`	Raft Log Entry Prefetch buffer sync puts. (gauge)
`<prefix>.causal_clustering.core.message_processing_delay`	Delay between Raft message receive and process. (gauge)
`<prefix>.causal_clustering.core.message_processing_timer`	Timer for Raft message processing. (counter, histogram)
`<prefix>.causal_clustering.core.replication_new`	The total number of Raft replication requests. It increases with write transactions (possibly internal) activity. (counter)
`<prefix>.causal_clustering.core.replication_attempt`	The total number of Raft replication requests attempts. It is bigger or equal than the replication requests. (counter)
`<prefix>.causal_clustering.core.replication_fail`	The total number of Raft replication attempts that have failed. (counter)
`<prefix>.causal_clustering.core.replication_maybe`	Raft Replication maybe count. (counter)
`<prefix>.causal_clustering.core.replication_success`	The total number of Raft replication requests that have succeeded. (counter)
`<prefix>.causal_clustering.core.last_leader_message`	The time elapsed since the last message from a leader in milliseconds. Should reset periodically. (gauge)

Important:

Metrics specific to *Causal Clustering* are deprecated, as the previous table shows. The deprecated Raft core metrics are replaced accordingly by the Raft metrics in the following table.

Raft metrics

Table 612. Raft metrics

Name	Description
`<prefix>.cluster.raft.append_index`	The append index of the Raft log. Each index represents a write transaction (possibly internal) proposed for commitment. The values mostly increase, but sometimes they can decrease as a consequence of leader changes. The append index should always be bigger than or equal to the commit index. (gauge)
`<prefix>.cluster.raft.commit_index`	The commit index of the Raft log. Represents the commitment of previously appended entries. Its value increases monotonically if you do not unbind the cluster state. The commit index should always be less than or equal to the append index and bigger than or equal to the applied index. (gauge)
`<prefix>.cluster.raft.applied_index`	The applied index of the Raft log. Represents the application of the committed Raft log entries to the database and internal state. The applied index should always be less than or equal to the commit index. The difference between this and the commit index can be used to monitor how up-to-date the follower database is. (gauge)
`<prefix>.cluster.raft.prune_index`	Introduced in 5.25 The head index of the Raft log. Represents the oldest Raft index that exists in the log. A prune event will increase this value. This can be used to track how much history of Raft logs the member has. (gauge)
`<prefix>.cluster.raft.term`	The Raft Term of this server. It increases monotonically if you do not unbind the cluster state. (gauge)
`<prefix>.cluster.raft.tx_retries`	Transaction retries. (counter)
`<prefix>.cluster.raft.is_leader`	Is this server the leader? Track this for each rafted primary database in the cluster. It reports `0` if it is not the leader and `1` if it is the leader. The sum of all of these should always be `1`. However, there are transient periods in which the sum can be more than `1` because more than one member thinks it is the leader. Action may be needed if the metric shows `0` for more than 30 seconds. (gauge)
`<prefix>.cluster.raft.in_flight_cache.total_bytes`	In-flight cache total bytes. (gauge)
`<prefix>.cluster.raft.in_flight_cache.max_bytes`	In-flight cache max bytes. (gauge)
`<prefix>.cluster.raft.in_flight_cache.element_count`	In-flight cache element count. (gauge)
`<prefix>.cluster.raft.in_flight_cache.max_elements`	In-flight cache maximum elements. (gauge)

Name	Description
<prefix>.cluster.raft.in_flight_cache.hits	In-flight cache hits. (counter)
<prefix>.cluster.raft.in_flight_cache.misses	In-flight cache misses. (counter)
<prefix>.cluster.raft.raft_log_entry_prefetch_buffer.lag	Raft Log Entry Prefetch Lag. (gauge)
<prefix>.cluster.raft.raft_log_entry_prefetch_buffer.bytes	Raft Log Entry Prefetch total bytes. (gauge)
<prefix>.cluster.raft.raft_log_entry_prefetch_buffer.size	Raft Log Entry Prefetch buffer size. (gauge)
<prefix>.cluster.raft.raft_log_entry_prefetch_buffer.async_put	Raft Log Entry Prefetch buffer async puts. (gauge)
<prefix>.cluster.raft.raft_log_entry_prefetch_buffer.sync_put	Raft Log Entry Prefetch buffer sync puts. (gauge)
<prefix>.cluster.raft.message_processing_delay	Delay between Raft message receive and process. (gauge)
<prefix>.cluster.raft.message_processing_timer	Timer for Raft message processing. (counter, histogram)
<prefix>.cluster.raft.replication_new	The total number of Raft replication requests. It increases with write transactions (possibly internal) activity. (counter)
<prefix>.cluster.raft.replication_attempt	The total number of Raft replication requests attempts. It is bigger or equal to the replication requests. (counter)
<prefix>.cluster.raft.replication_fail	The total number of Raft replication attempts that have failed. (counter)
<prefix>.cluster.raft.replication_maybe	Raft Replication maybe count. (counter)
<prefix>.cluster.raft.replication_success	The total number of Raft replication requests that have succeeded. (counter)
<prefix>.cluster.raft.last_leader_message	The time elapsed since the last message from a leader in milliseconds. Should reset periodically. (gauge)

Read Replica metrics

Deprecated in 5.0

Table 613. Read Replica metrics

Name	Description
<prefix>.causal_clustering.read_replica.pull_updates	The total number of pull requests made by this instance. (counter)
<prefix>.causal_clustering.read_replica.pull_update_highest_tx_id_requested	The highest transaction id requested in a pull update by this instance. (counter)

Name	Description
`<prefix>.causal_clustering.read_replica.pull_update_highest_tx_id_received`	The highest transaction id that has been pulled in the last pull updates by this instance. (counter)

Important:

Metrics specific to *Causal Clustering* are deprecated, as the previous table shows. The deprecated Read Replica metrics are replaced accordingly by the Story copy metrics in the following table.

Store copy metrics

Table 614. Store copy metrics

Name	Description
`<prefix>.cluster.store_copy.pull_updates`	The total number of pull requests made by this instance. (counter)
`<prefix>.cluster.store_copy.pull_update_highest_tx_id_requested`	The highest transaction id requested in a pull update by this instance. (counter)
`<prefix>.cluster.store_copy.pull_update_highest_tx_id_received`	The highest transaction id that has been pulled in the last pull updates by this instance. (counter)

Java Virtual Machine Metrics

The JVM metrics show information about garbage collections (for example, the number of events and time spent collecting), memory pools and buffers, and the number of active threads running. They are environment dependent and therefore, may vary on different hardware and with different JVM configurations. The metrics about the JVM's memory usage expose values that are provided by the MemoryPoolMXBeans and BufferPoolMXBeans. The memory pools are memory managed by the JVM, for example, `neo4j.dbms.vm.memory.pool.g1_survivor_space`. Therefore, if necessary, you can tune them using the JVM settings. The buffer pools are space outside of the memory managed by the garbage collector. Neo4j allocates buffers in those pools as it needs them. You can limit this memory using JVM settings, but there is never any good reason for you to set them.

JVM file descriptor metrics

Table 615. JVM file descriptor metrics

Name	Description
`<prefix>.vm.file.descriptors.count`	The current number of open file descriptors. (gauge)
`<prefix>.vm.file.descriptors.maximum`	(OS setting) The maximum number of open file descriptors. It is recommended to be set to 40K file handles, because of the native and Lucene indexing Neo4j uses. If this metric gets close to the limit, you should consider raising it. (gauge)

GC metrics

Table 616. GC metrics

Name	Description
`<prefix>.vm.gc.time.<gc>`	Accumulated garbage collection time in milliseconds. Long GCs can be an indication of performance issues or potential instability. If this approaches the heartbeat timeout in a cluster, it may cause unwanted leader switches. (counter)
`<prefix>.vm.gc.count.<gc>`	Total number of garbage collections. (counter)

JVM Heap metrics

Table 617. JVM Heap metrics

Name	Description
`<prefix>.vm.heap.committed`	Amount of memory (in bytes) guaranteed to be available for use by the JVM. (gauge)
`<prefix>.vm.heap.used`	Amount of memory (in bytes) currently used. This is the amount of heap space currently used at a given point in time. Monitor this to identify if you are maxing out consistently, in which case, you should increase the initial and max heap size, or if you are underutilizing, you should decrease the initial and max heap sizes. (gauge)
`<prefix>.vm.heap.max`	Maximum amount of heap memory (in bytes) that can be used. This is the amount of heap space currently used at a given point in time. Monitor this to identify if you are maxing out consistently, in which case, you should increase the initial and max heap size, or if you are underutilizing, you should decrease the initial and max heap sizes. (gauge)

JVM memory buffers metrics

Table 618. JVM memory buffers metrics

Name	Description
`<prefix>.vm.memory.buffer.<bufferpool>.count`	Estimated number of buffers in the pool. (gauge)
`<prefix>.vm.memory.buffer.<bufferpool>.used`	Estimated amount of memory used by the pool. (gauge)
`<prefix>.vm.memory.buffer.<bufferpool>.capacity`	Estimated total capacity of buffers in the pool. (gauge)

JVM memory pools metrics

Table 619. JVM memory pools metrics

Name	Description
`<prefix>.vm.memory.pool.<pool>`	Estimated amount of memory in bytes used by the pool. (gauge)

JVM pause time metrics

Table 620. JVM pause time metrics

Name	Description
`<prefix>.vm.pause_time`	Accumulated detected VM pause time. (counter)

JVM threads metrics

Table 621. JVM threads metrics

Name	Description
`<prefix>.vm.threads`	The total number of live threads including daemon and non-daemon threads. (gauge)

Manage queries

List all running queries

The procedure for listing queries, `dbms.listQueries()`, is replaced by the command for listing transactions, `SHOW TRANSACTIONS`. This command returns information about the currently executing query in the transaction. For more information on the command, see the Cypher manual → `SHOW TRANSACTIONS` command.

Terminate queries

Queries are terminated by terminating the transaction on which they are running. This is done using the `TERMINATE TRANSACTIONS transactionIds` command. The `transactionIds` can be found using the `SHOW TRANSACTIONS` command.

The `TERMINATE TRANSACTION` privilege determines what transactions can be terminated. However, the current user can always terminate all of their own transactions.

Syntax:

```
TERMINATE TRANSACTIONS transactionIds
```

Argument:

Name	Type	Description
`transactionIds`	Comma-separated strings	The IDs of all the transactions to be terminated.

Name	Type	Description
`transactionIds`	Single string parameter	The ID of the transaction to be terminated.
`transactionIds`	List parameter	The IDs of all the transactions to be terminated.

For more information on the command, see the Cypher manual → `TERMINATE TRANSACTIONS` command.

Manage connections

List all network connections

An administrator is able to view all network connections within the database instance. Alternatively, the current user may view all of their own network connections.

The procedure `dbms.listConnections` lists all accepted network connections for all configured connectors, including Bolt, HTTP, and HTTPS. Some listed connections might never perform authentication. For example, HTTP GET requests to the Neo4j Browser endpoint fetches static resources and does not need to authenticate. However, connections made using Neo4j Browser require the user to provide credentials and perform authentication. For more information on Neo4j Browser connections, see the Neo4j Browser documentation.

Syntax:

```
CALL dbms.listConnections()
```

Table 622. Data retrieved from a database

Name	Type	Description
`connectionId`	String	This is the ID of the network connection.
`connectTime`	String	This is the time at which the connection was started.
`connector`	String	Name of the connector that accepted the connection.
`username`	String	This is the username of the user who initiated the connection. This field will be null if the transaction was issued using embedded API. It can also be null if connection did not perform authentication.

Name	Type	Description
`userAgent`	String	Name of the software that is connected. For HTTP and HTTPS connections, this information is extracted from the `User-Agent` request header. For Bolt connections, the user agent is available natively and is supplied in an initialization message.
`serverAddress`	String	The server address this connection is connected to.
`clientAddress`	String	The client address of the connection.

Table 623. Default `userAgent` string formats

Neo4j client agent	`userAgent` default string format	Example
Cypher Shell	`"neo4j-cypher-shell/v${version}"`	`"neo4j-cypher-shell/v4.3.0"`
Neo4j Browser	`"neo4j-browser/v${version}"`	`"neo4j-browser/v4.3.0"`
Neo4j Bloom	`"neo4j-bloom/v${version}"`	`"neo4j-bloom/v1.7.0"`
Neo4j Java Driver	`"neo4j-java/x.y.z"`	`"neo4j-java/1.6.3"`
Neo4j .Net Driver	`"neo4j-dotnet/x.y"`	`"neo4j-dotnet/4.3"`
Neo4j Go Driver	`"Go Driver/x.y"`	`"Go Driver/4.3"`
Neo4j Python Driver	`"neo4j-python/x.y Python/x.y.z-a-b (<operating-system>)"`	`"neo4j-python/4.3 Python/3.7.6 (Linux)"`
Neo4j JavaScript Driver	`"neo4j-javascript/x.y.z"`	`"neo4j-javascript/4.3.0"`

Example 148. List all network connections

The following example shows that the user '**alwood**' is connected using Java driver and a Firefox web browser. The procedure call yields specific information about the connection, namely `connectionId`, `connectTime`, `connector`, `username`, `userAgent`, and `clientAddress`.

```
CALL dbms.listConnections() YIELD connectionId, connectTime, connector, username, userAgent,
clientAddress
```

```
+--------------+------------------------+-----------+----------+-----------------+-----------------+--------+
|"connectionId"|"connectTime"           |"connector"|"username"|"userAgent"      |"clientAddress"  |"status"|
+--------------+------------------------+-----------+----------+-----------------+-----------------+--------+
|"bolt-21"     |"2018-10-10T12:11:42.276Z"|"bolt"   |"alwood"  |"neo4j-java/1.6.3"|"127.0.0.1:53929"|"Running"|
+--------------+------------------------+-----------+----------+-----------------+-----------------+--------+
```

```
|"http-11"     |"2018-10-10T12:37:19.014Z"|"http"     |null       |"Mozilla/5.0 (Macintosh; Intel
macOS 10.13; rv:62.0) Gecko/20100101 Firefox/62.0"|"127.0.0.1:54118"|"Running"|
+-------------+--------------------------+-----------+-----------+-----------------------------------------------------------------------------+-----------------+---------+

2 rows
```

Terminate multiple network connections

An administrator is able to terminate within the instance all network connections with any of the given IDs. Alternatively, the current user may terminate all of their own network connections with any of the given IDs.

Syntax:

```
CALL dbms.killConnections(connectionIds)
```

Arguments:

Name	Type	Description
`ids`	List<String>	This is a list of the IDs of all the connections to be terminated.

Returns:

Name	Type	Description
`connectionId`	String	This is the ID of the terminated connection.
`username`	String	This is the username of the user who initiated the (now terminated) connection.
`message`	String	A message stating whether the connection was successfully found.

Considerations:

Bolt connections are stateful. Termination of a Bolt connection results in termination of the ongoing query/transaction.
Termination of an HTTP/HTTPS connection can terminate the ongoing HTTP/HTTPS request.

Example 149. Terminate multiple network connections

The following example shows that the administrator has terminated the connections with IDs '**bolt-37**' and '**https-11**', started by the users '**joesmith**' and '**annebrown**', respectively. The administrator also attempted to terminate the connection with ID '**http-42**' which did not exist.

```
CALL dbms.killConnections(['bolt-37', 'https-11', 'http-42'])
```

"connectionId"	"username"	"message"
"bolt-37"	"joesmith"	"Connection found"
"https-11"	"annebrown"	"Connection found"
"http-42"	"n/a"	"No connection found with this id"

3 rows

Terminate a single network connection

An administrator is able to terminate within the instance any network connection with the given ID. Alternatively, the current user may terminate their own network connection with the given ID.

Syntax:

```
CALL dbms.killConnection(connectionId)
```

Arguments:

Name	Type	Description
`id`	String	This is the ID of the connection to be terminated.

Returns:

Name	Type	Description
`connectionId`	String	This is the ID of the terminated connection.
`username`	String	This is the username of the user who initiated the (now terminated) connection.
`message`	String	A message stating whether the connection was successfully found.

Considerations:

Bolt connections are stateful. Termination of a Bolt connection results in termination of the ongoing query/transaction.
Termination of an HTTP/HTTPS connection can terminate the ongoing HTTP/HTTPS request.

Example 150. Terminate a single network connection

The following example shows that the user '**joesmith**' has terminated his connection with the ID '**bolt-4321**'.

```
CALL dbms.killConnection('bolt-4321')
```

"connectionId"	"username"	"message"
"bolt-4321"	"joesmith"	"Connection found"

1 row

The following example shows the output when trying to kill a connection with an ID that does not exist.

```
CALL dbms.killConnection('bolt-987')
```

"connectionId"	"username"	"message"
"bolt-987"	"n/a"	"No connection found with this id"

1 row

Manage background jobs

There are many types of background jobs performed in the DBMS, many of which are triggered as system jobs by the DBMS itself without any user action. For example, important background jobs include checkpoint or index population. The former is triggered by the DBMS, and the latter can be a result of a user creating or modifying an index definition.

Background jobs are of the following types:

- `IMMEDIATE` - a one-time action, triggered and run in the background.
- `DELAYED` - a one-time action, run in the background at a given point in the future.
- `PERIODIC` - a recurring action, run in the background at a given time interval.

The DBMS provides a way to show active and failed background jobs. Active jobs are those that are currently running, or are scheduled to be delayed or periodic jobs. If a background job fails, or fails to start, the details of the failure are stored in the failed jobs list. Please note that only the last 100 jobs are stored in the failed jobs list, and that this list is not persistent, so it is cleared with a DBMS restart.

Additionally, it should be noted that a single periodic job can contribute multiple times to the failed jobs list.

Listing active background jobs

An administrator can list background jobs active on an instance:

Syntax:

```
CALL dbms.scheduler.jobs()
```

Returns:

Name	Type
`jobId` ID of the job. Can be used to keep track of an active job, and to link a job to a failed job run.	String
`group` A job is a member of a job group. For example, `INDEX_POPULATION`, `LOG_ROTATION` or `RAFT_SERVER`.	String
`submitted` A timestamp for when the job was submitted, in ISO-8601 format.	String
`database` Jobs can have either a database or a DBMS scope: • For database, this column will display the name of the database. • For DBMS, this column will be blank.	String
`submitter` Jobs are either triggered as a result of user action, or as a system job by the DBMS itself. This column will contain a username for jobs triggered by users, or is otherwise blank.	String
`description` A short description of a job that, unlike `currentStateDescription`, does not change during the running of the job.	String
`type` Type of the job. The values can be `IMMEDIATE`, `DELAYED` or `PERIODIC`.	String
`scheduledAt` A timestamp for when a `DELAYED` or `PERIODIC` job will be run, in ISO-8601 format. This column is not applicable to `IMMEDIATE` jobs, and will be blank for that job type.	String
`period` A period of a `PERIODIC` job, in format `hh:mm:ss.sss`.	String
`state` A state of the job. Since this procedure lists only active jobs, they can be either in `SCHEDULED` or `EXECUTING` state. `SCHEDULED` state is applicable only to `DELAYED` or `PERIODIC` jobs, and means that the job is scheduled for a given time in the future.	String
`currentStateDescription` If a job supports reposting its progress, the progress will be reported in this column in a free-form format, specific for each job.	String

Listing failed job executions

An administrator can list job executions failed on an instance:

Syntax:

```
CALL dbms.scheduler.failedJobs()
```

Returns:

Name	Type
`jobId` ID of the failed job.	String
`group` A job is a member of a job group. For example, `INDEX_POPULATION`, `LOG_ROTATION` or `RAFT_SERVER`.	String
`database` Jobs can have either a database or a DBMS scope: • For database, this column will display the name of the database. • For DBMS, this column will be blank.	String
`submitter` Jobs are either triggered as a result of user action, or as a system job by the DBMS itself. This column will contain a username for jobs triggered by users, or is otherwise blank.	String
`description` A short description of a job that, unlike `currentStateDescription`, does not change during the running of the job.	String
`type` Type of the job. The values can be `IMMEDIATE`, `DELAYED` or `PERIODIC`.	String
`submitted` A timestamp for when the job was submitted, in ISO-8601 format.	String
`executionStart` A timestamp for when the failed execution started, in ISO-8601 format.	String
`failureTime` A timestamp for when the execution failed, in ISO-8601 format.	String
`failureDescription` A short description of the failure. If the failure description is insufficient, more information can be found in logs.	String

Tools

This chapter covers the following topics:

- Neo4j Admin and Neo4j CLI — A description of the *Neo4j Admin*, *Neo4j CLI* tools.
- Neo4j Admin commands
  - Consistency checker — How to check the consistency of a Neo4j database using Neo4j Admin.
  - Neo4j Admin report — How to collect the most common information needed for remote assessments.
  - Display store information — How to display information about a database store.
  - Memory recommendations — How to get an initial recommendation for Neo4j memory settings.
  - Import — How to import data into Neo4j using the command `neo4j-admin import`.
  - Unbind a Neo4j cluster server — How to remove cluster state data from a Neo4j server.
  - Upload to Neo4j Aura — How to upload an existing local database to a Neo4j Aura instance.
  - Migrate a database — How to migrate a Neo4j database from one store format to another or to a later `MAJOR` version of the same format.
  - Migrate the Neo4j configuration file — How to migrate a Neo4j configuration file.
  - Validate configurations — How to validate Neo4j configuration files, including the Log4j files.
- Cypher Shell — How to use the Cypher Shell.
- Neo4j Browser is a tool for developers to interact with the graph. It is the default interface for both Enterprise and Community Editions of the Neo4j database. Neo4j Browser is bundled with Neo4j DBMS, including both Neo4j server and Neo4j Desktop.

Neo4j Admin and Neo4j CLI

Introduction

`neo4j-admin` and `neo4j` are command-line tools for managing and administering a Neo4j DBMS. Both are installed as part of the product and can be executed with a number of commands. The `neo4j` commands are equivalent to the most important commands in the `neo4j-admin` server category.

Both `neo4j-admin` and `neo4j` commands support the `--help` option, which prints the command's usage and options, and the `--version` option, which prints the version of the command. All admin command options can also be provided in a file and passed to the command using the `@` prefix. This is useful when the command line becomes too long to manage. For example, `neo4j-admin database import full @/path/to/your/<args-filename> mydb`. For more information, see Picocli → AtFiles official documentation.

Note:

All admin commands must be invoked with the same user as Neo4j runs as. This guarantees that Neo4j will have full rights to start and work with the database files you use.

The `neo4j-admin` tool

The `neo4j-admin` command-line tool is located in the *bin* directory.

General synopsis

`neo4j-admin` has the following general synopsis:

```
neo4j-admin [category] [command] [subcommand]
```

`neo4j-admin` commands per category

All administration commands, except for `help` and `version`, are organized into the following four categories:

- `dbms` - DBMS-wide (for single and clustered environments) administration tasks
- `server` - server-wide administration tasks
- `database` - database-specific administration tasks
- Introduced in 5.25 Enterprise Edition `backup` - backup-specific tasks.

Table 624. Available commands per category

Category	Command	Description
`dbms`	`set-default-admin`	Sets the default admin user when no roles are present.
	`set-initial-password`	Sets the initial password of the initial admin user (`neo4j`). For details, see Set an initial password.
	`unbind-system-db`	Introduced in 5.0 Enterprise Edition Removes and archives the cluster state of the `system` database so the instance can rebind to a new cluster state of the `system` database. For details, see Unbind the `system` database.

Category	Command	Description
`server`	`console`	Starts DBMS server in the console.
	`get-id`	Displays the server ID of an instance. The server ID can be used to specify a server in Cypher commands.
	`license`	Accept the license agreement. Possible options are `--accept-commercial` for the commercial or `--accept-evaluation` for the evaluation license. From Neo4j 5.4 onwards, this command is required to be run before starting the Neo4j Enterprise Edition.
	`memory-recommendation`	Prints recommendations for Neo4j heap and page cache memory usage. For details, see Memory recommendations.
	`migrate-configuration`	Migrates server configuration from the previous major version.
	`report`	Produces a ZIP/TAR of the most common information needed for remote assessments. For details, see Neo4j Admin report.
	`restart`	Restarts the server daemon.
	`start`	Starts the server as a daemon.
	`status`	Gets the status of the server.
	`stop`	Stops the server daemon.
	`unbind`	Enterprise Edition Removes cluster state data from a stopped Neo4j server. For details, see Unbind a Neo4j cluster server.
	`validate-config`	Performs configuration validation without starting the server.
	`windows-service`	A command whose subcommands can be used to install, uninstall, and update Neo4j as a Windows service.

Category	Command	Description
`database`	`aggregate-backup`	Enterprise Edition Aggregates a chain of backup artifacts into a single artifact.
	`backup`	Enterprise Edition Performs an online backup from a running Neo4j enterprise server.
	`check`	Checks the consistency of a database. For details, see Consistency checker.
	`copy`	Enterprise Edition Copies a database and optionally applies filters. For details, see Copy a database store.
	`dump`	Dumps a database into a single-file archive.
	`import`	Imports a collection of CSV files. For details, see Import.
	`info`	Prints information about a Neo4j database store. For details, see Display store information.
	`load`	Loads a database from an archive created with the `dump` command.
	`migrate`	Migrates a database from one store format to another or between versions of the same format.
	`restore`	Enterprise Edition Restores a backed up database.
	`upload`	Pushes a local database to a Neo4j Aura instance. For details, see Upload to Neo4j AuraDB.
Introduced in 5.25 Enterprise Edition `backup`	`inspect`	Introduced in 5.25 The inspect command lists the metadata stored in the header of backup files. For details, see Inspect the metadata of a backup file.

The `neo4j` tool

The `neo4j` command-line tool is located in the *bin* directory.

General synopsis

`neo4j` has the following general synopsis:

```
neo4j [command]
```

`neo4j` commands

The command is an alias for the most important commands in the `neo4j-admin server` category.

Table 625. Equivalence between `neo4j` and `neo4j-admin` commands

`neo4j` command	Equivalent `neo4j-admin` command
`neo4j console`	`neo4j-admin server console`
`neo4j restart`	`neo4j-admin server restart`
`neo4j start`	`neo4j-admin server start`
`neo4j status`	`neo4j-admin server status`
`neo4j stop`	`neo4j-admin server stop`
`neo4j windows-service`	`neo4j-admin server windows-service`

Version command

Version can be obtained by invoking the `version` command, `--version` command option, or its short alternative `-V`, on the root level of both `neo4j` and `neo4j-admin` commands. For example, `neo4j --version`, `neo4j-admin -V`, `neo4j-admin version`, or `neo4j version`.

Help command

Help can be obtained by invoking the `help` command, `--help` command option, or its short alternative `-h`, with both `neo4j` and `neo4j-admin` commands. `--help` and `-h` options can be invoked on any level, namely root, category, command, and subcommand. For example, `neo4j --help`, `neo4j [command] -h`, `neo4j-admin -h`, `neo4j-admin [category] --help`, or `neo4j-admin [category] [command] [subcommand] -h`.

The help command can be invoked on any level except the last one, which means command-level for commands that do not have subcommands or subcommand level for commands with subcommands. The help command also accepts a parameter. For example, `neo4j help`, `neo4j-admin help`, `neo4j-admin [category] help`, `neo4j-admin help [category]`, `neo4j help [command]`, or `neo4j-admin [category] [command ] help [subcommand]`.

Limitations

When using both a multi-value option and a positional parameter, the multi-value option is "greedy" and pulls in the next positional parameter via its convertor. This is a limitation of the underlying library, Picocli, and is not specific to Neo4j Admin. See Picocli → Variable Arity Options and Positional Parameters official documentation for more information.

Configuration

Administration operations use the configuration specified in the *neo4j.conf* file. Sharing configuration between the DBMS and its administration tasks makes sense as most settings are the same. In some cases, however, it is better to override some settings specified in *neo4j.conf* by configuring the tasks

(instead of updating the config settings in the *neo4j.conf* file) because administration tasks generally use fewer resources than the DBMS. For instance, if the page cache of your DBMS is configured to a very high value in *neo4j.conf*, and you want to override this because the admin tasks like backup do not need so much memory, you provide configuration for the admin tasks instead of updating the page cache setting in the *neo4j.conf* file.

There are several options for overriding settings specified in the *neo4j.conf* file using administration tasks:

- `--additional-config` option — almost all administration commands support the `--additional-config` option, which you can use to provide a path (full path, local path, or symlinks) to a file with additional configuration. The file format should be the same as *neo4j.conf* (or *neo4j-admin.conf*). The file must be readable by the user running the admin command.
- *neo4j-admin.conf* — a configuration file located in the same directory as the `neo4j.conf` file, which you can use to provide administration-task-specific settings.
- Some commands also support a command-specific configuration file. Such files are also looked for in the same directory as the *neo4j.conf* file. The following table lists command-specific configuration files:

  Table 626. Command-specific configuration files

Command	Configuration file
`neo4j-admin database backup`	`neo4j-admin-database-backup.conf`
`neo4j-admin database check`	`neo4j-admin-database-check.conf`
`neo4j-admin database copy`	`neo4j-admin-database-copy.conf`
`neo4j-admin database dump`	`neo4j-admin-database-dump.conf`
`neo4j-admin database import`	`neo4j-admin-database-import.conf`
`neo4j-admin database load`	`neo4j-admin-database-load.conf`
`neo4j-admin database migrate`	`neo4j-admin-database-migrate.conf`
`neo4j-admin database restore`	`neo4j-admin-database-restore.conf`

All four configuration sources are optional and settings for administration commands are resolved from them with the following descending priority:

1. `--additional-config` option
2. command-specific configuration file
3. `neo4j-admin.conf`
4. `neo4j.conf`

Note:

The commands for launching the DBMS, `neo4j start` and `neo4j console`, must be configured only in the *neo4j.conf* file.

Environment variables

Neo4j Admin can also use the following environment variables:

Environment variable	Description
`NEO4J_DEBUG`	Set to anything to enable debug output.
`NEO4J_HOME`	Neo4j home directory.
`NEO4J_CONF`	Path to the directory that contains *neo4j.conf*.
`HEAP_SIZE`	Set JVM maximum heap size during command execution. Takes a number and a unit, for example, 512m.
`JAVA_OPTS`	Additional JVM arguments.

If set, `HEAP_SIZE` and `JAVA_OPTS` override all relevant settings specified in the configuration file.

Exit codes

When `neo4j` and `neo4j-admin` finish as expected, they exit with code `0`. A non-zero exit code means something undesired happened during command execution.

Table 627. Exit codes

Exit code	Description
`0`	Successful execution.
`1`	The command failed to execute.
`3`	The command failed to execute because the database is not running.
`64`	The command was invoked with incorrect options/parameters. See the printed usage for details.
`70`	An exception was thrown, not handled otherwise.

The non-zero exit code can contain further information about the error, for example, see the `backup` command's exit codes.

Command-line completion

From 5.4 onwards, Neo4j supports command-line completion.

- For Unix-based systems, the tab completion applies to the `neo4j` and `neo4j-admin` command line interfaces in terminals such as Bash and ZSH.
- For RPM and DEB packaged installations, the necessary files are automatically installed in `bash-completion`.
- For tarball installations, the files are located in the *bin/completion/* directory with detailed instructions for manual installation.

Consistency checker

You can use the `neo4j-admin database check` command to check the consistency of a database, a dump, or a full backup. The `neo4j-admin` tool is located in the */bin* directory.

Syntax

The `neo4j-admin database check` command has the following syntax:

```
neo4j-admin database check [-h] [--expand-commands] [--force] [--verbose]
                           [--check-counts[=true|false]] [--check-graph[=true|false]]
                           [--check-indexes[=true|false]] [--check-property-owners[=true|false]]
                           [--additional-config=<file>] [--max-off-heap-memory=<size>]
                           [--report-path=<path>] [--threads=<number of threads>]
                           [[--from-path-data=<path> --from-path-txn=<path>] | [--from-path=<path> [--
temp-path=<path>]]]
                           <database>
```

Description

This command allows for checking the consistency of a database, a dump, or a full backup. It cannot be used with a database that is currently in use.

Some checks can be quite expensive, so it may be useful to turn some of them off for very large databases. Increasing the heap size might be a good idea.

> **Note:**
>
> It is not recommended to use an NFS to check the consistency of a database, a dump, or a full backup as this slows the process down significantly.

Parameters

Table 628. `neo4j-admin database check` *parameters*

Parameter	Description
`<database>`	Name of the database to check.

Options

The `neo4j-admin database check` command has the following options:

Table 629. `neo4j-admin database check` *options*

Option	Description	Default
`--verbose`	Enable verbose output.	
`-h, --help`	Show this help message and exit.	
`--expand-commands`	Allow command expansion in config value evaluation.	

Option	Description	Default
`--additional-config=<file>`[1]	Configuration file with additional configuration.	
`--force`	Force a consistency check to be run, despite resources, and may run a more thorough check.	
`--check-indexes[=true|false]`	Perform consistency checks on indexes.	`true`
`--check-graph[=true|false]`	Perform consistency checks between nodes, relationships, properties, types, and tokens.	`true`
`--check-counts[=true|false]`	Perform consistency checks on the counts. Requires <check-graph>, and may implicitly enable <check-graph> if it were not explicitly disabled.	`<check-graph>`
`--check-property-owners[=true|false]`	Perform consistency checks on the ownership of properties. Requires <check-graph>, and may implicitly enable <check-graph> if it were not explicitly disabled.	`false`
`--report-path=<path>`	Path to where a consistency report will be written. Interpreted as a directory, unless it has an extension of `.report`.	`.`
`--max-off-heap-memory=<size>`	Maximum memory that `neo4j-admin` can use for page cache and various caching data structures to improve performance. Value can be plain numbers, like `10000000` or e.g. `20G` for 20 gigabytes, or even e.g. `70%`, which will amount to 70% of currently free memory on the machine.	`90%`
`--threads=<number of threads>`	Number of threads used to check the consistency.	`The number of CPUs on the machine.`
`--from-path-data=<path>`	Path to the databases directory, containing the database directory to source from.	`server.directories.data/databases`
`--from-path-txn=<path>`	Path to the transactions directory, containing the transaction directory for the database to source from.	`server.directories.transaction.logs.root`
`--from-path=<path>`	Path to the directory containing dump/backup artifacts that need to be checked for consistency. If the directory contains multiple backups, it will select the most recent backup chain, based on the transaction IDs found, to perform the consistency check.	
`--temp-path=<path>`	Path to directory to be used as a staging area to extract dump/backup artifacts, if needed.	`<from-path>`

Note:

The `--from-path=<path>` option can also check database artifacts in AWS S3 buckets (from Neo4j 5.19) and Google Cloud storage buckets (from Neo4j 5.21). For more information, see Check the consistency of a backup/dump stored in a cloud storage.

Output

If the consistency checker does not find errors, it exits cleanly and does not produce a report. If the consistency checker finds errors, it exits with an exit code other than `0` and writes a report file with a name in the format `inconsistencies-YYYY-MM-DD.HH24.MI.SS.report`. The location of the report file is the current working directory, or as specified by the parameter `report-path`.

Examples

The following are examples of how to check the consistency of a database, a dump, or a backup.

Note:

`neo4j-admin database check` cannot be applied to Composite databases. It must be run directly on the databases that are associated with that Composite database.

Check the consistency of a local database

Note that the database must be stopped first.

```
bin/neo4j-admin database check neo4j
```

The output will look similar to the following:

```
Running consistency check with max off-heap:618.6MiB
  Store size:160.0KiB
  Allocated page cache:160.0KiB
  Off-heap memory for caching:618.5MiB
ID Generator consistency check
..................  10%
..................  20%
..................  30%
..................  40%
..................  50%
..................  60%
..................  70%
..................  80%
..................  90%
.................. 100%
Index structure consistency check
..................  10%
..................  20%
..................  30%
..................  40%
..................  50%
..................  60%
..................  70%
..................  80%
..................  90%
.................. 100%
Consistency check
..................  10%
..................  20%
..................  30%
..................  40%
..................  50%
..................  60%
..................  70%
..................  80%
..................  90%
```

```
..................  100%
```

Check the consistency of a backup/dump

> **Note:**
>
> Note that consistency check is not supported for differential backups. The backup chain must be aggregated into a full backup artifact before running consistency check.

Run with the `--from-path` option to check the consistency of a backup or a dump:

```
bin/neo4j-admin database check --from-path=<directory-with-backup-or-dump> neo4j
```

Check the consistency of a backup/dump stored in a cloud storage

The following examples show how to check the consistency of a full backup or a dump stored in a cloud storage bucket using the `--from-path` option.

> **Note:**
>
> Neo4j uses the AWS SDK v2 to call the APIs on AWS using AWS URLs. Alternatively, you can override the endpoints so that the AWS SDK can communicate with alternative storage systems, such as Ceph, Minio, or LocalStack, using the system variables `aws.endpointUrls3`, `aws.endpointUrlS3`, or `aws.endpointUrl`, or the environments variables `AWS_ENDPOINT_URL_S3` or `AWS_ENDPOINT_URL`.

1. Install the AWS CLI by following the instructions in the AWS official documentation — Install the AWS CLI version 2.
2. Create an S3 bucket and a directory to store the backup files using the AWS CLI:

   ```
   aws s3 mb --region=us-east-1 s3://myBucket
   aws s3api put-object --bucket myBucket --key myDirectory/
   ```

   For more information on how to create a bucket and use the AWS CLI, see the AWS official documentation — Use Amazon S3 with the AWS CLI and Use high-level (s3) commands with the AWS CLI.

3. Verify that the `~/.aws/config` file is correct by running the following command:

   ```
   cat ~/.aws/config
   ```

   The output should look like this:

   ```
   [default]
   ```

```
region=us-east-1
```

4. Configure the access to your AWS S3 bucket by setting the `aws_access_key_id` and `aws_secret_access_key` in the `~/.aws/credentials` file and, if needed, using a bucket policy. For example:

   a. Use `aws configure set aws_access_key_id aws_secret_access_key` command to set your IAM credentials from AWS and verify that the `~/.aws/credentials` is correct:

   ```
   cat ~/.aws/credentials
   ```

   The output should look like this:

   ```
   [default]
   aws_access_key_id=this.is.secret
   aws_secret_access_key=this.is.super.secret
   ```

   b. Additionally, you can use a resource-based policy to grant access permissions to your S3 bucket and the objects in it. Create a policy document with the following content and attach it to the bucket. Note that both resource entries are important to be able to download and upload files.

   ```
   {
       "Version": "2012-10-17",
       "Id": "Neo4jBackupAggregatePolicy",
       "Statement": [
           {
               "Sid": "Neo4jBackupAggregateStatement",
               "Effect": "Allow",
               "Action": [
                   "s3:ListBucket",
                   "s3:GetObject",
                   "s3:PutObject",
                   "s3:DeleteObject"
               ],
               "Resource": [
                   "arn:aws:s3:::myBucket/*",
                   "arn:aws:s3:::myBucket"
               ]
           }
       ]
   }
   ```

5. Run the `bin/neo4j-admin database check` command to check the consistency of your database located in your AWS S3 storage bucket. The example assumes that you have backup or dump artifacts located in the `myBucket/myDirectory` folder in your bucket.

   ```
   bin/neo4j-admin database check mydatabase --from-path=s3://myBucket/myDirectory/
   ```

1. Ensure you have a Google account and a project created in the Google Cloud Platform (GCP).

   a. Install the `gcloud` CLI by following the instructions in the Google official documentation — Install the gcloud CLI.

b. Create a service account and a service account key using Google official documentation — Create service accounts and Creating and managing service account keys.

c. Download the JSON key file for the service account.

d. Set the `GOOGLE_APPLICATION_CREDENTIALS` and `GOOGLE_CLOUD_PROJECT` environment variables to the path of the JSON key file and the project ID, respectively:

```
export GOOGLE_APPLICATION_CREDENTIALS="/path/to/keyfile.json"
export GOOGLE_CLOUD_PROJECT=YOUR_PROJECT_ID
```

e. Authenticate the `gcloud` CLI with the e-mail address of the service account you have created, the path to the JSON key file, and the project ID:

```
gcloud auth activate-service-account service-account@example.com --key-file
=$GOOGLE_APPLICATION_CREDENTIALS --project=$GOOGLE_CLOUD_PROJECT
```

For more information, see the Google official documentation — gcloud auth activate-service-account.

f. Create a bucket in the Google Cloud Storage using Google official documentation — Create buckets.

g. Verify that the bucket is created by running the following command:

```
gcloud storage ls
```

The output should list the created bucket.

2. Run the `bin/neo4j-admin database check` command to check the consistency of your database located in your Google storage bucket. The example assumes that you have backup or dump artifacts located in the `myBucket/myDirectory` folder in your bucket.

```
bin/neo4j-admin database check mydatabase --from-path=gs://myBucket/myDirectory/
```

1. Ensure you have an Azure account, an Azure storage account, and a blob container.

   a. You can create a storage account using the Azure portal.
   For more information, see the Azure official documentation on Create a storage account.

   b. Create a blob container in the Azure portal.
   For more information, see the Azure official documentation on Quickstart: Upload, download, and list blobs with the Azure portal.

2. Install the Azure CLI by following the instructions in the Azure official documentation — Azure official documentation.

3. Authenticate the neo4j or neo4j-admin process against Azure using the default Azure credentials.
   See the Azure official documentation on default Azure credentials for more information.

```
az login
```

Then you should be ready to use Azure URLs in either neo4j or neo4j-admin.

4. To validate that you have access to the container with your login credentials, run the following commands:

```
# Upload a file:
az storage blob upload --file someLocalFile  --account-name accountName - --container
someContainer --name remoteFileName  --auth-mode login

# Download the file
az storage blob download  --account-name accountName --container someContainer --name
remoteFileName --file downloadedFile --auth-mode login

# List container files
az storage blob list  --account-name someContainer --container someContainer  --auth-mode
login
```

5. Run the `bin/neo4j-admin database check` command to check the consistency of your database located in your Azure blob storage container. The example assumes that you have backup or dump artifacts located in the `myStorageAccount/myContainer/myDirectory` folder in Azure.

```
bin/neo4j-admin database check mydatabase --from-path
=azb://myStorageAccount/myContainer/myDirectory/
```

Neo4j Admin report

The `neo4j-admin server report` command collects information about the Neo4j deployment and saves it to an archive (ZIP/TAR) for remote assessments.

Syntax

The `neo4j-admin server report` command has the following syntax:

```
neo4j-admin server report [-h]
                         [--expand-commands] [--list] [--verbose]
                         [--ignore-disk-space-check[=true|false]]
                         [--additional-config=<file>] [--database=<database>]
                         [--to-path=<path>] [<classifier>...] [COMMAND]
```

Description

The command collects information about the system and packages everything in an archive. If you specify `all`, everything is included. You can also finetune the selection by passing classifiers to the tool, e.g `logs tx threads`.

Starting from Neo4j version 5.16, the Neo4j Admin report tool allows you to choose the databases for which you want to include database-specific information. You can generate a report covering all databases in the DBMS, a specific database, or databases that match a specified pattern. For example, if you run the

command `neo4j-admin server report --database=ne*`, a report will be generated for all databases that start with "ne". If not specified, the tool generates a report for all databases in the DBMS.

> **Note:**
>
> The `--database` option determines from which database(s) the database-level information is collected. It does not influence the DBMS-level information, which is gathered based on the specified classifiers.

This tool does not send any information automatically. To share this information with the Neo4j Support organization, you have to send it manually.

Parameters

Table 630. `neo4j-admin server report` parameters

Parameter	Default
`[<classifier>…]`	`[config, logs, metrics, plugins, ps, sysprop, threads, tree, version]`

Table 631. Classifiers

Classifier	Online	Description
`all`		Include all of the available classifiers.
`ccstate`		Include the current cluster state.
`config`		Include Neo4j configuration files.
`heap`	✔	Include a heap dump.
`logs`		Include log files, e.g., *debug.log*, *neo4j.log*, etc.
`metrics`		Include the collected metrics.
`plugins`		Include a text view of the plugin directory (no files are collected).
`ps`		Include a list of running processes.
`raft`		Include the raft log.
`sysprop`	✔	Include a list of Java system properties.
`threads`	✔	Include a thread dump of the running instance.
`tree`		Include a text view of the folder structure of the data directory (no files are collected).
`tx`		Include transaction logs.

The classifiers marked as *Online* work only when you have a running Neo4j instance that the tool can find.

If no classifiers are specified, the following classifiers are used: `logs`, `config`, `plugins`, `tree`, `metrics`, `threads`, `sysprop`, `ps`, and `version`.

The reporting tool does not read any data from your database. However, the heap, the raft logs, and the

transaction logs may contain data. Additionally, even though the standard *neo4j.conf* file does not contain password information, for specific configurations, it may have this type of information. Therefore, be aware of your organization's data security rules before using the classifiers `heap`, `tx`, `raft`, and `config`.

Options

The `neo4j-admin server report` command has the following options:

Table 632. `neo4j-admin server report` options

Option	Description	Default
`--additional-config=<file>`[2]	Configuration file with additional configuration.	
`--database=<database>`	Name of the database to report for. Can contain * and ? for globbing. Note that * and ? have special meaning in some shells and might need to be escaped or used with quotes.	`*`
`--expand-commands`	Allow command expansion in config value evaluation.	
`-h, --help`	Show this help message and exit.	
`--ignore-disk-space-check[=true|false]`	Ignore disk full warning.	`false`
`--list`	List all available classifiers.	
`--to-path=<path>`	Destination directory for reports. Defaults to a system tmp directory.	
`--verbose`	Enable verbose output.	

By default, the tool tries to estimate the final size of the report and uses that to assert that there is enough disk space available for it. If there is not enough available space, the tool aborts. However, this estimation is pessimistic and does not consider compression. Therefore, if you are confident that you do have enough disk space, you can disable this check with the option `--ignore-disk-space-check`.

Examples

Note:

This tool uses the Java Attach API to gather data from a running Neo4j instance. Therefore, it requires the Java JDK to run properly.

Example 151. Invoke `neo4j-admin server report` using the default classifiers

The following command gathers information about the Neo4j instance using the default classifiers and saves it to the default location:

```
bin/neo4j-admin server report
```

Example 152. Invoke `neo4j-admin server report` using all classifiers

The following command gathers information about the Neo4j instance using all classifiers and saves it to a specified location:

```
bin/neo4j-admin server report --to-path=./report all
```

Example 153. Invoke `neo4j-admin server report` to gather only logs and thread dumps

The following command gathers only logs and thread dumps from the running Neo4j instance and saves it to a specified location:

```
bin/neo4j-admin server report --to-path=./report threads logs
```

Display store information

You can use the `neo4j-admin database info` command to print information about a Neo4j database store. The following information about the *store format* of a given *database store* can be retrieved:

- The store format version.
- When the store format version was introduced.
- Whether the store format needs to be migrated to a newer version.

For more information on database store formats, see Store formats.

The `neo4j-admin database info` command is located in the *bin* directory.

Syntax

The `neo4j-admin database info` command should be invoked against an **offline** database store or a backup and has the following syntax:

```
neo4j-admin database info [-h] [--expand-commands] [--verbose]
                          [--additional-config=<file>]
                          [--format=text|json]
                          [--from-path=<path>] [<database>]
```

Parameters

Table 633. `neo4j-admin database info` parameters

Parameter	Description	Default
`[<database>]`	Name of the database to show info for. Can contain * and ? for globbing. Note that * and ? have special meaning in some shells and might need to be escaped or used with quotes.	*

Options

The `neo4j-admin database info` command has the following options:

Table 634. `neo4j-admin database info` options

Option	Description	Default
`--additional-config=<file>`[3]	Configuration file with additional configuration.	
`--expand-commands`	Allow command expansion in config value evaluation.	
`--format=text|json`	The format of the returned information.	`text`
`--from-path=<path>`	Path to databases directory.	
`-h, --help`	Show this help message and exit.	
`--verbose`	Enable verbose output.	

Examples

The following examples show how to use the `neo4j-admin database info` command to display information about a database and its store format. All examples assume that the Neo4j server is Enterprise Edition and that the database is offline.

Example 154. Invoke `neo4j-admin database info` against a database store

```
bin/neo4j-admin database info healthcare
```

Output

```
Database name:                healthcare
Database in use:              false
Store format version:         block-block-1.1
Store format introduced in:   5.14.0
Last committed transaction id:29
Store needs recovery:         false
```

Example 155. Invoke `neo4j-admin database info` against all databases

```
bin/neo4j-admin database info --from-path=../data/databases
```

Output

```
Database name:                healthcare
Database in use:              false
Store format version:         block-block-1.1
Store format introduced in:   5.14.0
Last committed transaction id:29
Store needs recovery:         false

Database name:                neo4j
Database in use:              false
Store format version:         block-block-1.1
Store format introduced in:   5.14.0
```

```
Last committed transaction id:27
Store needs recovery:         false

Database name:                system
Database in use:              false
Store format version:         record-aligned-1.1
Store format introduced in:   5.0.0
Last committed transaction id:213
Store needs recovery:         false
```

Note:

When the command is invoked against several databases, if some are **online** they will simply report as `in use` and exclude all other information.

Example 156. Invoke `neo4j-admin database info` against a database and output JSON

If you are parsing the results of this command, you may use the `--format=json` option to receive the output as JSON. All the same fields are included and all values are strings.

```
bin/neo4j-admin database info --from-path ../data/databases --format=json foo
```

Output

```
{"databaseName":"healthcare","inUse":"false","storeFormat":"block-block-
1.1","storeFormatIntroduced":"5.14.0","storeFormatSuperseded":null,"lastCommittedTransaction":"29","r
ecoveryRequired":"false"}
```

Memory recommendations

You can use the `neo4j-admin server memory-recommendation` command to get an initial recommendation on how to configure the memory parameters of your Neo4j DBMS.

Syntax

The `neo4j-admin server memory-recommendation` command has the following syntax:

```
neo4j-admin server memory-recommendation [-h] [--docker] [--expand-commands]
                                         [--verbose] [--additional-config=<file>]
                                         [--memory=<size>]
```

Description

The command prints heuristic memory settings recommendations for the Neo4j JVM heap and pagecache. It either uses the total system memory or the amount of memory specified in the `--memory` argument. The heuristic assumes that the system is dedicated to running Neo4j. If this is not the case, then use the --memory argument to specify how much memory can be expected to be dedicated to Neo4j. The output is formatted such that it can be copy-pasted into the *neo4j.conf* file.

The argument `--docker` outputs environmental variables that can be passed to a Neo4j Docker container.

For a detailed example, see Use Neo4j Admin for memory recommendations.

Options

The `neo4j-admin server memory-recommendation` command has the following options:

Table 635. `neo4j-admin server memory-recommendation` options

Option	Description	Default
`--additional-config=<file>`[4]	Configuration file with additional configuration.	
`--docker`	The recommended memory settings are produced in the form of environment variables that can be directly passed to a Neo4j docker container. The recommended use is to save the generated environment variables to a file and pass the file to a docker container using the `--env-file` docker option.	
`--expand-commands`	Allow command expansion in config value evaluation.	
`-h, --help`	Show this help message and exit.	
`--memory=<size>`	Recommend memory settings with respect to the given amount of memory, instead of the total memory of the system running the command. Valid units are: `k`, `K`, `m`, `M`, `g`, `G`.	`The memory capacity of the machine.`
`--verbose`	Enable verbose output.	

Considerations

The `neo4j-admin server memory-recommendation` command calculates a valid starting point for the Neo4j memory settings, based on the provided memory. The specific conditions for your use case may warrant adjustment of these values. See Memory configuration for a description of the memory settings in Neo4j.

Example

Example 157. Use the `memory-recommendation` command of `neo4j-admin`

The following example illustrates how `neo4j-admin server memory-recommendation` provides a recommendation on how to use 16g of memory:

```
bin/neo4j-admin server memory-recommendation --memory=16g

...
...
...
# Based on the above, the following memory settings are recommended:
server.memory.heap.initial_size=5g
server.memory.heap.max_size=5g
server.memory.pagecache.size=7g
```

> **Note:**
>
> For an example of how to use the `neo4j-admin server memory-recommendation` command, see Inspect the memory settings of all databases in a DBMS.

Import

`neo4j-admin database import` writes CSV data into Neo4j's native file format as fast as possible. Starting with version 5.26, Neo4j also provides support for the Parquet file format.

You should use this tool when:

- Import performance is important because you have a large amount of data (millions/billions of entities).
- The database can be taken offline and you have direct access to one of the servers hosting your Neo4j DBMS.
- The database is non-existent or empty and you need to perform the initial data load.
- You need to update your graph with a large amount of data. In this case, importing data incrementally can be more performant than transactional insertion.

  > **Note:**
  >
  > The incremental import can be done either within a single command or in stages. For details, see Incremental import in a single command and Incremental import in stages.

- The CSV data is clean/fault-free (nodes are not duplicated and relationships' start and end nodes exist). This tool can handle data faults but performance is not optimized. If your data has a lot of faults, it is recommended to clean it using a dedicated tool before import.

Other methods of importing data into Neo4j might be better suited to non-admin users:

- Cypher® - CSV data can be bulk loaded via the Cypher command `LOAD CSV`. See Cypher Manual → `LOAD CSV`.
- Graphical Tools - Neo4j AuraDB → Importing data.

> **Note:**
>
> Change Data Capture does **not** capture any data changes resulting from the use of `neo4j-admin database import`. See Change Data Capture → Key considerations for more information.

Overview

The `neo4j-admin database import` command has two modes both used for initial data import:

- *full* — used to import data into a non-existent empty database.

- *incremental* — used when import cannot be completed in a single *full* import, by allowing the import to be a series of smaller imports.

> **Note:**
>
> The user running `neo4j-admin database import` must have `WRITE` capabilities into `server.directories.data` and `server.directories.log`.

This section describes the `neo4j-admin database import` option.

> **Tip:**
>
> For information on `LOAD CSV`, see the Cypher Manual → `LOAD CSV`. For in-depth examples of using the command `neo4j-admin database import`, refer to the Tutorials → Neo4j Admin import.

These are some things you need to keep in mind when creating your input files:

- Fields are comma-separated by default but a different delimiter can be specified.
- All files must use the same delimiter.
- Multiple data sources can be used for both nodes and relationships.
- A data source can optionally be provided using multiple files.
- A separate file with a header that provides information on the data fields, must be the first specified file of each data source.
- Fields without corresponding information in the header are not read.
- UTF-8 encoding is used.
- By default, the importer trims extra whitespace at the beginning and end of strings. Quote your data to preserve leading and trailing whitespaces.

> **Note:**
>
> **Indexes and constraints**
>
> Indexes and constraints are not created during the import. Instead, you have to add these afterward (see Cypher Manual → Indexes).
>
> Starting from Neo4j 5.24, you can use the `--schema` option to create indexes and contraints during the import process. The option is available in the Enterprise Edition and works only for the `block` format. See Provide indexes and constraints during import for more information.
>
> Vector indexes can take advantage of the incubated Java Vector API for noticeable speed improvements. If you are using a compatible version of Java, you can uncomment the following value of the `server.jvm.additional` setting in the *neo4j-admin.conf* file:

```
server.jvm.additional=--add-modules=jdk.incubator.vector
```

Full import

Syntax

The syntax for importing a set of CSV files is:

```
neo4j-admin database import full [-h] [--expand-commands] [--verbose] [--auto-skip-subsequent-
headers[=true|false]]
                                 [--ignore-empty-strings[=true|false]] [--ignore-extra-
columns[=true|false]]
                                 [--legacy-style-quoting[=true|false]] [--normalize-types[=true|false]]
                                 [--overwrite-destination[=true|false]] [--skip-bad-entries-
logging[=true|false]]
                                 [--skip-bad-relationships[=true|false]] [--skip-duplicate-
nodes[=true|false]] [--strict
                                 [=true|false]] [--trim-strings[=true|false]] [--additional-config=<file>]
                                 [--array-delimiter=<char>] [--bad-tolerance=<num>] [--delimiter=<char>]
                                 [--format=<format>] [--high-parallel-io=on|off|auto] [--id-
type=string|integer|actual]
                                 [--input-encoding=<character-set>] [--input-type=csv|parquet]
                                 [--max-off-heap-memory=<size>] [--quote=<char>] [--read-buffer-
size=<size>]
                                 [--report-file=<path>] [--schema=<path>] [--threads=<num>]
--nodes=[<label>[:
                                 <label>]...=]<files>... [--nodes=[<label>[:<label>]...=]<files>...]...
                                 [--relationships=[<type>=]<files>...]... [--multiline-
fields=true|false|<path>[,
                                 <path>] [--multiline-fields-format=v1|v2]] <database>
```

Description

Initial import into a non-existent empty database.

Parameters

Table 636. `neo4j-admin database import full` parameters

Parameter	Description	Default
`<database>`	Name of the database to import. If the database into which you import does not exist prior to importing, you must create it subsequently using `CREATE DATABASE`.	`neo4j`

Warning:

Some of the options below are marked as **Advanced**. These options should not be used for experimentation.

For more information, please contact Neo4j Professional Services.

Options

Starting from Neo4j 5.26, the importer also supports the Parquet file format. An additional parameter

`--input-type=csv|parquet` has been introduced to explicitly specify whether to use CSV or Parquet for the importer. If not defined, the default value will be CSV. The examples for CSV can also be used with Parquet.

Table 637. `neo4j-admin database import full` options

Option	Description	Default
`--additional-config=<file>`[5]	Configuration file with additional configuration.	
`--array-delimiter=<char>`	Delimiter character between array elements within a value in CSV data. Also accepts `TAB` and e.g. `U+20AC` for specifying a character using Unicode. • ASCII character — e.g. `--array-delimiter=";"`. • `\ID` — Unicode character with ID, e.g. `--array-delimiter="\59"`. • `U+XXXX` — Unicode character specified with 4 HEX characters, e.g. `--array-delimiter="U+20AC"`. • `\t` — horizontal tabulation (HT), e.g. `--array-delimiter="\t"`. For horizontal tabulation (HT), use `\t` or the Unicode character ID `\9`. Unicode character ID can be used if prepended by `\`.	`;`
`--auto-skip-subsequent-headers[=true|false]`[6]	Automatically skip accidental header lines in subsequent files in file groups with more than one file.	`false`
`--bad-tolerance=<num>`	Number of bad entries before the import is aborted. The import process is optimized for error-free data. Therefore, cleaning the data before importing it is highly recommended. If you encounter any bad entries during the import process, you can set the number of bad entries to a specific value that suits your needs. However, setting a high value may affect the performance of the tool.	`1000`
`--delimiter=<char>`[6]	Delimiter character between values in CSV data. Also accepts `TAB` and e.g. `U+20AC` for specifying a character using Unicode. • ASCII character — e.g. `--delimiter=","`. • `\ID` — Unicode character with ID, e.g. `--delimiter="\44"`. • `U+XXXX` — Unicode character specified with 4 HEX characters, e.g. `--delimiter="U+20AC"`. • `\t` — horizontal tabulation (HT), e.g. `--delimiter="\t"`. For horizontal tabulation (HT), use `\t` or the Unicode character ID `\9`. Unicode character ID can be used if prepended by `\`.	`,`
`--expand-commands`	Allow command expansion in config value evaluation.	

Option	Description	Default
`--format=<format>`	Name of database format. The imported database will be created in the specified format or use the format set in the configuration. Valid formats are `standard`, `aligned`, `high_limit`, and `block`.	
`-h, --help`	Show this help message and exit.	
`--high-parallel-io=on|off|auto`	Ignore environment-based heuristics and indicate if the target storage subsystem can support parallel IO with high throughput or auto detect. Typically this is `on` for SSDs, large raid arrays, and network-attached storage.	`auto`
`--id-type=string|integer|actual`	Each node must provide a unique ID. This is used to find the correct nodes when creating relationships. Possible values are: • `string` — arbitrary strings for identifying nodes. • `integer` — arbitrary integer values for identifying nodes. • `actual` — (advanced) actual node IDs.	`string`
`--ignore-empty -strings[=true|false]`	Whether or not empty string fields, i.e. "" from input source are ignored, i.e. treated as null.	`false`
`--ignore-extra -columns[=true|false]`[6]	If unspecified columns should be ignored during the import.	`false`
`--input-encoding=<character -set>`[6]	Character set that input data is encoded in.	`UTF-8`
`--input-type=csv|parquet`	Introduced in 5.26 File type to import from. Can be csv or parquet. Defaults to csv.	`csv`
`--legacy-style -quoting[=true|false]`	Whether or not a backslash-escaped quote e.g. \" is interpreted as an inner quote.	`false`
`--max-off-heap-memory=<size>`	Maximum memory that `neo4j-admin` can use for various data structures and caching to improve performance. Values can be plain numbers, such as `10000000`, or `20G` for 20 gigabytes. It can also be specified as a percentage of the available memory, for example `70%`.	`90%`
`--multiline -fields=true|false|<path>[,<path>]`[6]	Changed in 5.26 In v1, whether or not fields from an input source can span multiple lines, i.e. contain newline characters. Setting `--multiline -fields=true` can severely degrade the performance of the importer. Therefore, use it with care, especially with large imports. In v2, this option will specify the list of files that contain multiline fields. Files can also be specified using regular expressions.	`null`
`--multiline-fields -format=v1|v2`[6]	Introduced in 5.26 Controls the parsing of input source that can span multiple lines, i.e. contain newline characters. When set to v1, the value for `--multiline-fields` can only be true or false. When set to v2, the value for `--multiline-fields` should be the list of files that contain multiline fields.	`null`

Option	Description	Default
`--nodes=[<label>[:<label>]…=]<files>…`	Node CSV header and data. • Multiple files will be logically seen as one big file from the perspective of the importer. • The first line must contain the header. • Multiple data sources like these can be specified in one import, where each data source has its own header. • Files can also be specified using regular expressions. It is possible to import files from AWS S3 buckets, Google Cloud storage buckets, and Azure buckets using the appropriate URI as the path. For an example, see Import data from CSV files using regular expression.	
`--normalize-types[=true|false]`	When `true`, non-array property values are converted to their equivalent Cypher types. For example, all integer values will be converted to 64-bit long integers.	`true`
`--overwrite-destination[=true|false]`	Delete any existing database files prior to the import.	`false`
`--quote=<char>` [6] [7]	Character to treat a quotation mark for values in CSV data. For example, quotes can be escaped as per RFC 4180 by doubling them. Thus `""` would be interpreted as a literal `"`. You cannot escape using `\`.	`"`
`--read-buffer-size=<size>`	Size of each buffer for reading input data. It has to be at least large enough to hold the biggest single value in the input data. The value can be a plain number or a byte units string, e.g. `128k`, `1m`.	`4194304`
`--relationships=[<type>=]<files>…`	Relationship CSV header and data. • Multiple files will be logically seen as one big file from the perspective of the importer. • The first line must contain the header. • Multiple data sources like these can be specified in one import, where each data source has its own header. • Files can also be specified using regular expressions. It is possible to import files from AWS S3 buckets, Google Cloud storage buckets, and Azure buckets using the appropriate URI as the path. For an example, see Import data from CSV files using regular expression.	

Option	Description	Default
`--report-file=<path>`	File in which to store the report of the csv-import. The location of the import log file can be controlled using the `--report-file` option. If you run large imports of CSV files that have low data quality, the import log file can grow very large. For example, CSV files that contain duplicate node IDs, or that attempt to create relationships between non-existent nodes, could be classed as having low data quality. In these cases, you may wish to direct the output to a location that can handle the large log file. If you are running on a UNIX-like system and you are not interested in the output, you can get rid of it altogether by directing the report file to `/dev/null`. If you need to debug the import, it might be useful to collect the stack trace. This is done by using the `--verbose` option.	`import.report`
`--schema=<path>`	Introduced in 5.24 Enterprise edition Path to the file containing the Cypher commands for creating indexes and constraints during data import.	
`--skip-bad-entries-logging[=true|false]`	When set to `true`, the details of bad entries are not written in the log. Disabling logging can improve performance when the data contains lots of faults. Cleaning the data before importing it is highly recommended because faults dramatically affect the tool's performance even without logging.	`false`
`--skip-bad-relationships[=true|false]`	Whether or not to skip importing relationships that refer to missing node IDs, i.e. either start or end node ID/group referring to a node that was not specified by the node input data. Skipped relationships will be logged, containing at most the number of entities specified by `--bad-tolerance`, unless otherwise specified by the `--skip-bad-entries-logging` option.	`false`
`--skip-duplicate-nodes[=true|false]`	Whether or not to skip importing nodes that have the same ID/group. In the event of multiple nodes within the same group having the same ID, the first encountered will be imported, whereas consecutive such nodes will be skipped. Skipped nodes will be logged, containing at most the number of entities specified by `--bad-tolerance`, unless otherwise specified by the `--skip-bad-entries-logging` option.	`false`
`--strict[=true|false]`	Introduced in 5.6 Whether or not the lookup of nodes referred to from relationships needs to be checked strict. If disabled, most but not all relationships referring to non-existent nodes will be detected. If enabled all those relationships will be found but at the cost of lower performance.	`false` Changed in 5.8
`--threads=<num>`	(advanced) Max number of worker threads used by the importer. Defaults to the number of available processors reported by the JVM. There is a certain amount of minimum threads needed so for that reason there is no lower bound for this value. For optimal performance, this value should not be greater than the number of available processors.	`20`

Option	Description	Default
`--trim-strings[=true|false]`[6]	Whether or not strings should be trimmed for whitespaces.	`false`
`--verbose`	Enable verbose output.	

Note:

Heap size for the import

You want to set the maximum heap size to a relevant value for the import. This is done by defining the `HEAP_SIZE` environment parameter before starting the import. For example, 2G is an appropriate value for smaller imports.

If doing imports in the order of magnitude of 100 billion entities, 20G will be an appropriate value.

Note:

Record format

If your import data results in a graph that is larger than 34 billion nodes, 34 billion relationships, or 68 billion properties, you will need to configure the importer to use the `block` format. This is achieved by using the `format` option of the import command and setting the value to `block`:

```
bin/neo4j-admin database import full --format=block
```

The `block` format is available in Enterprise Edition only.

Note:

Providing arguments in a file

All options can be provided in a file and passed to the command using the `@` prefix. This is useful when the command line becomes too long to manage. For example, the following command:

```
bin/neo4j-admin database import full @/path/to/your/<args-filename> mydb
```

For more information, see Picocli → AtFiles official documentation.

Note:

Using both a multi-value option and a positional parameter

When using both a multi-value option, such as `--nodes` and `--relationships`, and a

positional parameter (for example, in `--additional-config neo4j.properties --nodes 0-nodes.csv mydatabase`), the `--nodes` option acts "greedy" and the next option, in this case, `mydatabase`, is pulled in via the nodes convertor.

This is a limitation of the underlying library, Picocli, and is not specific to Neo4j Admin. For more information, see Picocli → Variable Arity Options and Positional Parameters official documentation.

To resolve the problem, use one of the following solutions:

- Put the positional parameters first. For example, `mydatabase --nodes 0-nodes.csv`.
- Put the positional parameters last, after `--` and the final value of the last multi-value option. For example, `nodes 0-nodes.csv -- mydatabase`.

Note:

Importing from a cloud storage

The `--nodes` and `--relationships` options can also import files from AWS S3 buckets (from Neo4j 5.19), Google Cloud storage buckets (from Neo4j 5.21), and Azure buckets (from Neo4j 5.24). For more information, see Importing files from a cloud storage.

Examples

Note:

If importing to a database that has not explicitly been created before the import, it must be created subsequently in order to be used.

Import data from CSV files

Assume that you have formatted your data as per CSV header format so that you have it in six different files:

1. `movies_header.csv`
2. `movies.csv`
3. `actors_header.csv`
4. `actors.csv`
5. `roles_header.csv`
6. `roles.csv`

The following command imports the three datasets:

```
bin/neo4j-admin database import full --nodes import/movies_header.csv,import/movies.csv \
--nodes import/actors_header.csv,import/actors.csv \
--relationships import/roles_header.csv,import/roles.csv
```

Provide indexes and constraints during import

Enterprise Edition | Introduced in 5.24

Starting from Neo4j 5.24, you can use the `--schema` option that allows Cypher commands to be provided to create indexes/constraints during the initial import process. It currently only works for the block format and full import.

You should have a Cypher script containing only `CREATE INDEX|CONSTRAINT` commands to be parsed and executed. This file uses ';' as the separator.

For example:

```
CREATE INDEX PersonNameIndex FOR (i:Person) ON (i.name);
CREATE CONSTRAINT PersonAgeConstraint FOR (c:Person) REQUIRE c.age IS :: INTEGER
```

List of supported indexes and constraints that can be created by the import tool:

- RANGE
- LOOKUP
- POINT
- TEXT
- FULL-TEXT
- VECTOR

For example:

```
bin/neo4j-admin database import full neo4j --nodes=import/movies.csv --nodes=import/actors.csv
--relationships=import/roles.csv --schema=import/schema.cypher
```

Import data from CSV files using regular expression

Assume that you want to include a header and then multiple files that match a pattern, e.g. containing numbers. In this case, a regular expression can be used. It is guaranteed that groups of digits will be sorted in numerical order, as opposed to lexicograghic order.

For example:

```
bin/neo4j-admin database import full --nodes import/node_header.csv,import/node_data_\d+\.csv
```

> **Note:**
>
> In many scripting languages, you must use double backslashes (`\\`) when defining regular expression patterns inside string literals. This is because a single backslash (`\`) is treated as an escape character.

Import data from CSV files using a more complex regular expression

For regular expression patterns containing commas, which is also the delimiter between files in a group, the pattern can be quoted to preserve the pattern.

For example:

```
bin/neo4j-admin database import full --nodes import/node_header.csv,'import/node_data_\d{1,5}.csv'
databasename
```

Importing files from a cloud storage

The following examples show how to import data stored in a cloud storage bucket using the `--nodes` and `--relationships` options.

> **Note:**
>
> Neo4j uses the AWS SDK v2 to call the APIs on AWS using AWS URLs. Alternatively, you can override the endpoints so that the AWS SDK can communicate with alternative storage systems, such as Ceph, Minio, or LocalStack, using the system variables `aws.endpointUrls3`, `aws.endpointUrlS3`, or `aws.endpointUrl`, or the environments variables `AWS_ENDPOINT_URL_S3` or `AWS_ENDPOINT_URL`.

1. Install the AWS CLI by following the instructions in the AWS official documentation — Install the AWS CLI version 2.
2. Create an S3 bucket and a directory to store the backup files using the AWS CLI:

   ```
   aws s3 mb --region=us-east-1 s3://myBucket
   aws s3api put-object --bucket myBucket --key myDirectory/
   ```

   For more information on how to create a bucket and use the AWS CLI, see the AWS official documentation — Use Amazon S3 with the AWS CLI and Use high-level (s3) commands with the AWS CLI.
3. Verify that the `~/.aws/config` file is correct by running the following command:

   ```
   cat ~/.aws/config
   ```

   The output should look like this:

   ```
   [default]
   region=us-east-1
   ```
4. Configure the access to your AWS S3 bucket by setting the `aws_access_key_id` and `aws_secret_access_key` in the `~/.aws/credentials` file and, if needed, using a bucket policy. For example:

a. Use `aws configure set aws_access_key_id aws_secret_access_key` command to set your IAM credentials from AWS and verify that the `~/.aws/credentials` is correct:

```
cat ~/.aws/credentials
```

The output should look like this:

```
[default]
aws_access_key_id=this.is.secret
aws_secret_access_key=this.is.super.secret
```

b. Additionally, you can use a resource-based policy to grant access permissions to your S3 bucket and the objects in it. Create a policy document with the following content and attach it to the bucket. Note that both resource entries are important to be able to download and upload files.

```
{
    "Version": "2012-10-17",
    "Id": "Neo4jBackupAggregatePolicy",
    "Statement": [
        {
            "Sid": "Neo4jBackupAggregateStatement",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetObject",
                "s3:PutObject",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::myBucket/*",
                "arn:aws:s3:::myBucket"
            ]
        }
    ]
}
```

5. Run the `neo4j-admin database import` command to import your data from your AWS S3 storage bucket. The example assumes that you have data stored in the `myBucket/data` folder in your bucket.

```
bin/neo4j-admin database import full --nodes s3://myBucket/data/nodes.csv --relationships
s3://myBucket/data/relationships.csv newdb
```

1. Ensure you have a Google account and a project created in the Google Cloud Platform (GCP).

a. Install the `gcloud` CLI by following the instructions in the Google official documentation — Install the gcloud CLI.

b. Create a service account and a service account key using Google official documentation — Create service accounts and Creating and managing service account keys.

c. Download the JSON key file for the service account.

d. Set the `GOOGLE_APPLICATION_CREDENTIALS` and `GOOGLE_CLOUD_PROJECT` environment variables to the path of the JSON key file and the project ID, respectively:

```
export GOOGLE_APPLICATION_CREDENTIALS="/path/to/keyfile.json"
export GOOGLE_CLOUD_PROJECT=YOUR_PROJECT_ID
```

e. Authenticate the `gcloud` CLI with the e-mail address of the service account you have created, the path to the JSON key file, and the project ID:

```
gcloud auth activate-service-account service-account@example.com --key-file
=$GOOGLE_APPLICATION_CREDENTIALS --project=$GOOGLE_CLOUD_PROJECT
```

For more information, see the Google official documentation — gcloud auth activate-service-account.

f. Create a bucket in the Google Cloud Storage using Google official documentation — Create buckets.

g. Verify that the bucket is created by running the following command:

```
gcloud storage ls
```

The output should list the created bucket.

2. Run the `neo4j-admin database import` command to import your data from your Google storage bucket. The example assumes that you have data stored in the `myBucket/data` folder in your bucket.

```
bin/neo4j-admin database import full --nodes gs://myBucket/data/nodes.csv --relationships
gs://myBucket/data/relationships.csv newdb
```

1. Ensure you have an Azure account, an Azure storage account, and a blob container.

   a. You can create a storage account using the Azure portal.
   For more information, see the Azure official documentation on Create a storage account.

   b. Create a blob container in the Azure portal.
   For more information, see the Azure official documentation on Quickstart: Upload, download, and list blobs with the Azure portal.

2. Install the Azure CLI by following the instructions in the Azure official documentation — Azure official documentation.

3. Authenticate the neo4j or neo4j-admin process against Azure using the default Azure credentials.
   See the Azure official documentation on default Azure credentials for more information.

```
az login
```

Then you should be ready to use Azure URLs in either neo4j or neo4j-admin.

4. To validate that you have access to the container with your login credentials, run the following commands:

```
# Upload a file:
az storage blob upload --file someLocalFile  --account-name accountName - --container
someContainer --name remoteFileName  --auth-mode login

# Download the file
az storage blob download  --account-name accountName --container someContainer --name
remoteFileName --file downloadedFile --auth-mode login

# List container files
az storage blob list  --account-name someContainer --container someContainer  --auth-mode
login
```

5. Run the `neo4j-admin database import` command to import your data from your Azure blob storage container. The example assumes that you have data stored in the `myStorageAccount/myContainer/data` folder in your container.

```
bin/neo4j-admin database import full --nodes
azb://myStorageAccount/myContainer/data/nodes.csv --relationships
azb://myStorageAccount/myContainer/data/relationships.csv newdb
```

Incremental import

Enterprise Edition

> **Note:**
>
> Incremental import supports `block` format starting from Neo4j 5.20.

Incremental import allows you to incorporate large amounts of data in batches into the graph. You can run this operation as part of the initial data load when it cannot be completed in a single full import. Besides, you can update your graph by importing data incrementally, which is more performant than transactional insertion of such data.

Incremental import requires the use of `--force` and can be run on an existing database only.

You must stop your database, if you want to perform the incremental import within one command.

If you cannot afford a full downtime of your database, split the operation into several stages:

- *prepare* stage (offline)
- *build* stage (offline or read-only)
- *merge* stage (offline)

The database must be stopped for the `prepare` and `merge` stages. During the `build` stage, the database can be left online but put into read-only mode. For a detailed example, see Incremental import in stages.

Warning:

It is highly recommended to back up your database before running the incremental import, as if the *merge* stage fails, is aborted, or crashes, it may corrupt the database.

Syntax

The syntax for importing a set of CSV files incrementally is:

```
neo4j-admin database import incremental [-h] [--expand-commands] --force [--verbose] [--auto-skip-
subsequent-headers
                                        [=true|false]] [--ignore-empty-strings[=true|false]] [--ignore-
extra-columns
                                        [=true|false]] [--legacy-style-quoting[=true|false]] [--normalize-
types
                                        [=true|false]] [--skip-bad-entries-logging[=true|false]]
                                        [--skip-bad-relationships[=true|false]] [--skip-duplicate-
nodes[=true|false]]
                                        [--strict[=true|false]] [--trim-strings[=true|false]]
                                        [--additional-config=<file>] [--array-delimiter=<char>] [--bad-
tolerance=<num>]
                                        [--delimiter=<char>] [--high-parallel-io=on|off|auto]
                                        [--id-type=string|integer|actual] [--input-encoding=<character-
set>]
                                        [--input-type=csv|parquet] [--max-off-heap-memory=<size>] [--
quote=<char>]
                                        [--read-buffer-size=<size>] [--report-file=<path>] [--
schema=<path>]
                                        [--stage=all|prepare|build|merge] [--threads=<num>]
--nodes=[<label>[:
                                        <label>]...=]<files>... [--
nodes=[<label>[:<label>]...=]<files>...]...
                                        [--relationships=[<type>=]<files>...]... [--multiline-
fields=true|false|<path>[,
                                        <path>] [--multiline-fields-format=v1|v2]] <database>
```

Description

Incremental import into an existing database.

Usage and limitations

The importer works well on standalone servers.

To safely perform an incremental import in a clustered environment, follow these steps:

1. Run the incremental import command on a single server in the cluster. This server can then be used as the designated seeder from which other cluster members can copy the database.
2. Reconfigure the database topology to a single primary by running the `dbms.cluster.recreateDatabase()` procedure.
3. Then stop the database using STOP DATABASE.
4. Perform the incremental import on the server that hosts the database.
5. Then start the database with START DATABASE.
6. Lastly, restore the desired database topology using <<, ALTER DATABASE>>.

The incremental import command can be used to add:

- New nodes with labels and properties.

> **Warning:**
>
> Note that you must have node property uniqueness constraints in place for the property key and label combinations that form the primary key, or the uniquely identifiable nodes. Otherwise, the command will throw an error and exit. For more information, see CSV header format.

- New relationships between existing or new nodes.

The incremental import command cannot be used to:

- Add new properties to existing nodes or relationships.
- Update or delete properties in nodes or relationships.
- Update or delete labels in nodes.
- Delete existing nodes and relationships.

Parameters

Table 638. `neo4j-admin database import incremental` *parameters*

Parameter	Description	Default
`<database>`	Name of the database to import. If the database into which you import does not exist prior to importing, you must create it subsequently using `CREATE DATABASE`.	`neo4j`

Options

Table 639. `neo4j-admin database import incremental` *options*

Option	Description	Default
`--additional-config=<file>`[8]	Configuration file with additional configuration.	

Option	Description	Default
`--array-delimiter=<char>`	Delimiter character between array elements within a value in CSV data. Also accepts `TAB` and e.g. `U+20AC` for specifying a character using Unicode. • ASCII character — e.g. `--array-delimiter=";"`. • `\ID` — Unicode character with ID, e.g. `--array-delimiter="\59"`. • `U+XXXX` — Unicode character specified with 4 HEX characters, e.g. `--array-delimiter="U+20AC"`. • `\t` — horizontal tabulation (HT), e.g. `--array-delimiter="\t"`. For horizontal tabulation (HT), use `\t` or the Unicode character ID `\9`. Unicode character ID can be used if prepended by `\`.	`;`
`--auto-skip-subsequent-headers[=true|false]`[9]	Automatically skip accidental header lines in subsequent files in file groups with more than one file.	`false`
`--bad-tolerance=<num>`	Number of bad entries before the import is aborted. The import process is optimized for error-free data. Therefore, cleaning the data before importing it is highly recommended. If you encounter any bad entries during the import process, you can set the number of bad entries to a specific value that suits your needs. However, setting a high value may affect the performance of the tool.	`1000`
`--delimiter=<char>`[9]	Delimiter character between values in CSV data. Also accepts `TAB` and e.g. `U+20AC` for specifying a character using Unicode. • ASCII character — e.g. `--delimiter=","`. • `\ID` — Unicode character with ID, e.g. `--delimiter="\44"`. • `U+XXXX` — Unicode character specified with 4 HEX characters, e.g. `--delimiter="U+20AC"`. • `\t` — horizontal tabulation (HT), e.g. `--delimiter="\t"`. For horizontal tabulation (HT), use `\t` or the Unicode character ID `\9`. Unicode character ID can be used if prepended by `\`.	`,`
`--expand-commands`	Allow command expansion in config value evaluation.	
`--force`	Confirm incremental import by setting this flag.	
`-h, --help`	Show this help message and exit.	
`--high-parallel-io=on|off|auto`	Ignore environment-based heuristics and indicate if the target storage subsystem can support parallel IO with high throughput or auto detect. Typically this is `on` for SSDs, large raid arrays, and network-attached storage.	`auto`

Option	Description	Default
`--id-type=string|integer|actual`	Introduced in 5.1 Each node must provide a unique ID. This is used to find the correct nodes when creating relationships. Possible values are: • `string` — arbitrary strings for identifying nodes. • `integer` — arbitrary integer values for identifying nodes. • `actual` — (advanced) actual node IDs.	`string`
`--ignore-empty-strings[=true|false]`	Whether or not empty string fields, i.e. "" from input source are ignored, i.e. treated as null.	`false`
`--ignore-extra-columns[=true|false]`[9]	If unspecified columns should be ignored during the import.	`false`
`--input-encoding=<character-set>`[9]	Character set that input data is encoded in.	`UTF-8`
`--input-type=csv|parquet`	[.label-success]# Introduced in 5.26 #File type to import from. Can be csv or parquet. Defaults to csv.	`csv`
`--legacy-style-quoting[=true|false]`	Whether or not a backslash-escaped quote e.g. \" is interpreted as an inner quote.	`false`
`--max-off-heap-memory=<size>`	Maximum memory that `neo4j-admin` can use for various data structures and caching to improve performance. Values can be plain numbers, such as `10000000`, or `20G` for 20 gigabytes. It can also be specified as a percentage of the available memory, for example `70%`.	`90%`
`--multiline-fields=true|false|<path>[,<path>]`[9]	Changed in 5.26 In v1, whether or not fields from an input source can span multiple lines, i.e. contain newline characters. Setting `--multiline-fields=true` can severely degrade the performance of the importer. Therefore, use it with care, especially with large imports. In v2, this option will specify the list of files that contain multiline fields. Files can also be specified using regular expressions.	`null`
`--multiline-fields-format=v1|v2`[9]	Introduced in 5.26 Controls the parsing of input source that can span multiple lines, i.e. contain newline characters. When set to v1, the value for `--multiline-fields` can only be true or false. When set to v2, the value for `--multiline-fields` should be the list of files that contain multiline fields.	`null`

Option	Description	Default
`--nodes=[<label>[:<label>]…=]<files>…`	Node CSV header and data. • Multiple files will be logically seen as one big file from the perspective of the importer. • The first line must contain the header. • Multiple data sources like these can be specified in one import, where each data source has its own header. • Files can also be specified using regular expressions. It is possible to import files from AWS S3 buckets, Google Cloud storage buckets, and Azure buckets using the appropriate URI as the path. For an example, see Import data from CSV files using regular expression.	
`--normalize-types[=true|false]`	When `true`, non-array property values are converted to their equivalent Cypher types. For example, all integer values will be converted to 64-bit long integers.	`true`
`--quote=<char>`[9] [10]	Character to treat as a quotation mark for values in CSV data. For example, quotes can be escaped as per RFC 4180 by doubling them. Thus `""` would be interpreted as a literal `"`. You cannot escape using `\`.	`"`
`--read-buffer-size=<size>`	Size of each buffer for reading input data. It has to be at least large enough to hold the biggest single value in the input data. The value can be a plain number or a byte units string, e.g. `128k`, `1m`.	`4194304`
`--relationships=[<type>=]<files>…`	Relationship CSV header and data. • Multiple files will be logically seen as one big file from the perspective of the importer. • The first line must contain the header. • Multiple data sources like these can be specified in one import, where each data source has its own header. • Files can also be specified using regular expressions. It is possible to import files from AWS S3 buckets, Google Cloud storage buckets, and Azure buckets using the appropriate URI as the path. For an example, see Import data from CSV files using regular expression.	

Option	Description	Default
`--report-file=<path>`	File in which to store the report of the csv-import. The location of the import log file can be controlled using the `--report-file` option. If you run large imports of CSV files that have low data quality, the import log file can grow very large. For example, CSV files that contain duplicate node IDs, or that attempt to create relationships between non-existent nodes, could be classed as having low data quality. In these cases, you may wish to direct the output to a location that can handle the large log file. If you are running on a UNIX-like system and you are not interested in the output, you can get rid of it altogether by directing the report file to `/dev/null`. If you need to debug the import, it might be useful to collect the stack trace. This is done by using the `--verbose` option.	`import.report`
`--schema=<path>`[11]	Introduced in 5.24 Path to the file containing the Cypher commands for creating indexes and constraints during data import.	
`--skip-bad-entries-logging[=true|false]`	When set to `true`, the details of bad entries are not written in the log. Disabling logging can improve performance when the data contains lots of faults. Cleaning the data before importing it is highly recommended because faults dramatically affect the tool's performance even without logging.	`false`
`--skip-bad-relationships[=true|false]`	Whether or not to skip importing relationships that refer to missing node IDs, i.e. either start or end node ID/group referring to a node that was not specified by the node input data. Skipped relationships will be logged, containing at most the number of entities specified by `--bad-tolerance`, unless otherwise specified by the `--skip-bad-entries-logging` option.	`false`
`--skip-duplicate-nodes[=true|false]`	Whether or not to skip importing nodes that have the same ID/group. In the event of multiple nodes within the same group having the same ID, the first encountered will be imported, whereas consecutive such nodes will be skipped. Skipped nodes will be logged, containing at most the number of entities specified by `--bad-tolerance`, unless otherwise specified by the `--skip-bad-entries-logging` option.	`false`
`--stage=all|prepare|build|merge`	Stage of incremental import. For incremental import into an existing database use `all` (which requires the database to be stopped). For semi-online incremental import run `prepare` (on a stopped database) followed by `build` (on a potentially running database) and finally `merge` (on a stopped database).	`all`

Option	Description	Default
`--strict[=true|false]`	Introduced in 5.6 Whether or not the lookup of nodes referred to from relationships needs to be checked strict. If disabled, most but not all relationships referring to non-existent nodes will be detected. If enabled all those relationships will be found but at the cost of lower performance.	`false` Changed in 5.8
`--threads=<num>`	(advanced) Max number of worker threads used by the importer. Defaults to the number of available processors reported by the JVM. There is a certain amount of minimum threads needed so for that reason there is no lower bound for this value. For optimal performance, this value should not be greater than the number of available processors.	`20`
`--trim-strings[=true|false]`[9]	Whether or not strings should be trimmed for whitespaces.	`false`
`--verbose`	Enable verbose output.	

Note:

Using both a multi-value option and a positional parameter

When using both a multi-value option, such as `--nodes` and `--relationships`, and a positional parameter (for example, in `--additional-config neo4j.properties --nodes 0-nodes.csv mydatabase`), the `--nodes` option acts "greedy" and the next option, in this case, `mydatabase`, is pulled in via the nodes convertor.

This is a limitation of the underlying library, Picocli, and is not specific to Neo4j Admin. For more information, see Picocli → Variable Arity Options and Positional Parameters official documentation.

To resolve the problem, use one of the following solutions:

- Put the positional parameters first. For example, `mydatabase --nodes 0-nodes.csv`.
- Put the positional parameters last, after `--` and the final value of the last multi-value option. For example, `nodes 0-nodes.csv – mydatabase`.

Examples

There are two ways of importing data incrementally.

Incremental import in a single command

If downtime is not a concern, you can run a single command with the option `--stage=all`. This option requires the database to be stopped.

```
neo4j@system> STOP DATABASE db1 WAIT;
...
bin/neo4j-admin database import incremental --stage=all --nodes=N1=../../raw-data/incremental-
import/b.csv db1
```

Incremental import in stages

If you cannot afford a full downtime of your database, you can run the import in three stages.

1. `prepare` stage:

   During this stage, the import tool analyzes the CSV headers and copies the relevant data over to the new increment database path. The import command is run with the option `--stage=prepare` and the database must be stopped.

   a. Using the `system` database, stop the database `db1` with the `WAIT` option to ensure a checkpoint happens before you run the incremental import command. The database must be stopped to run `--stage=prepare`.

   ```
   STOP DATABASE db1 WAIT
   ```

   b. Run the incremental import command with the `--stage=prepare` option:

   ```
   bin/neo4j-admin database import incremental --stage=prepare --nodes=N1=../../raw-
   data/incremental-import/c.csv db1
   ```

2. `build` stage:

   During this stage, the import tool imports the data, deduplicates it, and validates it in the new increment database path. This is the longest stage and you can put the database in read-only mode to allow read access. The import command is run with the option `--stage=build`.

   a. Put the database in read-only mode:

   ```
   ALTER DATABASE db1 SET ACCESS READ ONLY
   ```

   b. Run the incremental import command with the `--stage=build` option:

   ```
   bin/neo4j-admin database import incremental --stage=build --nodes=N1=../../raw-
   data/incremental-import/c.csv db1
   ```

3. `merge` stage:

   During this stage, the import tool merges the new with the existing data in the database. It also updates the affected indexes and upholds the affected property uniqueness constraints and property existence constraints. The import command is run with the option `--stage=merge` and the database must be stopped. It is not necessary to include the `--nodes` or `--relationships` options when using `--stage=merge`.

   a. Using the `system` database, stop the database `db1` with the `WAIT` option to ensure a checkpoint happens before you run the incremental import command.

   ```
   STOP DATABASE db1 WAIT
   ```

b. Run the incremental import command with the `--stage=merge` option:

```
bin/neo4j-admin database import incremental --stage=merge db1
```

CSV header format

The header file of each data source specifies how the data fields should be interpreted. You must use the same delimiter for the header file and the data files.

The header contains information for each field, with the format `<name>:<field_type>`. The `<name>` is used for properties and node IDs. In all other cases, the `<name>` part of the field is ignored.

Incremental import

Note:

When using incremental import, you must have node property uniqueness constraints in place for the property key and label combinations that form the primary key, or the uniquely identifiable nodes. For example, importing nodes with a `Person` label that are uniquely identified with a `uuid` property key, the format of the header should be `uuid:ID{label:Person}`.

This is also true when working with multiple groups. For example, you can use `uuid:ID(Person){label:Person}`, where the relationship CSV data can refer to different groups for its `:START_ID` and `:END_ID`, just like the full import method.

- For more information on constraints, see Cypher Manual → Constraints.
- For examples of creating property uniqueness constraints, see Cypher Manual → Create property uniqueness constraints.

Node files

Files containing node data can have an `ID` field, a `LABEL` field, and properties.

ID

Each node must have a unique ID if it is to be connected by any relationships created in the import. Neo4j uses the IDs to find the correct nodes when creating relationships. Note that the ID has to be unique across all nodes within the group, regardless of their labels. The unique ID is persisted in a property whose name is defined by the `<name>` part of the field definition `<name>:ID`. If no such property `name` is defined, the unique ID will be used for the import but not be available for reference later. If no ID is specified, the node will be imported, but it will not be connected to other nodes during the import. When a property `name` is provided, that property type can be configured globally via the `--id-type` option (as for Property data types).
From Neo4j 5.1, you can specify a different value ID type to be stored for a node property in its group using the option `id-type` in the header, e.g: `id:ID(MyGroup){label:MyLabel, id-type: int}`. This ID type overrides the global `--id-type` option. For example, the global `id-type` can be a string, but the nodes will have their IDs stored as `int` type in their ID properties. For more information, see Storing a

different value type for IDs in a group.
From Neo4j 5.3, a node header can also contain multiple `ID` columns, where the relationship data references the composite value of all those columns. This also implies using `string` as `id-type`. For each `ID` column, you can specify to store its values as different node properties. However, the composite value cannot be stored as a node property. For more information, see Using multiple node IDs.

LABEL

Read one or more labels from this field. Like array values, multiple labels are separated by `;`, or by the character specified with `--array-delimiter`. Introduced in 5.25 The max length of label names for block format is 16,383 characters.

Example 158. Define node files

You define the headers for movies in the *movies_header.csv* file. Movies have the properties `movieId`, `year`, and `title`. You also specify a field for labels.

```
movieId:ID,title,year:int,:LABEL
```

You define three movies in the *movies.csv* file. They contain all the properties defined in the header file. All the movies are given the label `Movie`. Two of them are also given the label `Sequel`.

```
tt0133093,"The Matrix",1999,Movie
tt0234215,"The Matrix Reloaded",2003,Movie;Sequel
tt0242653,"The Matrix Revolutions",2003,Movie;Sequel
```

Similarly, you also define three actors in the *actors_header.csv* and *actors.csv* files. They all have the properties `personId` and `name`, and the label `Actor`.

```
personId:ID,name,:LABEL
```

```
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
```

Relationship files

Files containing relationship data have three mandatory fields and can also have properties. The mandatory fields are:

TYPE

The relationship type to use for this relationship. Introduced in 5.25 The max length of relationship type names for block format is 16,383 characters.

START_ID

The ID of the start node for this relationship.

END_ID

The ID of the end node for this relationship.

The `START_ID` and `END_ID` refer to the unique node ID defined in one of the node data sources, as explained in the previous section. None of these take a name, e.g. if `<name>:START_ID` or `<name>:END_ID` is defined, the `<name>` part will be ignored. Nor do they take a `<field_type>`, e.g. if `:START_ID:int` or `:END_ID:int` is defined, the `:int` part does not have any meaning in the context of type information.

Example 159. Define relationships files

In this example, you assume that the two node files from the previous example are used together with the following relationships file.

You define relationships between actors and movies in the files *roles_header.csv* and *roles.csv*. Each row connects a start node and an end node with a relationship of relationship type `ACTED_IN`. Notice how you use the unique identifiers `personId` and `movieId` from the nodes files above. The name of the character that the actor is playing in this movie is stored as a `role` property on the relationship.

```
:START_ID,role,:END_ID,:TYPE
```

```
keanu,"Neo",tt0133093,ACTED_IN
keanu,"Neo",tt0234215,ACTED_IN
keanu,"Neo",tt0242653,ACTED_IN
laurence,"Morpheus",tt0133093,ACTED_IN
laurence,"Morpheus",tt0234215,ACTED_IN
laurence,"Morpheus",tt0242653,ACTED_IN
carrieanne,"Trinity",tt0133093,ACTED_IN
carrieanne,"Trinity",tt0234215,ACTED_IN
carrieanne,"Trinity",tt0242653,ACTED_IN
```

Property data types

For properties, the `<name>` part of the field designates the property key, while the `<field_type>` part assigns a data type. You can have properties in both node data files and relationship data files. Introduced in 5.25 The max length of property keys for block format is 16,383 characters.

Use one of `int`, `long`, `float`, `double`, `boolean`, `byte`, `short`, `char`, `string`, `point`, `date`, `localtime`, `time`, `localdatetime`, `datetime`, and `duration` to designate the data type for properties. By default, types (except arrays) are converted to Cypher types. See Cypher Manual → Property, structural, and constructed values.

This behavior can be disabled using the option `--normalize-types=false`. Normalizing types can require more space on disk, but avoids Cypher converting the type during queries. If no data type is given, this defaults to `string`.

To define an array type, append `[]` to the type. By default, array values are separated by `;`. A different delimiter can be specified with `--array-delimiter`. Arrays are not affected by the `--normalize-types` flag. For example, if you want a byte array to be stored as a Cypher long array, you must explicitly declare the property as `long[]`.

Note:

CSV-based import does not import empty arrays, because they cannot be distinguished from arrays that are set to `null`. However, the Parquet import distinguishes between them and will import empty arrays as empty arrays and `null` as `null`.

Boolean values are *true* if they match exactly the text `true`. All other values are *false*. Values that contain the delimiter character need to be escaped by enclosing in double quotation marks, or by using a different delimiter character with the `--delimiter` option.

Example 160. Header format with data types

This example illustrates several different data types specified in the CSV header.

```
:ID,name,joined:date,active:boolean,points:int
user01,Joe Soap,2017-05-05,true,10
user02,Jane Doe,2017-08-21,true,15
user03,Moe Know,2018-02-17,false,7
```

Special considerations for the `point` data type

A point is specified using the Cypher syntax for maps. The map allows the same keys as the input to the Cypher Manual → Point function. The point data type in the header can be amended with a map of default values used for all values of that column, e.g. `point{crs: 'WGS-84'}`. Specifying the header this way allows you to have an incomplete map in the value position in the data file. Optionally, a value in a data file may override default values from the header.

Example 161. Property format for `point` data type

This example illustrates various ways of using the `point` data type in the import header and the data files.

You are going to import the name and location coordinates for cities. First, you define the header as:

```
:ID,name,location:point{crs:WGS-84}
```

You then define cities in the data file.

- The first city's location is defined using `latitude` and `longitude`, as expected when using the coordinate system defined in the header.
- The second city uses `x` and `y` instead. This would normally lead to a point using the coordinate reference system `cartesian`. Since the header defines `crs:WGS-84`, that coordinate reference system will be used.
- The third city overrides the coordinate reference system defined in the header and sets it explicitly to `WGS-84-3D`.

```
:ID,name,location:point{crs:WGS-84}
city01,"Malmö","{latitude:55.6121514, longitude:12.9950357}"
city02,"London","{y:51.507222, x:-0.1275}"
city03,"San Mateo","{latitude:37.554167, longitude:-122.313056, height: 100, crs:'WGS-84-3D'}"
```

Note that all point maps are within double quotation marks `"` in order to prevent the enclosed `,` character from being interpreted as a column separator. An alternative approach would be to use `--delimiter='\t'` and reformat the file with tab separators, in which case the `"` characters are not required.

```
:ID name    location:point{crs:WGS-84}
city01  Malmö   {latitude:55.6121514, longitude:12.9950357}
city02  London  {y:51.507222, x:-0.1275}
city03  San Mateo   {latitude:37.554167, longitude:-122.313056, height: 100, crs:'WGS-84-3D'}
```

Special considerations for temporal data types

The format for all temporal data types must be defined as described in Cypher Manual → Durations syntax. Two of the temporal types, *Time* and *DateTime*, take a time zone parameter that might be common between all or many of the values in the data file. It is therefore possible to specify a default time zone for *Time* and *DateTime* values in the header, for example: `time{timezone:+02:00}` and: `datetime{timezone:Europe/Stockholm}`. If no default time zone is specified, the default timezone is determined by the `db.temporal.timezone` configuration setting. The default time zone can be explicitly overridden in the values in the data file.

Example 162. Property format for temporal data types

This example illustrates various ways of using the `datetime` data type in the import header and the data files.

First, you define the header with two *DateTime* columns. The first one defines a time zone, but the second one does not:

```
:ID,date1:datetime{timezone:Europe/Stockholm},date2:datetime
```

You then define dates in the data file.

- The first row has two values that do not specify an explicit timezone. The value for `date1` will use the `Europe/Stockholm` time zone that was specified for that field in the header. The value for `date2` will use the configured default time zone of the database.
- In the second row, both `date1` and `date2` set the time zone explicitly to be `Europe/Berlin`. This overrides the header definition for `date1`, as well as the configured default time zone of the database.

```
1,2018-05-10T10:30,2018-05-10T12:30
2,2018-05-10T10:30[Europe/Berlin],2018-05-10T12:30[Europe/Berlin]
```

Using ID spaces

By default, the import tool assumes that node identifiers are unique across node files. In many cases, the ID is unique only across each entity file, for example, when your CSV files contain data extracted from a relational database and the ID field is pulled from the primary key column in the corresponding table. To handle this situation you define *ID spaces*. ID spaces are defined in the `ID` field of node files using the

syntax `ID(<ID space identifier>)`. To reference an ID of an ID space in a relationship file, you use the syntax `START_ID(<ID space identifier>)` and `END_ID(<ID space identifier>)`.

Example 163. Define and use ID spaces

Define a `Movie-ID` ID space in the *movies_header.csv* file.

```
movieId:ID(Movie-ID),title,year:int,:LABEL
```

```
1,"The Matrix",1999,Movie
2,"The Matrix Reloaded",2003,Movie;Sequel
3,"The Matrix Revolutions",2003,Movie;Sequel
```

Define an `Actor-ID` ID space in the header of the *actors_header.csv* file.

```
personId:ID(Actor-ID),name,:LABEL
```

```
1,"Keanu Reeves",Actor
2,"Laurence Fishburne",Actor
3,"Carrie-Anne Moss",Actor
```

Now use the previously defined ID spaces when connecting the actors to movies.

```
:START_ID(Actor-ID),role,:END_ID(Movie-ID),:TYPE
```

```
1,"Neo",1,ACTED_IN
1,"Neo",2,ACTED_IN
1,"Neo",3,ACTED_IN
2,"Morpheus",1,ACTED_IN
2,"Morpheus",2,ACTED_IN
2,"Morpheus",3,ACTED_IN
3,"Trinity",1,ACTED_IN
3,"Trinity",2,ACTED_IN
3,"Trinity",3,ACTED_IN
```

Using multiple node IDs

From Neo4j 5.3, a node header can also contain multiple `ID` columns, where the relationship data references the composite value of all those columns. This also implies using `string` as `id-type`.

For each `ID` column, you can specify to store its values as different node properties. However, the composite value cannot be stored as a node property.

Note:

Incremental import doesn't support the use of multiple node identifiers. This functionality is only available with a full import.

Example 164. Define multiple IDs as node properties

You can define multiple `ID` columns in the node header. For example, you can define a node header with two `ID` columns.

nodes_header.csv

```
:ID,:ID,name
```

nodes.csv

```
aa,11,John
bb,22,Paul
```

Now use both IDs when defining the relationship:

relationships_header.csv

```
:START_ID,:TYPE,:END_ID
```

relationships.csv

```
aa11,WORKS_WITH,bb22
```

Example 165. Define multiple IDs stored in ID spaces

Define a `MyGroup` ID space in the *nodes_header.csv* file.

nodes_header.csv

```
personId:ID(MyGroup),memberId:ID(MyGroup),name
```

nodes.csv

```
aa,11,John
bb,22,Paul
```

Now use the defined ID space when connecting John with Paul, and use both IDs in the relationship.

relationships_header.csv

```
:START_ID(MyGroup),:TYPE,:END_ID(MyGroup)
```

relationships.csv

```
aa11,WORKS_WITH,bb22
```

Storing a different value type for IDs in a group

From Neo4j 5.1, you can control the ID type of the node property that will be stored by defining the `id-`

`type` option in the header, for example, `:ID{id-type:long}`. The `id-type` option in the header overrides the global `--id-type` value provided to the command. This way, you can have property values of different types for different groups of nodes. For example, the global `id-type` can be a string, but some nodes can have their IDs stored as `long` type in their ID properties.

Example 166. Import nodes with different ID value types

persons_header.csv

```
id:ID(GroupOne){id-type:long},name,:LABEL
```

persons.csv

```
123,P1,Person
456,P2,Person
```

games_header.csv

```
id:ID(GroupTwo),name,:LABEL
```

games.csv

```
ABC,G1,Game
DEF,G2,Game
```

Import the nodes

```
neo4j_home$ --nodes persons.csv --nodes games.csv --id-type string
```

The `id` property of the nodes in the `persons` group will be stored as `long` type, while the `id` property of the nodes in the `games` group will be stored as `string` type, as the global `id-type` is a string.

Importing data that spans multiple lines

The `--multiline-fields` option allows fields from an input source to span multiple lines, i.e. contain newline characters. For example:

```
bin/neo4j-admin database import full --nodes import/node_header.csv,import/node_data.csv --multiline
-fields=true databasename
```

Where `import/node_data.csv` contains multiline fields, such as:

```
id,name,birthDate,birthYear,birthLocation,description
1,John,October 1st,2000,New York,This is a multiline
description
```

Note:

Setting `--multiline-fields=true` can severely degrade the performance of the importer.

Therefore, use it with care, especially with large imports.

Starting from 5.26, you can optionally specify the format of the `--multiline-fields` to control the parsing of the input source by setting the `--multiline-fields-format` option. Possible values are:

- `v1` - the default format, which uses the current processing method for multiline fields.
- `v2` - a more efficient processing method that requires text fields to be quoted. For `v2`, the `--multiline-fields` option must be set to a list of files (regular expressions are allowed) that contain multiline fields.

Both formats have the restriction that the entirety of every row must be able to fit into the buffer (default is 4m). The `--multiline-fields-format` option is available in the `full` and `incremental` import modes.

For example:

```
bin/neo4j-admin database import full --nodes import/node_header.csv,import/node_data.csv
--multiline-fields=true --multiline-fields-format=v1 databasename
```

Where `import/node_data.csv` contains multiline fields, such as:

```
id,name,birthDate,birthYear,birthLocation,description
1,John,October 1st,2000,New York,This is a multiline
description
```

```
bin/neo4j-admin database import full --nodes import/node_header.csv,import/node_data.csv
--multiline-fields=import/node_data.csv --multiline-fields-format=v2 databasename
```

Where `import/node_data.csv` contains multiline fields, such as:

```
id,name,birthDate,birthYear,birthLocation,description
1,"John","October 1st",2000,"New York","This is a multiline
description"
```

Skipping columns

IGNORE

If there are fields in the data that you wish to ignore completely, this can be done using the `IGNORE` keyword in the header file. `IGNORE` must be prepended with a `:`.

Example 167. Skip a column

In this example, you are not interested in the data in the third column of the nodes file and wish to skip over it. Note that the `IGNORE` keyword is prepended by a `:`.

```
personId:ID,name,:IGNORE,:LABEL
```

```
keanu,"Keanu Reeves","male",Actor
laurence,"Laurence Fishburne","male",Actor
carrieanne,"Carrie-Anne Moss","female",Actor
```

If all your superfluous data is placed in columns located to the right of all the columns that you wish to import, you can instead use the command line option `--ignore-extra-columns`.

Importing compressed files

The import tool can handle files compressed with `zip` or `gzip`. Each compressed file must contain a single file.

Example 168. Perform an import using compressed files

```
neo4j_home$ ls import
actors-header.csv  actors.csv.zip  movies-header.csv  movies.csv.gz  roles-header.csv  roles.csv.gz
```

```
bin/neo4j-admin database import --nodes import/movies-header.csv,import/movies.csv.gz --nodes
import/actors-header.csv,import/actors.csv.zip --relationships import/roles-
header.csv,import/roles.csv.gz
```

Resuming a stopped or canceled import

Enterprise Edition

An import that is stopped or fails before completing can be resumed from a point closer to where it was stopped. An import can be resumed from the following points:

- Linking of relationships
- Post-processing

Unbind a Neo4j cluster server

You can use the `neo4j-admin server unbind` command to remove and archive the cluster state of a cluster server so that it can rebind to a cluster.

Warning:

Due to the new clustering architecture in Neo4j 5.x, the behavior of the `unbind` command has changed significantly compared to Neo4j 4.4.

Running the `unbind` command on any number of servers may result in the loss of cluster quorum, severely affecting clustering functionality. Therefore, it is strongly recommended to consult with Neo4j Support before using the `unbind` command.

In Neo4j 5.x, use the `unbind` command only when troubleshooting **a specific server** and remember there is no guarantee that the allocator will reassign the same databases to this server, potentially resulting in orphaned database stores.

The `unbind` command preserves all database stores on the server; and when the unbound server is restarted, it is seen as an entirely new server. Therefore, it will not host any of the databases it hosted before the operation.

In Neo4j 5.x, the `unbind` command cannot be used to convert a cluster member into a standalone server. Instead, it is recommended to take backups, create the standalone server, and then use those backups to restore the databases.

Syntax

The `neo4j-admin server unbind` command has the following syntax:

```
neo4j-admin server unbind [-h] [--expand-commands] [--verbose]
                          [--archive-cluster-state[=true|false]]
                          [--additional-config=<file>]
                          [--archive-path=<path>]
```

Options

The `neo4j-admin server unbind` command has the following options:

Table 640. `neo4j-admin server unbind` *options*

Option	Description	Default
`--additional-config=<file>`[12]	Configuration file with additional configuration.	
`--archive-cluster-state[=true|false]`	Enable or disable the cluster state archiving.	`false`
`--archive-path=<path>`	Destination (file or folder) of the cluster state archive.	
`--expand-commands`	Allow command expansion in config value evaluation.	
`-h, --help`	Show this help message and exit.	
`--verbose`	Enable verbose output.	

Limitations

The Neo4j server must be shut down before running the `neo4j-admin server unbind` command.

Usage

You can use the `neo4j-admin server unbind` command to remove or archive the cluster state of a cluster server.

Remove the cluster state of a server

To remove the cluster state of a server, run the `neo4j-admin server unbind` command from the <*NEO4J_HOME*> folder of that server. When restarted, an unbound server rejoins the cluster as a new server and has to be enabled using the `ENABLE SERVER` command.

Archive cluster state

If something goes wrong and debugging is needed, you can archive the cluster state, from the <*NEO4J_HOME*> folder, run the `neo4j-admin server unbind` command with the arguments `--archive-cluster-state=true` and `--archive-path=<destination-folder>`:

```
bin/neo4j-admin server unbind --archive-path=/path/to/archive-folder --archive-cluster-state=true
```

The default resultant file is named:

```
unbound_cluster_state.<YYYYMMDDHH24MM>.zip
```

Upload to Neo4j Aura

Available on Aura

The `neo4j-admin database upload` command uploads a local Neo4j database dump or backup file into a Neo4j Aura instance. The option of using a `.backup` file with the `neo4j-admin database upload` command was introduced in 5.22. The following table shows the compatibility between the dump/backup version that you want to upload and the version of the Neo4j Aura instance.

Dump/Backup version	Aura version
5.x	5.latest
4.4	4 and 5.latest

> **Note:**
>
> This operation is secured and TLS encrypted end to end.

Prerequisites

Before you can use the `neo4j-admin database upload` command, you must meet the following prerequisites:

- Verify that your Neo4j Aura instance is running.
- Verify that your Neo4j Aura instance is accessible from the machine running `neo4j-admin`. Otherwise, the upload will fail with SSL errors.

Syntax

The `neo4j-admin database upload` command has the following syntax:

```
neo4j-admin database upload [-h] [--expand-commands] [--verbose]
                            [--overwrite-destination[=true|false]]
                            [--additional-config=<file>] --from-path=<path> [--to=<destination>]
                            [--to-password=<password>] --to-uri=<uri> [--to-user=<username>] <database>
```

Description

Push a local database to a Neo4j Aura instance. The target location is a Neo4j Aura Bolt URI. If Neo4j Cloud username and password are not provided either as a command option or as an environment variable, they will be requested interactively.

Parameters

Table 641. `neo4j-admin database upload` parameters

Parameter	Description
`<database>`	Name of the database that should be uploaded. The name is used to select a file which is expected to be named <database>.dump or <database>.backup.

Options

The `neo4j-admin database upload` command has the following options:

Table 642. `neo4j-admin database upload` options

Option	Description	Default
`--additional-config=<file>`[13]	Configuration file with additional configuration.	
`--expand-commands`	Allow command expansion in config value evaluation.	
`--from-path=<path>`	*/path/to/directory-containing-dump-or-backup* Path to a directory containing a database dump or backup file to upload.	
`-h, --help`	Show this help message and exit.	
`--overwrite-destination[=true|false]`	Overwrite the data in the target database.	`false`
`--to=<destination>`	The destination for the upload.	`aura`
`--to-password=<password>`	Password of the target database to push this database to. Prompt will ask for a password if not provided.	`The value of the NEO4J_PASSWORD environment variable.`
`--to-uri=<uri>`	`neo4j://mydatabaseid.databases.neo4j.io` Bolt URI of the target database.	
`--to-user=<username>`	Username of the target database to push this database to. Prompt will ask for a username if not provided.	`The value of the NEO4J_USERNAME environment variable.`
`--verbose`	Enable verbose output.	

Output

If the `upload` function completes successfully, it exits with the following log line:

```
“Your data was successfully pushed to Aura and is now running”.
```

If the `upload` function encounters an error at any point, you will be provided with instructions on how to try again or to contact Neo4j Aura support.

Additionally, you can use the `--verbose` option to enable verbose output.

Example

The following examples show how to use the `neo4j-admin database upload` command to upload a database dump to a Neo4j Aura instance. You need your Aura instance URI (`neo4j+s://your-aura-instance-id.databases.neo4j.io`), as can be seen in the Aura console, and your Aura instance password.

Note:

You should use the `--overwrite-destination=true` option to overwrite the target database. Otherwise, the command aborts and throws an error.

Caution:

This command does not currently support private linking. Please raise a support ticket if you have public traffic disabled and need to use this command.

```
bin/neo4j-admin database upload <database> --from-path=<path-to-directory-with-database-dump> --to-uri
=<neo4j+s://your-aura-instance-id.databases.neo4j.io> --overwrite-destination=true
Neo4j cloud database user name: neo4j
Neo4j cloud database password:
Upload
..................  10%
..................  20%
..................  30%
..................  40%
..................  50%
..................  60%
..................  70%
..................  80%
..................  90%
.................. 100%
We have received your export and it is currently being loaded into your Aura instance.
You can wait here, or abort this command and head over to the console to be notified of when your database
is running.
Import progress
..................  10%
..................  20%
..................  30%
..................  40%
..................  50%
..................  60%
..................  70%
..................  80%
..................  90%
.................. 100%
Your data was successfully pushed to Aura and is now running.
```

On Windows, the backslashes `\` in the file paths must be escaped with another backslash. For example:

```
bin\neo4j-admin database upload dbname --from-path=c:\\db-dump-file\\ --to-uri=<neo4j+s://your-aura-
instance-id.databases.neo4j.io> --overwrite-destination=true
```

Migrate a database

Not available on Aura

You can use the `neo4j-admin database migrate` command to migrate a Neo4j database from one store format to another or to a later `MAJOR` version of the same format.

A store format defines how the data of a database is stored on the file system.

As of Neo4j 5, the store format of a database is versioned with the `MAJOR.MINOR` scheme, independent of Neo4j versioning. An upgrade to the latest `MINOR` format version is an automatic operation performed on database startup. A migration to a higher `MAJOR` format version or another format is a manual action performed with the `migrate` command.

The store format for new databases can be set with the `db.format` configuration setting.

The table below shows supported migration paths between different Neo4j store formats. However, the `standard` and `high_limit` store formats are deprecated starting with Neo4j 5.23, meaning that migration to these formats is not recommended from this version onward.

Table 643. Migration paths between Neo4j store formats

Store format	Migration paths			
	To `block`	To `aligned`	To `high_limit`	To `standard`
`block` [1 - Installation]		✔ Not recommended	✔ Not recommended	
`aligned` [2 - Installation]	✔		✔ Not recommended	
`high_limit` [14]	✔			
`standard` [15]	✔	✔	✔ Not recommended	

Before migrating from `block` to the `aligned` format, you have to ensure that your graph fits in the `aligned` since the `block` format has higher limits. For details, see Database internals and transactional behavior → Store formats.

Syntax

The `neo4j-admin database migrate` has the following syntax:

```
neo4j-admin database migrate [-h] [--expand-commands]
                             [--force-btree-indexes-to-range]
                             [--verbose]
                             [--additional-config=<file>] [--pagecache=<size>]
                             [--to-format=standard|high_limit|aligned|block] <database>
```

> **Note:**
>
> The `neo4j-admin database migrate` command is run only on a stopped database.

Parameters

Table 644. `neo4j-admin database migrate` parameters

Parameter	Description
`<database>`	Name of the database to migrate. Can contain * and ? for globbing. Note that * and ? have special meaning in some shells and might need to be escaped or used with quotes.

Options

The `neo4j-admin database migrate` command has the following options:

Table 645. `neo4j-admin database migrate` options

Option	Description
`--additional-config=<file>`[16]	Configuration file with additional configuration.
`--expand-commands`	Allow command expansion in config value evaluation.
`--force-btree-indexes-to-range`	Special option for automatically turning all BTREE indexes/constraints into RANGE. Be aware that RANGE indexes are not always the optimal replacement of BTREEs and performance may be affected while the new indexes are populated. See the Neo4j v5 migration guide online for more information. The newly created indexes will be populated in the background on the first database start up following the migration and users should monitor the successful completion of that process. BTREE indexes are not supported in Neo4j 5 and the `neo4j-admin database migrate` command will remove all BTREE indexes and BTREE index-backed constraints. It is recommended to create valid replacements before migrating the database. For more information, see [Upgrade and Migration Guide → Prepare indexes.
`-h, --help`	Show this help message and exit.
`--pagecache=<size>`	The size of the page cache to use for the migration process. The general rule is that values up to the size of the database proportionally increase performance.

Option	Description
`--to-format=standard|high_limit|aligned|block`	Name of the format to migrate the store to. If the format is specified, the target database is migrated to the latest known combination of `MAJOR` and `MINOR` versions of the specified format. If not specified, the tool migrates the target database to the latest known combination of `MAJOR` and `MINOR` versions of the current format.
`--verbose`	Enable verbose output.

> **Note:**
>
> The block format is introduced in Neo4j 5.14 and from Neo4j 5.22, is the default format for all newly-created databases in Enterprise Edition as long as they do not have the `db.format` setting specified. For more information on the block format, see Store formats.

Example

The following example migrates the `movies` database to the latest known combination of `MAJOR` and `MINOR` versions of the `high_limit` format:

```
bin/neo4j-admin database migrate --to-format=high_limit movies
```

Example output

```
2022-11-10 12:55:35.542+0000 INFO  [o.n.c.d.MigrateStoreCommand] Starting migration for database 'movies'
2022-11-10 12:55:36.061+0000 INFO  [o.n.k.i.s.StoreMigrator] 'record-aligned-1.1' has been identified as
the current version of the store
2022-11-10 12:55:36.061+0000 INFO  [o.n.k.i.s.StoreMigrator] 'record-high_limit-1.1' has been identified
as the target version of the store migration
2022-11-10 12:55:36.065+0000 INFO  [o.n.k.i.s.StoreMigrator] Starting migration of database
2022-11-10 12:55:36.154+0000 INFO  [o.n.k.i.s.StoreMigrator] Migrating Store files (1/7):
2022-11-10 12:55:36.828+0000 INFO  [o.n.k.i.s.f.RecordFormatSelector] Selected
RecordFormat:HighLimitV5_0[high_limit-1.1] record format from store
$NEO4J_HOME/data/databases/movies/migrate
2022-11-10 12:55:36.828+0000 INFO  [o.n.k.i.s.f.RecordFormatSelector] Selected format from the store
files: RecordFormat:HighLimitV5_0[high_limit-1.1]
2022-11-10 12:55:37.020+0000 INFO  [o.n.i.b.ImportLogic] Import starting
2022-11-10 12:55:37.924+0000 INFO  [o.n.k.i.s.StoreMigrator]   10% completed
2022-11-10 12:55:37.924+0000 INFO  [o.n.k.i.s.StoreMigrator]   20% completed
2022-11-10 12:55:37.924+0000 INFO  [o.n.k.i.s.StoreMigrator]   30% completed
2022-11-10 12:55:37.924+0000 INFO  [o.n.k.i.s.StoreMigrator]   40% completed
2022-11-10 12:55:37.924+0000 INFO  [o.n.k.i.s.StoreMigrator]   50% completed
2022-11-10 12:55:37.924+0000 INFO  [o.n.k.i.s.StoreMigrator]   60% completed
2022-11-10 12:55:37.924+0000 INFO  [o.n.k.i.s.StoreMigrator]   70% completed
2022-11-10 12:55:37.924+0000 INFO  [o.n.k.i.s.StoreMigrator]   80% completed
2022-11-10 12:55:37.924+0000 INFO  [o.n.k.i.s.StoreMigrator]   90% completed
2022-11-10 12:55:37.924+0000 INFO  [o.n.k.i.s.StoreMigrator]   100% completed
2022-11-10 12:55:37.925+0000 INFO  [o.n.i.b.ImportLogic] Import completed successfully, took 903ms.
Imported:
  171 nodes
  253 relationships
  564 properties
2022-11-10 12:55:38.515+0000 INFO  [o.n.k.i.s.StoreMigrator] Migrating text-1.0 (2/7):
2022-11-10 12:55:38.515+0000 INFO  [o.n.k.i.s.StoreMigrator]   10% completed
2022-11-10 12:55:38.515+0000 INFO  [o.n.k.i.s.StoreMigrator]   20% completed
2022-11-10 12:55:38.515+0000 INFO  [o.n.k.i.s.StoreMigrator]   30% completed
2022-11-10 12:55:38.515+0000 INFO  [o.n.k.i.s.StoreMigrator]   40% completed
2022-11-10 12:55:38.515+0000 INFO  [o.n.k.i.s.StoreMigrator]   50% completed
2022-11-10 12:55:38.515+0000 INFO  [o.n.k.i.s.StoreMigrator]   60% completed
2022-11-10 12:55:38.515+0000 INFO  [o.n.k.i.s.StoreMigrator]   70% completed
```

```
2022-11-10 12:55:38.515+0000 INFO  [o.n.k.i.s.StoreMigrator]   80% completed
2022-11-10 12:55:38.516+0000 INFO  [o.n.k.i.s.StoreMigrator]   90% completed
2022-11-10 12:55:38.516+0000 INFO  [o.n.k.i.s.StoreMigrator]   100% completed
2022-11-10 12:55:38.516+0000 INFO  [o.n.k.i.s.StoreMigrator] Migrating range-1.0 indexes (3/7):
2022-11-10 12:55:38.516+0000 INFO  [o.n.k.i.s.StoreMigrator]   10% completed
2022-11-10 12:55:38.516+0000 INFO  [o.n.k.i.s.StoreMigrator]   20% completed
2022-11-10 12:55:38.516+0000 INFO  [o.n.k.i.s.StoreMigrator]   30% completed
2022-11-10 12:55:38.516+0000 INFO  [o.n.k.i.s.StoreMigrator]   40% completed
2022-11-10 12:55:38.516+0000 INFO  [o.n.k.i.s.StoreMigrator]   50% completed
2022-11-10 12:55:38.516+0000 INFO  [o.n.k.i.s.StoreMigrator]   60% completed
2022-11-10 12:55:38.516+0000 INFO  [o.n.k.i.s.StoreMigrator]   70% completed
2022-11-10 12:55:38.516+0000 INFO  [o.n.k.i.s.StoreMigrator]   80% completed
2022-11-10 12:55:38.516+0000 INFO  [o.n.k.i.s.StoreMigrator]   90% completed
2022-11-10 12:55:38.516+0000 INFO  [o.n.k.i.s.StoreMigrator]   100% completed
2022-11-10 12:55:38.517+0000 INFO  [o.n.k.i.s.StoreMigrator] Migrating Fulltext indexes (4/7):
2022-11-10 12:55:38.517+0000 INFO  [o.n.k.i.s.StoreMigrator]   10% completed
2022-11-10 12:55:38.517+0000 INFO  [o.n.k.i.s.StoreMigrator]   20% completed
2022-11-10 12:55:38.517+0000 INFO  [o.n.k.i.s.StoreMigrator]   30% completed
2022-11-10 12:55:38.517+0000 INFO  [o.n.k.i.s.StoreMigrator]   40% completed
2022-11-10 12:55:38.517+0000 INFO  [o.n.k.i.s.StoreMigrator]   50% completed
2022-11-10 12:55:38.517+0000 INFO  [o.n.k.i.s.StoreMigrator]   60% completed
2022-11-10 12:55:38.517+0000 INFO  [o.n.k.i.s.StoreMigrator]   70% completed
2022-11-10 12:55:38.517+0000 INFO  [o.n.k.i.s.StoreMigrator]   80% completed
2022-11-10 12:55:38.517+0000 INFO  [o.n.k.i.s.StoreMigrator]   90% completed
2022-11-10 12:55:38.517+0000 INFO  [o.n.k.i.s.StoreMigrator]   100% completed
2022-11-10 12:55:38.517+0000 INFO  [o.n.k.i.s.StoreMigrator] Migrating point-1.0 indexes (5/7):
2022-11-10 12:55:38.517+0000 INFO  [o.n.k.i.s.StoreMigrator]   10% completed
2022-11-10 12:55:38.518+0000 INFO  [o.n.k.i.s.StoreMigrator]   20% completed
2022-11-10 12:55:38.518+0000 INFO  [o.n.k.i.s.StoreMigrator]   30% completed
2022-11-10 12:55:38.518+0000 INFO  [o.n.k.i.s.StoreMigrator]   40% completed
2022-11-10 12:55:38.518+0000 INFO  [o.n.k.i.s.StoreMigrator]   50% completed
2022-11-10 12:55:38.518+0000 INFO  [o.n.k.i.s.StoreMigrator]   60% completed
2022-11-10 12:55:38.518+0000 INFO  [o.n.k.i.s.StoreMigrator]   70% completed
2022-11-10 12:55:38.518+0000 INFO  [o.n.k.i.s.StoreMigrator]   80% completed
2022-11-10 12:55:38.518+0000 INFO  [o.n.k.i.s.StoreMigrator]   90% completed
2022-11-10 12:55:38.518+0000 INFO  [o.n.k.i.s.StoreMigrator]   100% completed
2022-11-10 12:55:38.518+0000 INFO  [o.n.k.i.s.StoreMigrator] Migrating Token indexes (6/7):
2022-11-10 12:55:38.518+0000 INFO  [o.n.k.i.s.StoreMigrator]   10% completed
2022-11-10 12:55:38.519+0000 INFO  [o.n.k.i.s.StoreMigrator]   20% completed
2022-11-10 12:55:38.519+0000 INFO  [o.n.k.i.s.StoreMigrator]   30% completed
2022-11-10 12:55:38.519+0000 INFO  [o.n.k.i.s.StoreMigrator]   40% completed
2022-11-10 12:55:38.519+0000 INFO  [o.n.k.i.s.StoreMigrator]   50% completed
2022-11-10 12:55:38.519+0000 INFO  [o.n.k.i.s.StoreMigrator]   60% completed
2022-11-10 12:55:38.519+0000 INFO  [o.n.k.i.s.StoreMigrator]   70% completed
2022-11-10 12:55:38.519+0000 INFO  [o.n.k.i.s.StoreMigrator]   80% completed
2022-11-10 12:55:38.519+0000 INFO  [o.n.k.i.s.StoreMigrator]   90% completed
2022-11-10 12:55:38.519+0000 INFO  [o.n.k.i.s.StoreMigrator]   100% completed
2022-11-10 12:55:38.519+0000 INFO  [o.n.k.i.s.StoreMigrator] Migrating text-2.0 (7/7):
2022-11-10 12:55:38.519+0000 INFO  [o.n.k.i.s.StoreMigrator]   10% completed
2022-11-10 12:55:38.519+0000 INFO  [o.n.k.i.s.StoreMigrator]   20% completed
2022-11-10 12:55:38.519+0000 INFO  [o.n.k.i.s.StoreMigrator]   30% completed
2022-11-10 12:55:38.520+0000 INFO  [o.n.k.i.s.StoreMigrator]   40% completed
2022-11-10 12:55:38.520+0000 INFO  [o.n.k.i.s.StoreMigrator]   50% completed
2022-11-10 12:55:38.520+0000 INFO  [o.n.k.i.s.StoreMigrator]   60% completed
2022-11-10 12:55:38.520+0000 INFO  [o.n.k.i.s.StoreMigrator]   70% completed
2022-11-10 12:55:38.520+0000 INFO  [o.n.k.i.s.StoreMigrator]   80% completed
2022-11-10 12:55:38.520+0000 INFO  [o.n.k.i.s.StoreMigrator]   90% completed
2022-11-10 12:55:38.520+0000 INFO  [o.n.k.i.s.StoreMigrator]   100% completed
2022-11-10 12:55:38.628+0000 INFO  [o.n.k.i.s.StoreMigrator] Starting transaction logs migration.
2022-11-10 12:55:38.660+0000 INFO  [o.n.k.i.s.StoreMigrator] Transaction logs migration completed.
2022-11-10 12:55:38.696+0000 INFO  [o.n.k.i.s.StoreMigrator] Successfully finished migration of database,
took 2s 631ms
2022-11-10 12:55:38.698+0000 INFO  [o.n.c.d.MigrateStoreCommand] Database migration completed successfully
```

Migrate the Neo4j configuration file

You can use the `migrate-configuration` command to migrate a legacy Neo4j configuration file to the current format. The new version will be written in a target configuration directory. The default location for both the source and target configuration directory is the configuration directory specified by `NEO_CONF` or the default configuration directory for this installation. If the source and target directories are the same, the original configuration files will be renamed. A configuration provided using `--additional-config` option

will not be migrated.

Why use the command?

Tip:

- Configuration migration is a purely mechanical process, and using the explicit migration with the `migrate-configuration` command allows you to inspect and customize the output.
- The command output provides valuable insight into the migration process, including notification about settings that could not have been meaningfully migrated, for instance, because the concept or behavior no longer exists in the new `MAJOR` version of the DBMS.

Syntax

The `neo4j-admin server migrate-configuration` command has the following syntax:

```
neo4j-admin server migrate-configuration [-h] [--expand-commands]
                                         [--verbose] [--from-path=<path>]
                                         [--to-path=<path>]
```

Options

The `neo4j-admin server migrate-configuration` command has the following options:

Table 646. `neo4j-admin server migrate-configuration` options

Option	Description
`--expand-commands`	Allow command expansion in config value evaluation.
`--from-path=<path>`	Path to the configuration directory used as a source for the migration.
`-h, --help`	Show this help message and exit.
`--to-path=<path>`	Path to a directory where the migrated configuration files should be written.
`--verbose`	Enable verbose output.

Example

The following example shows how to migrate a legacy configuration file to the current format:

```
bin/neo4j-admin server migrate-configuration --from-path=/path/to/legacy/neo4j-enterprise-4.4.10/conf/
--to-path=/path/to/new/neo4j-enterprise-5.26.25/conf/
```

Example output

```
Keeping original user-logs.xml file at: /neo4j-enterprise-5.1.0/conf/user-logs.xml.old
User logging configuration xml file generated: /neo4j-enterprise-5.1.0/conf/user-logs.xml
Keeping original server-logs.xml file at: /neo4j-enterprise-5.1.0/conf/server-logs.xml.old
Server logging configuration xml file generated: /neo4j-enterprise-5.1.0/conf/server-logs.xml
dbms.directories.import=import MIGRATED -> server.directories.import=import
dbms.tx_state.memory_allocation=ON_HEAP MIGRATED -> db.tx_state.memory_allocation=ON_HEAP
```

```
dbms.connector.bolt.enabled=true MIGRATED -> server.bolt.enabled=true
dbms.connector.http.enabled=true MIGRATED -> server.http.enabled=true
dbms.connector.https.enabled=false MIGRATED -> server.https.enabled=false
metrics.csv.rotation.compression=zip MIGRATED -> server.metrics.csv.rotation.compression=zip
dbms.jvm.additional=-XX:+UseG1GC MIGRATED -> server.jvm.additional=-XX:+UseG1GC
dbms.jvm.additional=-XX:-OmitStackTraceInFastThrow MIGRATED -> server.jvm.additional=-XX:-
OmitStackTraceInFastThrow
dbms.jvm.additional=-XX:+AlwaysPreTouch MIGRATED -> server.jvm.additional=-XX:+AlwaysPreTouch
dbms.jvm.additional=-XX:+UnlockExperimentalVMOptions MIGRATED -> server.jvm.additional=-
XX:+UnlockExperimentalVMOptions
dbms.jvm.additional=-XX:+TrustFinalNonStaticFields MIGRATED -> server.jvm.additional=-
XX:+TrustFinalNonStaticFields
dbms.jvm.additional=-XX:+DisableExplicitGC MIGRATED -> server.jvm.additional=-XX:+DisableExplicitGC
dbms.jvm.additional=-XX:MaxInlineLevel=15 MIGRATED -> server.jvm.additional=-XX:MaxInlineLevel=15
dbms.jvm.additional=-XX:-UseBiasedLocking MIGRATED -> server.jvm.additional=-XX:-UseBiasedLocking
dbms.jvm.additional=-Djdk.nio.maxCachedBufferSize=262144 MIGRATED -> server.jvm.additional=-
Djdk.nio.maxCachedBufferSize=262144
dbms.jvm.additional=-Dio.netty.tryReflectionSetAccessible=true MIGRATED -> server.jvm.additional=-
Dio.netty.tryReflectionSetAccessible=true
dbms.jvm.additional=-Djdk.tls.ephemeralDHKeySize=2048 MIGRATED -> server.jvm.additional=-
Djdk.tls.ephemeralDHKeySize=2048
dbms.jvm.additional=-Djdk.tls.rejectClientInitiatedRenegotiation=true MIGRATED -> server.jvm.additional=-
Djdk.tls.rejectClientInitiatedRenegotiation=true
dbms.jvm.additional=-XX:FlightRecorderOptions=stackdepth=256 MIGRATED -> server.jvm.additional=-
XX:FlightRecorderOptions=stackdepth=256
dbms.jvm.additional=-XX:+UnlockDiagnosticVMOptions MIGRATED -> server.jvm.additional=-
XX:+UnlockDiagnosticVMOptions
dbms.jvm.additional=-XX:+DebugNonSafepoints MIGRATED -> server.jvm.additional=-XX:+DebugNonSafepoints
dbms.jvm.additional=-Dlog4j2.disable.jmx=true MIGRATED -> server.jvm.additional=-Dlog4j2.disable.jmx=true
dbms.windows_service_name=neo4j MIGRATED -> server.windows_service_name=neo4j
Keeping original configuration file at: /neo4j-enterprise-5.1.0/conf/neo4j.conf.old
```

Note:

The example output is not to be used to populate a new Neo4j 5 *neo4j.conf* file.

The 5.x syntactically correct configuration file can be found at */path/to/new/neo4j-enterprise-5.26.25/conf/*, where `/path/to/new/neo4j-enterprise-5.26.25/conf/` is the value of `--to-path=`.

Validate configurations

Introduced in 5.5

The `neo4j-admin server validate-config` command validates the Neo4j and Log4j configurations. It is located in the `bin` directory and is used to validate the configuration files before starting Neo4j.

Syntax

The `neo4j-admin server validate-config` command has the following syntax:

```
neo4j-admin server validate-config [-h] [--expand-commands] [--verbose]
```

Options

The `neo4j-admin server validate-config` command has the following options:

Table 647. `neo4j-admin server validate-config` *options*

Option	Description
`--expand-commands`	Allow command expansion in config value evaluation.
`-h, --help`	Show this help message and exit.
`--verbose`	Enable verbose output.

Example

Invoke `neo4j-admin server validate-config`

```
bin/neo4j-admin server validate-config
```

Example output

```
Validating Neo4j configuration: /path/to/neo4j-enterprise-{neo4j-version-exact}/conf/neo4j.conf
No issues found.

Validating server Log4j configuration: /path/to/neo4j-enterprise-{neo4j-version-exact}conf/user-logs.xml
No issues found.

Validating user Log4j configuration: /path/to/neo4j-enterprise-{neo4j-version-exact}/conf/server-logs.xml
No issues found.

Validation successful.
```

Cypher Shell

About Cypher Shell CLI

Cypher Shell is a command-line tool used to run queries and perform administrative tasks against a Neo4j instance. By default, the shell is interactive, but you can also use it for scripting by passing Cypher directly on the command line or by piping a file with Cypher statements (requires PowerShell on Windows). It communicates via the Bolt protocol.

Cypher Shell is located in the `bin` directory if installed as part of the product. Alternatively, you can download it from Neo4j Deployment Center and install it separately.

Syntax

The syntax for running Cypher Shell is:

```
cypher-shell [-h] [-a ADDRESS]
             [-u USERNAME] [--impersonate IMPERSONATE]
             [-p PASSWORD] [--encryption {true,false,default}]
             [-d DATABASE] [--access-mode {read,write}]
             [--format {auto,verbose,plain}]
             [-P PARAM] [--non-interactive] [--sample-rows SAMPLE-ROWS]
             [--wrap {true,false}] [-v] [--driver-version]
             [-f FILE] [--change-password] [--log [LOG-FILE]]
             [--history HISTORY-BEHAVIOUR]
             [--notifications] [--fail-fast | --fail-at-end]
             [--idle-timeout IDLE-TIMEOUT]
             [cypher]
```

Positional arguments

Option	Description
`cypher`	An optional string of Cypher to execute and then exit.

Named arguments

Option	Description	Default
`-h, --help`	Show this help message and exit.	
`--fail-fast`	Exit and report failure on the first error when reading from a file (this is the default behavior).	
`--fail-at-end`	Exit and report failures at the end of the input when reading from a file.	
`--enable-autocompletions`	Whether to enable Cypher autocompletions inside the CLI, which are disabled by default.	
`--format {auto,verbose,plain}`	Desired output format. Displays the results in tabular format if you use the shell interactively and with minimal formatting if you use it for scripting. `verbose` displays results in a tabular format and prints statistics. `plain` displays data with minimal formatting.	`auto`
`-P PARAM, --param PARAM`	Add a parameter to this session. Example: `-P '{a: 1}'`, `-P '{a: 1, b: duration({seconds: 1})}'`, or using arrow syntax `-P 'a => 1'`. This argument can be specified multiple times.	`[]`
`--non-interactive`	Force non-interactive mode. Only useful when auto-detection fails (like on Windows).	`false`
`--sample-rows SAMPLE-ROWS`	Number of rows sampled to compute table widths (only for format=VERBOSE)	`1000`
`--wrap {true,false}`	Wrap table column values if the column is too narrow (only for format=VERBOSE).	`true`
`-v, --version`	Print Cypher Shell version and exit.	`false`
`--driver-version`	Print Neo4j Driver version and exit.	`false`
`-f FILE, --file FILE`	Pass a file with Cypher statements to be executed. After executing all statements, Cypher Shell shuts down.	
`--change-password`	Change the neo4j user password and exit.	`false`
`--log [LOG-FILE]`	Enable logging to the specified file, or standard error if the file is omitted.	

Option	Description	Default
`--history [HISTORY-BEHAVIOUR]`	File path of query and command history file or `in-memory` for in-memory history. Defaults to <user home>/.neo4j/.cypher_shell_history. It can also be set using the environmental variable NEO4J_CYPHER_SHELL_HISTORY.	
`--notifications`	Enable query notifications in interactive mode.	`false`
`--idle-timeout IDLE-TIMEOUT`	Introduced in 5.22 Closes the application after the specified amount of idle time in interactive mode. You can specify the duration using the format `<hours>h<minutes>m<seconds>s`, for example `1h` (1 hour), `1h30m` (1 hour 30 minutes), or `30m` (30 minutes).	`disable`

Connection arguments

Option	Description	Default
`-a ADDRESS, --address ADDRESS, --uri ADDRESS`	Address and port to connect to. Defaults to neo4j://localhost:7687. Can also be specified using the environment variable `NEO4J_ADDRESS` or `NEO4J_URI`.	`neo4j://localhost:7687`
`-u USERNAME, --username USERNAME`	Username to connect as. Can also be specified using the environment variable NEO4J_USERNAME.	
`--impersonate IMPERSONATE`	User to impersonate.	
`-p PASSWORD, --password PASSWORD`	Password to connect with. Can also be specified using the environment variable NEO4J_PASSWORD.	
`--encryption {true,false,default}`	Whether the connection to Neo4j should be encrypted. This must be consistent with Neo4j's configuration. If choosing `default`, the encryption setting is deduced from the specified address. For example, the `neo4j+ssc` protocol uses encryption.	`default`
`-d DATABASE --database DATABASE`	Database to connect to. Can also be specified using the environment variable NEO4J_DATABASE.	
`--access-mode {read,write}`	Access mode. Defaults to WRITE.	`write`

Running Cypher Shell within the Neo4j distribution

You can connect to a live Neo4j DBMS by running `cypher-shell` and passing in a username and a password argument:

```
bin/cypher-shell -u neo4j -p <password>
```

The output is the following:

```
Connected to Neo4j at neo4j://localhost:7687 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
```

Running Cypher Shell from a different server

You can also install the Cypher Shell tool on a different server (without Neo4j) and connect to a Neo4j DBMS. Cypher Shell requires Java 17.

> **Note:**
>
> DEB/RPM distributions both install Java, if it is not already installed, and the Cypher Shell executable. The *cypher-shell* files are available in the same DEB/RPM Linux repositories as Neo4j.
>
> The TAR distribution contains only the *cypher-shell* files, so you must install Java manually.

1. Download Cypher Shell from Neo4j Deployment Center.
2. Connect to a Neo4j DBMS by running the `cypher-shell` command providing the Neo4j address, a username, and a password:

   ```
   cypher-shell/cypher-shell -a neo4j://IP-address:7687 -u neo4j -p <password>
   ```

   The output is the following:

   ```
   Connected to Neo4j at neo4j://IP-address:7687 as user neo4j.
   Type :help for a list of available commands or :exit to exit the shell.
   Note that Cypher queries must end with a semicolon.
   ```

Changing the access mode

This feature is available in Neo4j 5.20 and later.

By default, the access mode is set to `write`. However, you can change the access mode to `read` or `write` using the `--access-mode` argument when connecting to a Neo4j DBMS with the `cypher-shell` command or by using the `:access-mode` command in the interactive shell. Keep in mind that access mode can affect which servers in a cluster a query can get routed to. For example, a server with `modeConstraint=SECONDARY` can only do reads.

The following is an example of how you can connect to a Neo4j DBMS in read mode and then change the access mode to write in the interactive shell.

1. Connect to a Neo4j DBMS in read mode:

   ```
   bin/cypher-shell -u neo4j -p <password> --access-mode read
   ```

```
Connected to Neo4j using Bolt protocol version 5.4 at neo4j://localhost:7687 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
```

2. Try to create a node in read access mode:

```
create ();
```

```
Writing in read access mode not allowed. Attempted write to neo4j
```

3. Change the access mode to write in the interactive shell:

```
:access-mode write
```

4. Verify the access mode:

```
:access-mode
```

```
Access mode write
```

5. Create a node in write access mode:

```
create ();
```

```
0 rows
ready to start consuming query after 66 ms, results consumed after another 0 ms
Added 1 nodes
```

Tip:

For more information on the `:access-mode` command, run the following command in the interactive shell:

```
:help access-mode
```

```
usage: :access-mode - Display current access mode
:access-mode read - Reconnect with read access mode
:access-mode write - Reconnect with write access mode
```

Available commands

Once in the interactive shell, run the following command to display all available commands:

Example 169. Running `help`

```
:help
```

The output is the following:

```
Available commands:
  :access-mode  View or set access mode
  :begin        Open a transaction
  :commit       Commit the currently open transaction
  :connect      Connects to a database
  :disconnect   Disconnects from database
  :exit         Exit the logger
  :help         Show this help message
  :history      Statement history
  :impersonate  Impersonate user
  :param        Set the value of a query parameter
  :rollback     Rollback the currently open transaction
  :source       Executes Cypher statements from a file
  :sysinfo      Neo4j system information
  :use          Set the active database

For help on a specific command type:
    :help command

Keyboard shortcuts:
    Up and down arrows to access statement history.
    Tab for autocompletion of commands, hit twice to select suggestion from list using arrow keys.

For help on cypher please visit:
    https://neo4j.com/docs/cypher-manual/current/
```

Introduced in Neo4j 5.20

Introduced in Neo4j 5.11

Running Cypher statements

You can run Cypher statements in the following ways:

- Typing Cypher statements directly into the interactive shell.
- Running Cypher statements from a file with the interactive shell.
- Running Cypher statements from a file as a `cypher-shell` argument.

The examples in this section use the `MATCH (n) RETURN n LIMIT 5` Cypher statement and will return 5 nodes from the database.

Example 170. Typing a Cypher statement directly into the interactive shell

```
MATCH (n) RETURN n LIMIT 5;
```

Note:

The following two examples assume a file exists in the same folder you run the `cypher-`

`shell` command from called `example.cypher` with the following contents:

```
MATCH (n) RETURN n LIMIT 5;
```

Example 171. Running Cypher statements from a file with the interactive shell

You can use the `:source` command followed by the file name to run the Cypher statements in that file when in the Cypher interactive shell:

```
:source /path/to/your/example.cypher
```

Example 172. Running Cypher statements from a file as a `cypher-shell` argument.

You can pass a file containing Cypher statements as an argument when running `cypher-shell`.

The examples here use the `--format plain` flag for a simple output.

Using `cat` (UNIX)

```
cat example.cypher | bin/cypher-shell -u neo4j -p <password> --format plain
```

Using `type` (Windows)

```
type example.cypher | bin/cypher-shell.bat -u neo4j -p <password> --format plain
```

Query parameters

Cypher Shell supports querying based on parameters. Use `:param <Cypher Map>` to set parameters or the older arrow syntax `:param name => <Cypher Expression>`. When using the arrow syntax, expressions are restricted to a single line. List current parameters with `:param`. Clear parameters with `:param clear`.

Parameters can be set to any Cypher expression. Some expressions need to be evaluated online and require an open session. The parameter expression is evaluated once. For example, `:param {now: datetime()}` will set the parameter `now` to the current date and time at the time of setting the parameter.

Example 173. Use parameters within Cypher Shell

1. Set the parameter `alias` to `Robin` and `born` to `date('1940-03-20')` using the `:param` keyword:

   ```
   :param {alias: 'Robin', born: date('1940-03-20')}
   ```

2. Check the current parameters using the `:params` keyword:

   ```
   :param
   ```

```
{
  alias: 'Robin',
  born: date('1981-08-01')
}
```

3. Now use the `alias` and `born` parameters in a Cypher query:

```
CREATE (:Person {name : 'Dick Grayson', alias : $alias, born: $born });
```

```
Added 1 nodes, Set 3 properties, Added 1 labels
```

4. Verify the result:

```
MATCH (n) RETURN n;
```

```
+-------------------------------------------------------------------+
| n                                                                 |
+-------------------------------------------------------------------+
| (:Person {name: "Bruce Wayne", alias: "Batman"})                  |
| (:Person {name: "Selina Kyle", alias: ["Catwoman", "The Cat"]})   |
| (:Person {name: "Dick Grayson", alias: "Robin", born: 1940-03-20}) |
+-------------------------------------------------------------------+
3 rows available after 2 ms, consumed after another 2 ms
```

Transactions

Cypher Shell supports explicit and implicit transactions. Transaction states are controlled using the keywords `:begin`, `:commit`, and `:rollback`.

Both explicit and implicit transactions run from Cypher Shell will have default transaction metadata attached that follows the convention (see Attach metadata to a transaction).

Example 174. Use fine-grained transaction control

The example uses a dataset called the Movie Graph, available in the built-in Neo4j Browser guide.

1. Run a query that shows there is only one person in the database, who is born in 1964.

```
MATCH (n:Person) WHERE n.born=1964 RETURN n.name AS name;
```

```
+----------------+
| name           |
+----------------+
| "Keanu Reeves" |
+----------------+

1 row
ready to start consuming query after 9 ms, results consumed after another 0 ms
```

2. Start a transaction and create another person born in the same year:

```
:begin
neo4j# CREATE (:Person {name : 'Edward Mygma', born:1964});
```

```
0 rows
ready to start consuming query after 38 ms, results consumed after another 0 ms
Added 1 nodes, Set 2 properties, Added 1 labels
```

3. If you open a second Cypher Shell session and run the query from step 1, you will notice no changes from the latest `CREATE` statement.

```
MATCH (n:Person) WHERE n.born=1964 RETURN n.name AS name;
```

```
+----------------+
| name           |
+----------------+
| "Keanu Reeves" |
+----------------+

1 row
ready to start consuming query after 9 ms, results consumed after another 0 ms
```

4. Go back to the first session and commit the transaction.

```
neo4j# :commit
```

5. Now, if you run the query from step 1, you will see that Edward Mygma has been added to the database.

```
MATCH (n:Person) WHERE n.born=1964 RETURN n.name AS name;
```

```
+----------------+
| name           |
+----------------+
| "Keanu Reeves" |
| "Edward Mygma" |
+----------------+

2 rows
ready to start consuming query after 1 ms, results consumed after another 1 ms
```

Procedures

Cypher Shell supports running any procedures for which the current user is authorized.

Example 175. Call the `dbms.showCurrentUser` procedure

```
CALL dbms.showCurrentUser();
```

```
+------------------------------+
| username | roles     | flags |
+------------------------------+
```

```
| "neo4j"  | ["admin"] | []    |
+------------------------------+

1 row available after 66 ms, consumed after another 2 ms
```

Supported operating systems

You can use the Cypher Shell CLI via `cmd` on Windows systems, and `bash` on Unix systems.

Other shells may work as intended, but there is no test coverage to guarantee compatibility.

Keyboard shortcuts

The following keyboard commands are available in interactive mode.

Key	Operation
↑ and ↓ (arrow keys)	Access statement history.
⇥ (tab)	Autocompletion of commands and Cypher syntax. Suggestions for Cypher syntax is not complete.
Home (key)	Moves the cursor to the first character in the current line.
End (key)	Moves the cursor to the last character in the current line.

[1] See Tools → Configuration for details.
[2] See Tools → Configuration for details.
[3] See Tools → Configuration for details.
[4] See Tools → Configuration for details.
[5] See Tools → Configuration for details.
[6] Ignored by Parquet import.
[7] To escape quotation marks in the CSV data, you should double the configured character.
[8] See Tools → Configuration for details.
[9] Ignored by Parquet import.
[10] To escape quotation marks in the CSV data, you should double the configured character.
[11] The `--schema` option is available in this version but not yet supported. It will be functional in a future release.
[12] See Tools → Configuration for details.
[13] See Tools → Configuration for details.
[14] Enterprise Edition The `high_limit` format is deprecated in 5.23.
[15] Community Edition The `standard` format is deprecated in 5.23.
[16] See Tools → Configuration for details.

Procedures

This page provides a complete reference to Neo4j's built-in procedures.

It also lists current deprecated procedures and the procedures removed in Neo4j 5, along with their replacements.

The available procedures on a server depends on several factors:

- Neo4j Enterprise Edition provides a larger set of procedures than Neo4j Community Edition.
- Neo4j's Aura instances, but have to be installed separately on on-prem servers.
- Cluster members have procedures that are not available in standalone mode.

To check which procedures are available in your Neo4j DBMS, use the Cypher command `SHOW PROCEDURES`:

List available procedures with default output columns

```
SHOW PROCEDURES
```

List available procedures with full output columns

```
SHOW PROCEDURES YIELD *
```

Note:

Some procedures can only be run by users with `Admin` privileges. Specifically, either the `EXECUTE ADMIN PROCEDURES` privilege or both the `EXECUTE PROCEDURES` and `EXECUTE BOOSTED PROCEDURES` privileges. These procedures are labeled with Admin Only.

For more information, see the `EXECUTE` privileges.

Authentication and authorization

For more information, see Authentication and authorization.

dbms.security.clearAuthCache()

Enterprise Edition Admin Only

Details

Syntax	`dbms.security.clearAuthCache()`
Description	Clears authentication and authorization cache.
Mode	DBMS

dbms.showCurrentUser()

Details

Syntax	`dbms.showCurrentUser() :: (username, roles, flags)`		
Description	Show the current user.		
Return arguments	**Name**	**Type**	**Description**
	`username`	`STRING`	The name of the current user.
	`roles`	`LIST<STRING>`	The roles assigned to the current user.
	`flags`	`LIST<STRING>`	The flags set on the current user.
Mode	DBMS		

Background job management

For more information, see Manage background jobs.

dbms.scheduler.failedJobs()

Enterprise Edition Admin Only

Details

Syntax	`dbms.scheduler.failedJobs() :: (jobId, group, database, submitter, description, type, submitted, executionStart, failureTime, failureDescription)`
Description	List failed job runs. There is a limit for amount of historical data.

Return arguments	Name	Type	Description
	`jobId`	`STRING`	The id of the failed job.
	`group`	`STRING`	The category of the failed job.
	`database`	`STRING`	The name of the database the job failed in.
	`submitter`	`STRING`	The creator of the failed job.
	`description`	`STRING`	Information about the failed job.
	`type`	`STRING`	The interval of the failed job.
	`submitted`	`STRING`	The submission time of the failed job.
	`executionStart`	`STRING`	The start time of the failed job.
	`failureTime`	`STRING`	The failure time of the failed job.
	`failureDescription`	`STRING`	Information about the job failure.
Mode	DBMS		

Note:

This procedure is not considered safe to run from multiple threads. It is therefore not supported by the parallel runtime (introduced in Neo4j 5.13). For more information, see the Cypher Manual → Parallel runtime.

dbms.scheduler.groups()

Enterprise Edition Admin Only

Details

Syntax	`dbms.scheduler.groups() :: (group, threads)`
Description	List the job groups that are active in the database internal job scheduler.

Return arguments	Name	Type	Description
	`group`	`STRING`	The name of the job group.
	`threads`	`INTEGER`	The number of active threads in that job group.
Mode	DBMS		

> **Note:**
>
> This procedure is not considered safe to run from multiple threads. It is therefore not supported by the parallel runtime (introduced in Neo4j 5.13). For more information, see the Cypher Manual → Parallel runtime.

dbms.scheduler.jobs()

Enterprise Edition | Admin Only

Details

Syntax	`dbms.scheduler.jobs() :: (jobId, group, submitted, database, submitter, description, type, scheduledAt, period, state, currentStateDescription)`
Description	List all jobs that are active in the database internal job scheduler.

Return arguments	Name	Type	Description
	`jobId`	`STRING`	The id of the job.
	`group`	`STRING`	The category of the job.
	`submitted`	`STRING`	The submission time of the job.
	`database`	`STRING`	The name of the database the job is in.
	`submitter`	`STRING`	The creator of the job.
	`description`	`STRING`	Information about the job.
	`type`	`STRING`	The interval of the job.
	`scheduledAt`	`STRING`	The start time of the job.
	`period`	`STRING`	The interval for jobs run periodically.
	`state`	`STRING`	The state of the job: ('EXECUTING', 'SCHEDULED').
	`currentStateDescription`	`STRING`	A description of the job state.
Mode	DBMS		

Note:

This procedure is not considered safe to run from multiple threads. It is therefore not supported by the parallel runtime (introduced in Neo4j 5.13). For more information, see the Cypher Manual → Parallel runtime.

Change Data Capture (CDC)

For more information, see the CDC documentation.

cdc.current()

Enterprise Edition | Introduced in 5.14 | Beta | Deprecated in 5.17

Details

Syntax	`cdc.current() :: (id)`
Description	Returns the current change identifier that can be used to stream changes from.

Return arguments	Name	Type	Description
	`id`	`STRING`	The change identifier.
Mode	READ		
Replaced by	`db.cdc.current()`		

cdc.earliest()

Enterprise Edition | Introduced in 5.13 | Beta | Deprecated in 5.17

Details

Syntax	`cdc.earliest() :: (id)`		
Description	Returns the earliest change identifier that can be used to stream changes from.		
Return arguments	Name	Type	Description
	`id`	`STRING`	The change identifier.
Mode	READ		
Replaced by	`db.cdc.earliest()`		

cdc.query()

Enterprise Edition | Introduced in 5.13 | Beta | Deprecated in 5.17 | Admin Only

Details

Syntax	`cdc.query([ from selectors ]) :: (id, txId, seq, metadata, event)`		
Description	Query changes happened from the provided change identifier.		
Input arguments	Name	Type	Description
	`from`	`STRING`	The change identifier to query changes from.
	`selectors`	`LIST<MAP>`	An optional list of selectors to filter out changes.

Return arguments	Name	Type	Description
	`id`	`STRING`	The change identifier for this change event. Used to continue querying from this change event.
	`txId`	`INTEGER`	A number identifying which transaction the change happened in, unique in combination with seq.
	`seq`	`INTEGER`	A number used for ordering changes that happened in the same transaction.
	`metadata`	`MAP`	Metadata from the transaction that caused this change event.
	`event`	`MAP`	The retrieved changes on the affected entity.
Mode	READ		
Replaced by	`db.cdc.query()`		

db.cdc.current()

Enterprise Edition | Introduced in 5.17 | Beta

Details

Syntax	`db.cdc.current() :: (id)`		
Description	Returns the current change identifier that can be used to stream changes from.		
Return arguments	Name	Type	Description
	`id`	`STRING`	The change identifier.
Mode	READ		

db.cdc.earliest()

Introduced in 5.17 | Beta

Details

Syntax	`db.cdc.earliest() :: (id)`

Description	Returns the earliest change identifier that can be used to stream changes from.		
Return arguments	Name	Type	Description
	`id`	`STRING`	The change identifier.
Mode	READ		

db.cdc.query()

Enterprise Edition | Introduced in 5.17 | Beta | Admin Only

Details

Syntax	`db.cdc.query([ from selectors ]) :: (id, txId, seq, metadata, event)`		
Description	Query changes happened from the provided change identifier.		
Input arguments	Name	Type	Description
	`from`	`STRING`	The change identifier to query changes from.
	`selectors`	`LIST<MAP>`	An optional list of selectors to filter out changes.
Return arguments	Name	Type	Description
	`id`	`STRING`	The change identifier for this change event. Used to continue querying from this change event.
	`txId`	`INTEGER`	A number identifying which transaction the change happened in, unique in combination with seq.
	`seq`	`INTEGER`	A number used for ordering changes that happened in the same transaction.
	`metadata`	`MAP`	Metadata from the transaction that caused this change event.
	`event`	`MAP`	The retrieved changes on the affected entity.
Mode	READ		

Cluster management

For more information, see Clustering.

dbms.cluster.checkConnectivity()

Enterprise Edition Admin Only

Details

Syntax	`dbms.cluster.checkConnectivity([ port-name server ]) :: (serverId, name, address, mode-constraint, port-name, port-address, result)`		
Description	Check the connectivity of this instance to other cluster members. Not all ports are relevant to all members. Valid values for 'port-name' are: [CLUSTER, INTRA_BOLT, RAFT]		
Input arguments	Name	Type	Description
	`port-name`	`STRING`	The name of the port: ('CLUSTER', 'RAFT').
	`server`	`STRING`	The id of the server to check connectivity of.
Return arguments	Name	Type	Description
	`serverId`	`STRING`	The id of the checked server.
	`name`	`STRING`	name :: STRING
	`address`	`STRING`	address :: STRING
	`mode-constraint`	`STRING`	The instance mode constraint of the server.
	`port-name`	`STRING`	The name of the checked port.
	`port-address`	`STRING`	The address of the checked port.
	`result`	`STRING`	Whether the check was successful or not.
Mode	DBMS		

dbms.cluster.cordonServer()

Enterprise Edition

Details

Syntax	`dbms.cluster.cordonServer(server)`		
Description	Mark a server in the topology as not suitable for new allocations. It will not force current allocations off the server. This is useful when deallocating databases when you have multiple unavailable servers.		
Input arguments	Name	Type	Description
	`server`	`STRING`	The name or id of the server to be cordoned.
Mode	WRITE		
Required privileges	`SERVER MANAGEMENT`		

> **Note:**
>
> Before Neo4j 5.23, the procedure `dbms.cluster.cordonServer()` can be run only by users with `Admin` privileges. Since Neo4j 5.23, it can be run with the `SERVER MANAGEMENT` privilege. It will still run with the `Admin` privilege, but that should be considered deprecated.

dbms.cluster.moveToNextDiscoveryVersion()

Enterprise Edition | Introduced in 5.26 | Admin Only

Details

Syntax	`dbms.cluster.moveToNextDiscoveryVersion()`
Description	The procedure triggers a switch to the next discovery service version for all known members of the cluster (as listed in the system database and discovery). For example, if the current member's discovery version is V1_ONLY, it will switch all members to V1_OVER_V2. In case of failure, the user must manually resolve the issue.
Mode	DBMS

dbms.cluster.deallocateDatabaseFromServer()

Enterprise Edition | Introduced in 5.23

Details

Syntax	`dbms.cluster.deallocateDatabaseFromServer(server, database [, dryrun ]) :: (database, fromServerName, fromServerId, toServerName, toServerId, mode)`
Description	Deallocate a specific user database from a specific server.

Input arguments	Name	Type	Description
	`server`	`STRING`	The id of the server to deallocate from.
	`database`	`STRING`	The name of the database to deallocate.
	`dryrun`	`BOOLEAN`	Set to `true` to dry run the procedure.
Return arguments	Name	Type	Description
	`database`	`STRING`	The name of the database.
	`fromServerName`	`STRING`	The name of the server.
	`fromServerId`	`STRING`	The id of the server.
	`toServerName`	`STRING`	The name of the server.
	`toServerId`	`STRING`	The id of the server.
	`mode`	`STRING`	The mode in which the database is hosted.
Mode	WRITE		
Required privileges	`SERVER MANAGEMENT`		

dbms.cluster.deallocateDatabaseFromServers()

Enterprise Edition | Introduced in 5.23

Details

Syntax	`dbms.cluster.deallocateDatabaseFromServers(servers, database [, dryrun ]) :: (database, fromServerName, fromServerId, toServerName, toServerId, mode)`		
Description	Deallocate a specific user database from a list of servers.		
Input arguments	Name	Type	Description
	`servers`	`LIST<STRING>`	The ids of the servers to deallocate from.
	`database`	`STRING`	The id of the database to deallocate.
	`dryrun`	`BOOLEAN`	Set to `true` to dry run the procedure.

Return arguments	Name	Type	Description
	`database`	`STRING`	The name of the database.
	`fromServerName`	`STRING`	The name of the server.
	`fromServerId`	`STRING`	The id of the server.
	`toServerName`	`STRING`	The name of the server.
	`toServerId`	`STRING`	The id of the server.
	`mode`	`STRING`	The mode in which the database is hosted.
Mode	WRITE		
Required privileges	`SERVER MANAGEMENT`		

dbms.cluster.deallocateNumberOfDatabases()

Enterprise Edition | Introduced in 5.23

Details

Syntax	`dbms.cluster.deallocateNumberOfDatabases(server, number [, dryrun ]) :: (database, fromServerName, fromServerId, toServerName, toServerId, mode)`		
Description	Deallocate a number of user databases from a specific server.		
Input arguments	Name	Type	Description
	`server`	`STRING`	The id of the server to deallocate from.
	`number`	`INTEGER`	The number of databases to deallocate.
	`dryrun`	`BOOLEAN`	Set to `true` to dry run the procedure.

Return arguments	Name	Type	Description
	`database`	`STRING`	The name of the database.
	`fromServerName`	`STRING`	The name of the server.
	`fromServerId`	`STRING`	The id of the server.
	`toServerName`	`STRING`	The name of the server.
	`toServerId`	`STRING`	The id of the server.
	`mode`	`STRING`	The mode in which the database is hosted.
Mode	WRITE		
Required privileges	`SERVER MANAGEMENT`		

dbms.cluster.protocols()

Enterprise Edition

Details

Syntax	`dbms.cluster.protocols() :: (orientation, remoteAddress, applicationProtocol, applicationProtocolVersion, modifierProtocols)`		
Description	Overview of installed protocols		
Return arguments	Name	Type	Description
	`orientation`	`STRING`	Direction of the protocol (inbound or outbound).
	`remoteAddress`	`STRING`	The socket address this protocol is available on.
	`applicationProtocol`	`STRING`	The name of the protocol.
	`applicationProtocolVersion`	`INTEGER`	The version of the protocol.
	`modifierProtocols`	`STRING`	Installed modifier protocols, for example, compression.
Mode	DBMS		

dbms.cluster.readReplicaToggle()

Admin Only Deprecated in 5.6 Enterprise Edition

Details

Syntax	`dbms.cluster.readReplicaToggle(databaseName, pause) :: (state)`		
Description	The toggle can pause or resume read replica (deprecated in favor of dbms.cluster.secondaryReplicationDisable)		
Input arguments	Name	Type	Description
	`databaseName`	`STRING`	The name of the database to toggle the secondary replication process for.
	`pause`	`BOOLEAN`	Whether or not to enable/disable the secondary replication process.
Return arguments	Name	Type	Description
	`state`	`STRING`	The current state of the secondary replication process.
Mode	DBMS		

Tip:

What is it for?

You can perform a point-in-time backup, as the backup will contain only the transactions up to the point where the transaction pulling was paused. Follow these steps to do so:

1. Connect directly to the server hosting the database in secondary mode. (Neo4j Driver use `bolt://` or use the HTTP API).
2. Pause transaction pulling for the specified database.
3. Back up the database, see Back up an online database.

If connected directly to a server hosting a database in secondary mode, Data Scientists can execute analysis on a specific database that is paused, the data will not unexpectedly change while performing the analysis.

Note:

This procedure can only be executed on a database that runs in a secondary role on the connected server.

Pause transaction pulling for database `neo4j`

```
CALL dbms.cluster.readReplicaToggle("neo4j", true)
```

Resume transaction pulling for database `neo4j`

```
CALL dbms.cluster.readReplicaToggle("neo4j", false)
```

dbms.cluster.reallocateDatabase()

Enterprise Edition | Introduced in 5.23

Details

Syntax	`dbms.cluster.reallocateDatabase(database [, dryrun ]) :: (database, fromServerName, fromServerId, toServerName, toServerId, mode)`		
Description	Reallocate a specific database.		
Input arguments	Name	Type	Description
	`database`	`STRING`	The name of the database to reallocate.
	`dryrun`	`BOOLEAN`	Set to `true` to dry run the procedure.
Return arguments	Name	Type	Description
	`database`	`STRING`	The name of the database.
	`fromServerName`	`STRING`	The name of the server.
	`fromServerId`	`STRING`	The id of the server.
	`toServerName`	`STRING`	The name of the server.
	`toServerId`	`STRING`	The id of the server.
	`mode`	`STRING`	The mode in which the database is hosted.
Mode	WRITE		
Required privileges	`SERVER MANAGEMENT`		

dbms.cluster.reallocateNumberOfDatabases()

Enterprise Edition | Introduced in 5.23

Details

Syntax	`dbms.cluster.reallocateNumberOfDatabases(number [, dryrun ]) :: (database, fromServerName, fromServerId, toServerName, toServerId, mode)`
Description	Reallocate a specified number of user databases.

Input arguments	Name	Type	Description
	`number`	`INTEGER`	The number of databases to reallocate.
	`dryrun`	`BOOLEAN`	Set to `true` to dry run the procedure.
Return arguments	**Name**	**Type**	**Description**
	`database`	`STRING`	The name of the database.
	`fromServerName`	`STRING`	The name of the server.
	`fromServerId`	`STRING`	The id of the server.
	`toServerName`	`STRING`	The name of the server.
	`toServerId`	`STRING`	The id of the server.
	`mode`	`STRING`	The mode in which the database is hosted.
Mode	WRITE		
Required privileges	`SERVER MANAGEMENT`		

dbms.cluster.recreateDatabase()

Enterprise Edition | Introduced in 5.24

Details

Syntax	`dbms.cluster.recreateDatabase(database :: STRING, options = {} :: MAP)`
Description	Recreates a database while keeping all RBAC settings. The procedure initiates a process, which when complete, will have synchronized and started all database instances within the cluster.

Input arguments	Name	Type	Description
	`database`	`STRING`	database :: STRING
	`options`	`MAP`	options = {} :: MAP
Mode	WRITE		

> **Note:**
>
> It is mandatory to specify either `seedURI` or `seedingServers` as seeding options in the `options` field.
>
> If no topology option is defined, the database will be recreated with the previous topology.

Further details on how to use the `dbms.cluster.recreateDatabase()` procedure are provided in the section Database administration → Recreate a database.

dbms.cluster.routing.getRoutingTable()

Deprecated in 5.21 | Enterprise Edition

Details

Syntax	`dbms.cluster.routing.getRoutingTable(context [, database ]) :: (ttl, servers)`		
Description	Returns the advertised bolt capable endpoints for a given database, divided by each endpoint's capabilities. For example, an endpoint may serve read queries, write queries, and/or future `getRoutingTable` requests.		
Input arguments	**Name**	**Type**	**Description**
	`context`	`MAP`	Routing context, for example, routing policies.
	`database`	`STRING`	The database to get a routing table for.
Return arguments	**Name**	**Type**	**Description**
	`ttl`	`INTEGER`	Time to live (in seconds) for the routing table.
	`servers`	`LIST<MAP>`	Servers grouped by whether they are readers, writers, or routers.
Mode	DBMS		

dbms.cluster.secondaryReplicationDisable()

Enterprise Edition | Introduced in 5.6 | Admin Only

Details

Syntax	`dbms.cluster.secondaryReplicationDisable(databaseName, pause) :: (state)`
Description	The toggle can pause or resume the secondary replication process.

Input arguments	Name	Type	Description
	`databaseName`	`STRING`	The name of the database to toggle the secondary replication process for.
	`pause`	`BOOLEAN`	Whether or not to enable/disable the secondary replication process.
Return arguments	**Name**	**Type**	**Description**
	`state`	`STRING`	The current state of the secondary replication process.
Mode	DBMS		

Tip:

What is it for?

You can perform a point-in-time backup, as the backup will contain only the transactions up to the point where the transaction pulling was paused. Follow these steps to do so:

1. Connect directly to the server hosting the database in secondary mode. (Neo4j Driver use `bolt://` or use the HTTP API).
2. Pause transaction pulling for the specified database.
3. Back up the database, see Back up an online database.

If connected directly to a server hosting a database in secondary mode, Data Scientists can execute analysis on a specific database that is paused, the data will not unexpectedly change while performing the analysis.

Note:

This procedure can only be executed on a database that runs in a secondary role on the connected server.

Pause transaction pulling for database `neo4j`

```
CALL dbms.cluster.secondaryReplicationDisable("neo4j", true)
```

Resume transaction pulling for database `neo4j`

```
CALL dbms.cluster.secondaryReplicationDisable("neo4j", false)
```

dbms.cluster.setAutomaticallyEnableFreeServers()

Enterprise Edition

Details

Syntax	`dbms.cluster.setAutomaticallyEnableFreeServers(autoEnable)`		
Description	With this method you can set whether free servers are automatically enabled.		
Input arguments	Name	Type	Description
	`autoEnable`	`BOOLEAN`	Whether or not to automatically enable free servers.
Mode	WRITE		
Required privileges	`SERVER MANAGEMENT`		

Note:

Before Neo4j 5.23, the procedure `dbms.cluster.setAutomaticallyEnableFreeServers()` can be run only by users with the `Admin` privileges. Since Neo4j 5.23, it can be run with the `SERVER MANAGEMENT` privilege. It will still run with the `Admin` privilege, but that should be considered deprecated.

dbms.cluster.showParallelDiscoveryState()

Enterprise Edition Introduced in 5.22 Admin Only

Details

Syntax	`dbms.cluster.showParallelDiscoveryState() :: (mode, stateComparison, v1ServerCount, v2ServerCount)`		
Description	Compare the states of Discovery service V1 and Discovery service V2.		
Return arguments	Name	Type	Description
	`mode`	`STRING`	mode :: STRING
	`stateComparison`	`STRING`	stateComparison :: STRING
	`v1ServerCount`	`STRING`	v1ServerCount :: STRING
	`v2ServerCount`	`STRING`	v2ServerCount :: STRING
Mode	DBMS		

dbms.cluster.statusCheck()

Enterprise Edition | Introduced in 5.24 | Admin Only

Details

Syntax	`dbms.cluster.statusCheck(databases, timeoutMilliseconds) :: (database, serverId, serverName, address, replicationSuccessful, memberStatus, recognisedLeader, recognisedLeaderTerm, requester, error)`		
Description	Performs a rafted status check.		
Input arguments	**Name**	**Type**	**Description**
	`databases`	`LIST<STRING>`	databases :: LIST<STRING>
	`timeoutMilliseconds`	`INTEGER`	timeoutMilliseconds = null :: INTEGER
Return arguments	**Name**	**Type**	**Description**
	`database`	`STRING`	database :: STRING
	`serverId`	`STRING`	serverId :: STRING
	`serverName`	`STRING`	serverName :: STRING
	`address`	`STRING`	address :: STRING
	`replicationSuccessful`	`BOOLEAN`	replicationSuccessful :: BOOLEAN
	`memberStatus`	`STRING`	memberStatus :: STRING
	`recognisedLeader`	`STRING`	recognisedLeader :: STRING
	`recognisedLeaderTerm`	`INTEGER`	recognisedLeaderTerm :: INTEGER
	`requester`	`BOOLEAN`	requester :: BOOLEAN
	`error`	`STRING`	error :: STRING
Mode	DBMS		

dbms.cluster.switchDiscoveryServiceVersion()

Enterprise Edition | Introduced in 5.22 | Admin Only

Details

Syntax	`dbms.cluster.switchDiscoveryServiceVersion(mode)`

Description	Allows you to select which discovery service should be started. Possible values are: • `V1_ONLY` — it runs only discovery service v1. • `V1_OVER_V2` — it runs both Discovery Service V1 and Discovery Service V2, where V1 is the main service and V2 runs in the background. • `V2_OVER_V1` — it runs both Discovery Service V1 and Discovery Service V2, where V2 is the main service and V1 runs in the background. • `V2_ONLY` — it runs only discovery service v2.		
Input arguments	Name	Type	Description
	`mode`	`STRING`	mode :: STRING
Mode	DBMS		

dbms.cluster.uncordonServer()

Enterprise Edition | Deprecated in 5.23

Details

Syntax	`dbms.cluster.uncordonServer(server)`		
Description	Remove the cordon on a server, returning it to 'enabled'.		
Input arguments	Name	Type	Description
	`server`	`STRING`	The name or id of the server to be uncordoned.
Mode	WRITE		
Replaced by	`ENABLE SERVER`		
Required privileges	`SERVER MANAGEMENT`		

> **Note:**
>
> Before Neo4j 5.23, the procedure `dbms.cluster.uncordonServer()` can be run only by users with `Admin` privileges. Since Neo4j 5.23, it can be run with the `SERVER MANAGEMENT` privilege. It will still run with the `Admin` privilege, but that should be considered deprecated.

dbms.setDatabaseAllocator()

Enterprise Edition | Admin Only | Deprecated in 5.23

Details

Syntax	`dbms.setDatabaseAllocator(allocator)`

Description	With this method you can set the allocator that is responsible for selecting servers for hosting databases.		
Input arguments	Name	Type	Description
	`allocator`	`STRING`	The name of the allocator.
Mode	WRITE		

dbms.setDefaultAllocationNumbers()

Enterprise Edition Admin Only

Details

Syntax	`dbms.setDefaultAllocationNumbers(primaries, secondaries)`		
Description	With this method you can set the default number of primaries and secondaries.		
Input arguments	Name	Type	Description
	`primaries`	`INTEGER`	The default number of primaries.
	`secondaries`	`INTEGER`	The default number of secondaries.
Mode	WRITE		

dbms.showTopologyGraphConfig()

Enterprise Edition Admin Only

Details

Syntax	`dbms.showTopologyGraphConfig() :: (allocator, defaultPrimariesCount, defaultSecondariesCount, defaultDatabase, autoEnableFreeServers)`
Description	With this method the configuration of the Topology Graph can be displayed.

Return arguments	Name	Type	Description
	`allocator`	`STRING`	The name of the allocator.
	`defaultPrimariesCount`	`INTEGER`	The default number of primaries.
	`defaultSecondariesCount`	`INTEGER`	The default number of secondaries.
	`defaultDatabase`	`STRING`	The name of the default database.
	`autoEnableFreeServers`	`BOOLEAN`	Whether or not to automatically enable free servers.
Mode	READ		

Configuration and DBMS info

For more information, see Configuration.

dbms.checkConfigValue()

Enterprise Edition Admin Only

Details

Syntax	`dbms.checkConfigValue(setting, value) :: (valid, message)`		
Description	Check if a potential config setting value is valid.		
Input arguments	Name	Type	Description
	`setting`	`STRING`	The name of the setting.
	`value`	`STRING`	The setting value to verify.
Return arguments	Name	Type	Description
	`valid`	`BOOLEAN`	Whether or not the setting value is valid.
	`message`	`STRING`	Details about the outcome of the procedure.
Mode	DBMS		

Note:

This procedure is not considered safe to run from multiple threads. It is therefore not supported by the parallel runtime (introduced in Neo4j 5.13). For more information, see the Cypher Manual → Parallel runtime.

dbms.components()

Details

Syntax	`dbms.components() :: (name, versions, edition)`		
Description	List DBMS components and their versions.		
Return arguments	Name	Type	Description
	`name`	`STRING`	The name of the component.
	`versions`	`LIST<STRING>`	The installed versions of the component.
	`edition`	`STRING`	The Neo4j edition of the DBMS.
Mode	DBMS		

dbms.info()

Details

Syntax	`dbms.info() :: (id, name, creationDate)`		
Description	Provides information regarding the DBMS.		
Return arguments	Name	Type	Description
	`id`	`STRING`	The id of the DBMS.
	`name`	`STRING`	The name of the DBMS.
	`creationDate`	`STRING`	The creation date of the DBMS.
Mode	DBMS		

dbms.listCapabilities()

Details

Syntax	`dbms.listCapabilities() :: (name, description, value)`
Description	List capabilities.

Return arguments	Name	Type	Description
	`name`	`STRING`	The full name of the capability (e.g. "dbms.instance.version").
	`description`	`STRING`	The capability description (e.g. "Neo4j version this instance is running").
	`value`	`ANY`	The capability object if it is present in the system (e.g. "5.20.0").
Mode	DBMS		

dbms.listConfig()

Admin Only

Details

Syntax	`dbms.listConfig([ searchString ]) :: (name, description, value, dynamic, defaultValue, startupValue, explicitlySet, validValues)`		
Description	List the currently active configuration settings of Neo4j.		
Input arguments	Name	Type	Description
	`searchString`	`STRING`	A string that filters on the name of config settings.

Return arguments	Name	Type	Description
	`name`	`STRING`	The name of the setting.
	`description`	`STRING`	The description of the setting.
	`value`	`STRING`	The set value of the setting.
	`dynamic`	`BOOLEAN`	If the setting can be set dynamically or not.
	`defaultValue`	`STRING`	The default value of the setting.
	`startupValue`	`STRING`	The value of the setting when the database started.
	`explicitlySet`	`BOOLEAN`	Whether or not the setting was explicitly set.
	`validValues`	`STRING`	A description of the valid values.
Mode	DBMS		

dbms.setConfigValue()

Enterprise Edition | Admin Only | Not available on Aura

Details

Syntax	`dbms.setConfigValue(setting, value)`		
Description	Update a given setting value. Passing an empty value results in removing the configured value and falling back to the default value. Changes do not persist and are lost if the server is restarted. In a clustered environment, `dbms.setConfigValue` affects only the cluster member it is run against.		
Input arguments	Name	Type	Description
	`setting`	`STRING`	The name of the setting.
	`value`	`STRING`	The value to set.
Mode	DBMS		

dbms.listPools()

Enterprise Edition

Details

Syntax	`dbms.listPools() :: (pool, databaseName, heapMemoryUsed, heapMemoryUsedBytes, nativeMemoryUsed, nativeMemoryUsedBytes, freeMemory, freeMemoryBytes, totalPoolMemory, totalPoolMemoryBytes)`		
Description	List all memory pools, including sub pools, currently registered at this instance that are visible to the user.		
Return arguments	Name	Type	Description
	`pool`	`STRING`	The name of the memory pool.
	`databaseName`	`STRING`	The name of the database.
	`heapMemoryUsed`	`STRING`	The amount of heap memory used.
	`heapMemoryUsedBytes`	`STRING`	The amount of heap memory used in bytes.
	`nativeMemoryUsed`	`STRING`	The amount of native memory used.
	`nativeMemoryUsedBytes`	`STRING`	The amount of native memory used in bytes.
	`freeMemory`	`STRING`	The amount of free memory.
	`freeMemoryBytes`	`STRING`	The amount of free memory in bytes.
	`totalPoolMemory`	`STRING`	The total pool memory.
	`totalPoolMemoryBytes`	`STRING`	The total pool memory in bytes.
Mode	DBMS		

Note:

This procedure is not considered safe to run from multiple threads. It is therefore not supported by the parallel runtime (introduced in Neo4j 5.13). For more information, see the Cypher Manual → Parallel runtime.

Connection management

For more information, see Manage connections.

dbms.listConnections()

Details

Syntax	`dbms.listConnections() :: (connectionId, connectTime, connector, username, userAgent, serverAddress, clientAddress)`		
Description	List all accepted network connections at this instance that are visible to the user.		
Return arguments	Name	Type	Description
	`connectionId`	`STRING`	The id of the connection.
	`connectTime`	`STRING`	The time the connection was established, formatted according to the ISO-8601 Standard.
	`connector`	`STRING`	The protocol of the connector.
	`username`	`STRING`	The username of the connected user.
	`userAgent`	`STRING`	The active agent.
	`serverAddress`	`STRING`	The address of the connected server.
	`clientAddress`	`STRING`	The address of the connected client.
Mode	DBMS		

dbms.killConnection()

Details

Syntax	`dbms.killConnection(id) :: (connectionId, username, message)`		
Description	Kill network connection with the given connection id.		
Input arguments	Name	Type	Description
	`id`	`STRING`	The id of the connection to kill.
Return arguments	Name	Type	Description
	`connectionId`	`STRING`	The id of the connection killed.
	`username`	`STRING`	The username of the user of the killed connection.
	`message`	`STRING`	Details about the outcome of the procedure.

Mode	DBMS

dbms.killConnections()

Details

Syntax	`dbms.killConnections(ids) :: (connectionId, username, message)`		
Description	Kill all network connections with the given connection ids.		
Input arguments	Name	Type	Description
	`ids`	`LIST<STRING>`	The ids of the connections to kill.
Return arguments	Name	Type	Description
	`connectionId`	`STRING`	The id of the connection killed.
	`username`	`STRING`	The username of the user of the killed connection.
	`message`	`STRING`	Details about the outcome of the procedure.
Mode	DBMS		

Database management

For more information, see Database administration and Database internals and transactional behavior.

db.checkpoint()

Enterprise Edition

Details

Syntax	`db.checkpoint() :: (success, message)`
Description	Initiate and wait for a new check point, or wait any already on-going check point to complete. Note that this temporarily disables the `db.checkpoint.iops.limit` setting in order to make the check point complete faster. This might cause transaction throughput to degrade slightly, due to increased IO load.

Return arguments	Name	Type	Description
	`success`	`BOOLEAN`	Whether the checkpoint has successfully completed.
	`message`	`STRING`	Details about the outcome of the procedure.
Mode	DBMS		

> **Note:**
>
> This procedure is not considered safe to run from multiple threads. It is therefore not supported by the parallel runtime (introduced in Neo4j 5.13). For more information, see the Cypher Manual → Parallel runtime.

db.info()

Details

Syntax	`db.info() :: (id, name, creationDate)`		
Description	Provides information regarding the database.		
Return arguments	Name	Type	Description
	`id`	`STRING`	The id of the database.
	`name`	`STRING`	The name of the database.
	`creationDate`	`STRING`	The creation date of the database, formatted according to the ISO-8601 Standard.
Mode	READ		

dbms.listActiveLocks()

Enterprise Edition

Details

Syntax	`dbms.listActiveLocks(queryId) :: (mode, resourceType, resourceId)`
Description	List the active lock requests granted for the transaction executing the query with the given query id.

Input arguments	Name	Type	Description
	`queryId`	`STRING`	The id of the query to check for active locks on.
Return arguments	Name	Type	Description
	`mode`	`STRING`	The lock type: ('SHARED', 'EXCLUSIVE').
	`resourceType`	`STRING`	The locked resource.
	`resourceId`	`INTEGER`	The id of the locked resource.
Mode	DBMS		

> **Note:**
>
> This procedure is not considered safe to run from multiple threads. It is therefore not supported by the parallel runtime (introduced in Neo4j 5.13). For more information, see the Cypher Manual → Parallel runtime.

db.listLocks()

Enterprise Edition | Admin Only

Details

Syntax	`db.listLocks() :: (mode, resourceType, resourceId, transactionId)`		
Description	List all locks at this database.		
Return arguments	Name	Type	Description
	`mode`	`STRING`	The locking mode this lock is using, either "SHARED" or "EXCLUSIVE".
	`resourceType`	`STRING`	The type of resource (e.g. nodes, relationships, labels) this lock protects.
	`resourceId`	`INTEGER`	The id of the resource this lock protects.
	`transactionId`	`STRING`	The id of the transaction that owns this lock.
Mode	DBMS		

db.ping()

Details

Syntax	`db.ping() :: (success)`		
Description	This procedure can be used by client side tooling to test whether they are correctly connected to a database. The procedure is available in all databases and always returns true. A faulty connection can be detected by not being able to call this procedure.		
Return arguments	Name	Type	Description
	`success`	`BOOLEAN`	Whether or not the connection call to the database has been successful.
Mode	READ		

dbms.routing.getRoutingTable()

Details

Syntax	`dbms.routing.getRoutingTable(context [, database ]) :: (ttl, servers)`		
Description	Returns the advertised bolt capable endpoints for a given database, divided by each endpoint's capabilities. For example, an endpoint may serve read queries, write queries, and/or future `getRoutingTable` requests.		
Input arguments	Name	Type	Description
	`context`	`MAP`	Routing context, for example, routing policies.
	`database`	`STRING`	The database to get a routing table for.
Return arguments	Name	Type	Description
	`ttl`	`INTEGER`	Time to live (in seconds) for the routing table.
	`servers`	`LIST<MAP>`	Servers grouped by whether they are readers, writers, or routers.
Mode	DBMS		

dbms.setDefaultDatabase()

Enterprise Edition Admin Only

Details

Syntax	`dbms.setDefaultDatabase(databaseName) :: (result)`		
Description	Change the default database to the provided value. The database must exist and the old default database must be stopped.		
Input arguments	Name	Type	Description
	`databaseName`	`STRING`	The name of the database.
Return arguments	Name	Type	Description
	`result`	`STRING`	Information about the default database.
Mode	WRITE		

dbms.quarantineDatabase()

Enterprise Edition Admin Only

Details

Syntax	`dbms.quarantineDatabase(databaseName, setStatus [, reason ]) :: (databaseName, quarantined, result)`		
Description	Place a database into quarantine or remove it from it.		
Input arguments	Name	Type	Description
	`databaseName`	`STRING`	The name of the database to set the quarantine status of.
	`setStatus`	`BOOLEAN`	Whether or not to quarantine the database.
	`reason`	`STRING`	The reason to quarantine the database.
Return arguments	Name	Type	Description
	`databaseName`	`STRING`	The name of the database.
	`quarantined`	`BOOLEAN`	Whether or not the database is quarantined.
	`result`	`STRING`	Details about the outcome of the procedure.
Mode	DBMS		

dbms.upgrade()

Admin Only Deprecated in 5.9

Details

Syntax	`dbms.upgrade() :: (status, upgradeResult)`		
Description	Upgrade the system database schema if it is not the current schema.		
Return arguments	Name	Type	Description
	`status`	`STRING`	The upgrade status of the system database.
	`upgradeResult`	`STRING`	Information about the upgrade outcome.
Mode	WRITE		

> **Note:**
>
> This procedure is not considered safe to run from multiple threads. It is therefore not supported by the parallel runtime (introduced in Neo4j 5.13). For more information, see the Cypher Manual → Parallel runtime.

dbms.upgradeStatus()

Admin Only Deprecated in 5.9

Details

Syntax	`dbms.upgradeStatus() :: (status, description, resolution)`		
Description	Report the current status of the system database sub-graph schema.		
Return arguments	Name	Type	Description
	`status`	`STRING`	The upgrade status of the system database.
	`description`	`STRING`	Information describing the upgrade status.
	`resolution`	`STRING`	Information about the steps necessary to upgrade.
Mode	READ		

> **Note:**
>
> This procedure is not considered safe to run from multiple threads. It is therefore not

supported by the parallel runtime (introduced in Neo4j 5.13). For more information, see the Cypher Manual → Parallel runtime.

GenAI and vectors

For more information, see:

- Cypher Manual → Vector indexes
- Cypher Manual → Vector functions
- GenAI plugin
- GenAI plugin → Functions
- GenAI documentation → Embeddings & Vector Indexes Tutorial

db.create.setNodeVectorProperty()

Introduced in 5.13 | Beta until 5.26

Details

Syntax	`db.create.setNodeVectorProperty(node, key, vector)`		
Description	Set a vector property on a given node in a more space efficient representation than Cypher's `SET`.		
Input arguments	**Name**	**Type**	**Description**
	`node`	`NODE`	The node on which the new property will be stored.
	`key`	`STRING`	The name of the new property.
	`vector`	`ANY`	The object containing the embedding.
Mode	WRITE		

Known issue

Note:

Procedure signatures from `SHOW PROCEDURES` renders the vector arguments with a type of `ANY` rather than the semantically correct type of `LIST<INTEGER | FLOAT>`. The types are still enforced as `LIST<INTEGER | FLOAT>`.

db.create.setRelationshipVectorProperty()

Introduced in 5.18 | Beta until 5.26

Details

Syntax	`db.create.setRelationshipVectorProperty(relationship, key, vector)`		
Description	Set a vector property on a given relationship in a more space efficient representation than Cypher's `SET`.		
Input arguments	Name	Type	Description
	`relationship`	`RELATIONSHIP`	The relationship on which the new property will be stored.
	`key`	`STRING`	The name of the new property.
	`vector`	`ANY`	The object containing the embedding.
Mode	WRITE		

Known issue

Note:

Procedure signatures from `SHOW PROCEDURES` renders the vector arguments with a type of `ANY` rather than the semantically correct type of `LIST<INTEGER | FLOAT>`. The types are still enforced as `LIST<INTEGER | FLOAT>`.

db.create.setVectorProperty()

Introduced in 5.11 | Beta | Deprecated in 5.13

Details

Syntax	`db.create.setVectorProperty(node, key, vector) :: (node)`		
Description	Set a vector property on a given node in a more space efficient representation than Cypher's SET.		
Input arguments	Name	Type	Description
	`node`	`NODE`	The node on which the new property will be stored.
	`key`	`STRING`	The name of the new property.
	`vector`	`ANY`	The object containing the embedding.

Return arguments	Name	Type	Description
	`node`	`NODE`	The node on which the vector property was set.
Mode	WRITE		
Replaced by	`db.create.setNodeVectorProperty()` and `db.create.setRelationshipVectorProperty()`		

Known issue

Note:

Procedure signatures from `SHOW PROCEDURES` renders the vector arguments with a type of `ANY` rather than the semantically correct type of `LIST<INTEGER | FLOAT>`. The types are still enforced as `LIST<INTEGER | FLOAT>`.

db.index.vector.createNodeIndex()

Introduced in 5.11 | Beta until 5.13 | Deprecated in 5.26

Details

Syntax	`db.index.vector.createNodeIndex(indexName, label, propertyKey, vectorDimension, vectorSimilarityFunction)`		
Description	Create a named node vector index for the specified label and property with the given vector dimensionality using either the EUCLIDEAN or COSINE similarity function. Both similarity functions are case-insensitive. Use the `db.index.vector.queryNodes` procedure to query the named index.		
Input arguments	Name	Type	Description
	`indexName`	`STRING`	indexName :: STRING
	`label`	`STRING`	label :: STRING
	`propertyKey`	`STRING`	propertyKey :: STRING
	`vectorDimension`	`INTEGER`	vectorDimension :: INTEGER
	`vectorSimilarityFunction`	`STRING`	vectorSimilarityFunction :: STRING
Mode	SCHEMA		

Note:

As of Neo4j 5.15, vector indexes can be created with the Cypher Command `CREATE VECTOR INDEX`. For more information, see the Cypher Manual → Create a vector index.

db.index.vector.queryNodes()

Introduced in 5.11 Beta until 5.13

Details

Syntax	`db.index.vector.queryNodes(indexName, numberOfNearestNeighbours, query) :: (node, score)`		
Description	Query the given node vector index. Returns requested number of nearest neighbors to the provided query vector, and their similarity score to that query vector, based on the configured similarity function for the index. The similarity score is a value between [0, 1]; where 0 indicates least similar, 1 most similar.		
Input arguments	**Name**	**Type**	**Description**
	`indexName`	`STRING`	The name of the vector index.
	`numberOfNearestNeighbours`	`INTEGER`	The size of the vector neighbourhood.
	`query`	`ANY`	The object to find approximate matches for.
Return arguments	**Name**	**Type**	**Description**
	`node`	`NODE`	A node which contains a vector property similar to the query object.
	`score`	`FLOAT`	The score measuring how similar the node property is to the query object.
Mode	READ		

db.index.vector.queryRelationships()

Introduced in 5.18

Details

Syntax	`db.index.vector.queryRelationships(indexName, numberOfNearestNeighbours, query) :: (relationship, score)`
Description	Query the given relationship vector index. Returns requested number of nearest neighbors to the provided query vector, and their similarity score to that query vector, based on the configured similarity function for the index. The similarity score is a value between [0, 1]; where 0 indicates least similar, 1 most similar.

Input arguments	Name	Type	Description
	`indexName`	`STRING`	The name of the vector index.
	`numberOfNearestNeighbours`	`INTEGER`	The size of the vector neighbourhood.
	`query`	`ANY`	The object to find approximate matches for.
Return arguments	Name	Type	Description
	`relationship`	`RELATIONSHIP`	A relationship which contains a vector property similar to the query object.
	`score`	`FLOAT`	The score measuring how similar the relationship property is to the query object.
Mode	READ		

genai.vector.encodeBatch()

Introduced in 5.17

Details

Syntax	`genai.vector.encodeBatch(resources, provider, configuration) :: (index, resource, vector)`
Description	Encode a given batch of resources as vectors using the named provider. For each element in the given resource LIST this returns: • the corresponding 'index' within that LIST, • the original 'resource' element itself, • and the encoded 'vector'.

Input arguments	Name	Type	Description
	`resources`	`LIST<STRING>`	The object to transform into an embedding.
	`provider`	`STRING`	The GenAI provider to use.
	`configuration`	`ANY`	The provider specific settings.

Return arguments	Name	Type	Description
	`index`	`INTEGER`	The index of the corresponding element in the input list.
	`resource`	`STRING`	The name of the input resource.
	`vector`	`ANY`	The generated vector embedding for the resource.
Mode	DEFAULT		

For more information, see the Cypher Manual → Generating a batch of embeddings.

> *Known issue*
>
> **Note:**
>
> Procedure signatures from `SHOW PROCEDURES` renders the vector arguments with a type of `ANY` rather than the semantically correct type of `LIST<INTEGER | FLOAT>`. The types are still enforced as `LIST<INTEGER | FLOAT>`.

genai.vector.listEncodingProviders()

Introduced in 5.19

Details

Syntax	`genai.vector.listEncodingProviders() :: (name, requiredConfigType, otionalConfigType, defaultConfig)`		
Description	Lists the available vector embedding providers.		
Return arguments	Name	Type	Description
	`name`	`STRING`	The name of the GenAI provider.
	`requiredConfigType`	`STRING`	The signature of the required config map.
	`optionalConfigType`	`STRING`	The signature of the optional config map.
	`defaultConfig`	`MAP`	The default values for the GenAI provider.
Mode	DEFAULT		

Index management

For more information, see:

- Index configuration
- Cypher Manual → Search performance indexes
- Cypher Manual → Full-text indexes

db.awaitIndex()

Details

Syntax	`db.awaitIndex(indexName [, timeOutSeconds ])`		
Description	Wait for an index to come online (for example: CALL db.awaitIndex("MyIndex", 300)).		
Input arguments	Name	Type	Description
	`indexName`	`STRING`	The name of the awaited index.
	`timeOutSeconds`	`INTEGER`	The maximum time to wait in seconds.
Mode	READ		

> **Note:**
>
> This procedure is not considered safe to run from multiple threads. It is therefore not supported by the parallel runtime (introduced in Neo4j 5.13). For more information, see the Cypher Manual → Parallel runtime.

db.awaitIndexes()

Details

Syntax	`db.awaitIndexes([ timeOutSeconds ])`		
Description	Wait for all indexes to come online (for example: CALL db.awaitIndexes(300)).		
Input arguments	Name	Type	Description
	`timeOutSeconds`	`INTEGER`	The maximum time to wait in seconds.
Mode	READ		

> **Note:**

This procedure is not considered safe to run from multiple threads. It is therefore not supported by the parallel runtime (introduced in Neo4j 5.13). For more information, see the Cypher Manual → Parallel runtime.

db.index.fulltext.awaitEventuallyConsistentIndexRefresh()

Details

Syntax	`db.index.fulltext.awaitEventuallyConsistentIndexRefresh()`
Description	Wait for the updates from recently committed transactions to be applied to any eventually-consistent full-text indexes.
Mode	READ

db.index.fulltext.listAvailableAnalyzers()

Details

Syntax	`db.index.fulltext.listAvailableAnalyzers() :: (analyzer, description, stopwords)`		
Description	List the available analyzers that the full-text indexes can be configured with.		
Return arguments	Name	Type	Description
	`analyzer`	`STRING`	The name of the analyzer.
	`description`	`STRING`	The description of the analyzer.
	`stopwords`	`LIST<STRING>`	The stopwords used by the analyzer to tokenize strings.
Mode	READ		

db.index.fulltext.queryNodes()

Details

Syntax	`db.index.fulltext.queryNodes(indexName, queryString [, options ]) :: (node, score)`
Description	Query the given full-text index. Returns the matching nodes and their Lucene query score, ordered by score. Valid *key*: *value* pairs for the `options` map are: • 'skip' — to skip the top N results. • 'limit' — to limit the number of results returned. • 'analyzer' — to use the specified analyzer as a search analyzer for this query. The `options` map and any of the keys are optional. An example of the `options` map: `{skip: 30, limit: 10, analyzer: 'whitespace'}`

Input arguments	Name	Type	Description
	`indexName`	`STRING`	The name of the full-text index.
	`queryString`	`STRING`	The string to find approximate matches for.
	`options`	`MAP`	{skip :: INTEGER, limit :: INTEGER, analyzer :: STRING}
Return arguments	Name	Type	Description
	`node`	`NODE`	A node which contains a property similar to the query string.
	`score`	`FLOAT`	The score measuring how similar the node property is to the query string.
Mode	READ		

db.index.fulltext.queryRelationships()

Details

Syntax	`db.index.fulltext.queryRelationships(indexName, queryString [, options ]) :: (relationship, score)`
Description	Query the given full-text index. Returns the matching relationships and their Lucene query score, ordered by score. Valid *key: value* pairs for the `options` map are: • 'skip' — to skip the top N results. • 'limit' — to limit the number of results returned. • 'analyzer' — to use the specified analyzer as a search analyzer for this query. The `options` map and any of the keys are optional. An example of the `options` map: `{skip: 30, limit: 10, analyzer: 'whitespace'}`

Input arguments	Name	Type	Description
	`indexName`	`STRING`	The name of the full-text index.
	`queryString`	`STRING`	The string to find approximate matches for.
	`options`	`MAP`	{skip :: INTEGER, limit :: INTEGER, analyzer :: STRING}

Return arguments	Name	Type	Description
	`relationship`	`RELATIONSHIP`	A relationship which contains a property similar to the query string.
	`score`	`FLOAT`	The score measuring how similar the relationship property is to the query string.
Mode	READ		

db.resampleIndex()

Details

Syntax	`db.resampleIndex(indexName)`		
Description	Schedule resampling of an index (for example: CALL db.resampleIndex("MyIndex")).		
Input arguments	Name	Type	Description
	`indexName`	`STRING`	The name of the index.
Mode	READ		

> **Note:**
>
> This procedure is not considered safe to run from multiple threads. It is therefore not supported by the parallel runtime (introduced in Neo4j 5.13). For more information, see the Cypher Manual → Parallel runtime.

db.resampleOutdatedIndexes()

Details

Syntax	`db.resampleOutdatedIndexes()`
Description	Schedule resampling of all outdated indexes.
Mode	READ

> **Note:**
>
> This procedure is not considered safe to run from multiple threads. It is therefore not supported by the parallel runtime (introduced in Neo4j 5.13). For more information, see the Cypher Manual → Parallel runtime.

Metrics

dbms.queryJmx()

Details

Syntax	`dbms.queryJmx(query) :: (name, description, attributes)`		
Description	Query JMX management data by domain and name. For instance, use : to find all JMX beans.		
Input arguments	Name	Type	Description
	`query`	`STRING`	A query for MBeans on this MBeanServer (e.g. ':,name=**neo4j**' for all metrics in neo4j database).
Return arguments	Name	Type	Description
	`name`	`STRING`	The name of the metric.
	`description`	`STRING`	The description of the metric.
	`attributes`	`MAP`	A collection with the attributes (values) of that metric.
Mode	DBMS		

Schema and metadata

db.schema.nodeTypeProperties()

Details

Syntax	`db.schema.nodeTypeProperties() :: (nodeType, nodeLabels, propertyName, propertyTypes, mandatory)`
Description	Show the derived property schema of the nodes in tabular form.

Return arguments	Name	Type	Description
	`nodeType`	`STRING`	A name generated from the labels on the node.
	`nodeLabels`	`LIST<STRING>`	A list containing the labels on a category of node.
	`propertyName`	`STRING`	A property key on a category of node.
	`propertyTypes`	`LIST<STRING>`	All types of a property belonging to a node category.
	`mandatory`	`BOOLEAN`	Whether or not the property is present on all nodes belonging to a node category.
Mode	READ		

Note:

This procedure is not considered safe to run from multiple threads. It is therefore not supported by the parallel runtime (introduced in Neo4j 5.13). For more information, see the Cypher Manual → Parallel runtime.

db.schema.relTypeProperties()

Details

Syntax	`db.schema.relTypeProperties() :: (relType, propertyName, propertyTypes, mandatory)`		
Description	Show the derived property schema of the relationships in tabular form.		
Return arguments	Name	Type	Description
	`relType`	`STRING`	A name generated from the type on the relationship.
	`propertyName`	`STRING`	A property key on a category of relationship.
	`propertyTypes`	`LIST<STRING>`	All types of a property belonging to a relationship category.
	`mandatory`	`BOOLEAN`	Whether or not the property is present on all relationships belonging to a relationship category.

Mode	READ

> **Note:**
>
> This procedure is not considered safe to run from multiple threads. It is therefore not supported by the parallel runtime (introduced in Neo4j 5.13). For more information, see the Cypher Manual → Parallel runtime.

db.schema.visualization()

Details

Syntax	`db.schema.visualization() :: (nodes, relationships)`		
Description	Visualizes the schema of the data based on available statistics. A new node is returned for each label. The properties represented on the node include: `name` (label name), `indexes` (list of indexes), and `constraints` (list of constraints). A relationship of a given type is returned for all possible combinations of start and end nodes. The properties represented on the relationship include: `name` (type name). Note that this may include additional relationships that do not exist in the data due to the information available in the count store.		
Return arguments	Name	Type	Description
	`nodes`	`LIST<NODE>`	A list of virtual nodes representing each label in the database.
	`relationships`	`LIST<RELATIONSHIP>`	A list of virtual relationships representing all combinations between start and end nodes in the database.
Mode	READ		

> **Note:**
>
> This procedure is not considered safe to run from multiple threads. It is therefore not supported by the parallel runtime (introduced in Neo4j 5.13). For more information, see the Cypher Manual → Parallel runtime.

db.createLabel()

Details

Syntax	`db.createLabel(newLabel)`
Description	Create a label

Input arguments	Name	Type	Description
	`newLabel`	`STRING`	Label name.
Mode	WRITE		

db.createProperty()

Details

Syntax	`db.createProperty(newProperty)`		
Description	Create a Property		
Input arguments	Name	Type	Description
	`newProperty`	`STRING`	Property name.
Mode	WRITE		

db.createRelationshipType

Details

Syntax	`db.createRelationshipType(newRelationshipType)`		
Description	Create a RelationshipType		
Input arguments	Name	Type	Description
	`newRelationshipType`	`STRING`	Relationship type name.
Mode	WRITE		

db.labels()

Details

Syntax	`db.labels() :: (label)`		
Description	List all labels attached to nodes within a database according to the user's access rights. The procedure returns empty results if the user is not authorized to view those labels.		
Return arguments	Name	Type	Description
	`label`	`STRING`	A label within the database.
Mode	READ		

db.propertyKeys()

Details

Syntax	`db.propertyKeys() :: (propertyKey)`		
Description	List all property keys in the database.		
Return arguments	Name	Type	Description
	`propertyKey`	`STRING`	A property key in the database.
Mode	READ		

> **Note:**
>
> This procedure is not considered safe to run from multiple threads. It is therefore not supported by the parallel runtime (introduced in Neo4j 5.13). For more information, see the Cypher Manual → Parallel runtime.

db.relationshipTypes()

Details

Syntax	`db.relationshipTypes() :: (relationshipType)`		
Description	List all types attached to relationships within a database according to the user's access rights. The procedure returns empty results if the user is not authorized to view those relationship types.		
Return arguments	Name	Type	Description
	`relationshipType`	`STRING`	A relationship type in the database.
Mode	READ		

> **Note:**
>
> This procedure is not considered safe to run from multiple threads. It is therefore not supported by the parallel runtime (introduced in Neo4j 5.13). For more information, see the Cypher Manual → Parallel runtime.

Statistics and query planning

For more information, see Statistics and execution plans

db.clearQueryCaches()

Admin Only

Details

Syntax	`db.clearQueryCaches() :: (value)`		
Description	Clears all query caches.		
Return arguments	Name	Type	Description
	`value`	`STRING`	Information about the number of cleared query caches.
Mode	DBMS		

db.prepareForReplanning()

Admin Only

Details

Syntax	`db.prepareForReplanning([ timeOutSeconds ])`		
Description	Triggers an index resample and waits for it to complete, and after that clears query caches. After this procedure has finished queries will be planned using the latest database statistics.		
Input arguments	Name	Type	Description
	`timeOutSeconds`	`INTEGER`	The maximum time to wait in seconds.
Mode	READ		

> **Note:**
>
> This procedure is not considered safe to run from multiple threads. It is therefore not supported by the parallel runtime (introduced in Neo4j 5.13). For more information, see the Cypher Manual → Parallel runtime.

db.stats.clear

Admin Only

Details

Syntax	`db.stats.clear(section) :: (section, success, message)`		
Description	Clear collected data of a given data section. Valid sections are 'QUERIES'		
Input arguments	Name	Type	Description
	`section`	`STRING`	The section to clear. The only available section is: 'QUERIES'.

Return arguments	Name	Type	Description
	`section`	`STRING`	The section cleared.
	`success`	`BOOLEAN`	Whether the section was successfully cleared.
	`message`	`STRING`	Details about the outcome of the procedure.
Mode	READ		

db.stats.collect()

Admin Only

Details

Syntax	`db.stats.collect(section [, config ]) :: (section, success, message)`		
Description	Start data collection of a given data section. Valid sections are 'QUERIES'		
Input arguments	Name	Type	Description
	`section`	`STRING`	The section to collect. The only available section is: 'QUERIES'.
	`config`	`MAP`	{durationSeconds = -1 INTEGER}
Return arguments	Name	Type	Description
	`section`	`STRING`	The section collected.
	`success`	`BOOLEAN`	Whether the section was successfully collected.
	`message`	`STRING`	Details about the outcome of the procedure.
Mode	READ		

db.stats.retrieve()

Admin Only

Details

Syntax	`db.stats.retrieve(section [, config ]) :: (section, data)`

Description	Retrieve statistical data about the current database. Valid sections are 'GRAPH COUNTS', 'TOKENS', 'QUERIES', 'META'		
Input arguments	Name	Type	Description
	`section`	`STRING`	A section of stats to retrieve: ('GRAPH COUNTS', 'TOKENS', 'QUERIES', 'META').
	`config`	`MAP`	{maxInvocations = 100 INTEGER}
Return arguments	Name	Type	Description
	`section`	`STRING`	The section retrieved.
	`data`	`MAP`	Data pertaining to the retrieved statistics.
Mode	READ		

db.stats.retrieveAllAnonymized()

Admin Only

Details

Syntax	`db.stats.retrieveAllAnonymized(graphToken [, config ]) :: (section, data)`		
Description	Retrieve all available statistical data about the current database, in an anonymized form.		
Input arguments	Name	Type	Description
	`graphToken`	`STRING`	The name of the graph token.
	`config`	`MAP`	{maxInvocations = 100 INTEGER}
Return arguments	Name	Type	Description
	`section`	`STRING`	The section retrieved.
	`data`	`MAP`	Data pertaining to the retrieved statistics.
Mode	READ		

db.stats.status()

Admin Only

Details

Syntax	`db.stats.status() :: (section, status, data)`		
Description	Retrieve the status of all available collector daemons, for this database.		
Return arguments	Name	Type	Description
	`section`	`STRING`	String with the message "QUERIES".
	`status`	`STRING`	The status of the QueryCollector: "idle" or "collecting".
	`data`	`MAP`	data :: MAP
Mode	READ		

db.stats.stop()

Admin Only

Details

Syntax	`db.stats.stop(section) :: (section, success, message)`		
Description	Stop data collection of a given data section. Valid sections are 'QUERIES'		
Input arguments	Name	Type	Description
	`section`	`STRING`	The section to stop. The only available section is: 'QUERIES'.
Return arguments	Name	Type	Description
	`section`	`STRING`	The stopped section.
	`success`	`BOOLEAN`	Whether the section was successfully stopped.
	`message`	`STRING`	Details about the outcome of the procedure.
Mode	READ		

Transaction management

tx.getMetaData()

Details

Syntax	`tx.getMetaData() :: (metadata)`		
Description	Provides attached transaction metadata.		
Return arguments	Name	Type	Description
	`metadata`	`MAP`	Metadata about the transaction.
Mode	DBMS		

tx.setMetaData()

Details

Syntax	`tx.setMetaData(data)`		
Description	Attaches a map of data to the transaction. The data will be printed when listing queries, and inserted into the query log.		
Input arguments	Name	Type	Description
	`data`	`MAP`	Metadata to attach to the transaction.
Mode	DBMS		

List of deprecated procedures

Neo4j 5 contains several deprecated procedures. These procedures have been replaced either by Cypher commands or different procedures. The procedures deprecated in Neo4j 5 will be removed in the next major release of Neo4j.

▼ *See all deprecated procedures*

Name	Community Edition	Enterprise Edition	Comment
`cdc.current()`	No	Yes	Introduced in 5.13 Beta Deprecated in 5.17 Replaced by: `db.cdc.current()`
`cdc.earliest()`	No	Yes	Introduced in 5.13 Beta Deprecated in 5.17 Replaced by: `db.cdc.earliest()`
`cdc.query()`	No	Yes	Introduced in 5.13 Beta Admin Only Deprecated in 5.17 Replaced by: `db.cdc.query()`
`db.create.setVectorProperty()`	Yes	Yes	Introduced in 5.11 Beta Deprecated in 5.13 Replaced by: `db.create.setNodeVectorProperty()`

Name	Community Edition	Enterprise Edition	Comment
`dbms.cluster.readReplicaToggle()`	No	Yes	Admin Only Deprecated in 5.6. Replaced by: `dbms.cluster.secondaryReplicationDisable()`.
`dbms.cluster.routing.getRoutingTable()`	Yes	Yes	Deprecated in 5.21. Replaced by: `dbms.routing.getRoutingTable()`.
`dbms.cluster.uncordonServer()`	No	Yes	Deprecated in 5.23. Before Neo4j 5.23, the procedure can be run only with the `Admin` privileges. Replaced by `ENABLE SERVER`.
`dbms.setDatabaseAllocator()`	No	Yes	Admin Only Deprecated in 5.23
`dbms.upgrade()`	Yes	Yes	Admin Only Deprecated in 5.9
`dbms.upgradeStatus()`	Yes	Yes	Admin Only Deprecated in 5.9

List of removed procedures

Several procedures were removed with the release of Neo4j. They were functionally replaced by Cypher commands or different procedures.

▼ *See all procedures removed in Neo4j 5.0 and their replacements*

Name	Community Edition	Enterprise Edition	Replaced by
`db.constraints()`	Yes	Yes	`SHOW CONSTRAINTS`
`db.createIndex()`	Yes	Yes	`CREATE INDEX`
`db.createNodeKey()`	No	Yes	`CREATE CONSTRAINT … IS NODE KEY`
`db.createUniquePropertyConstraint()`	Yes	Yes	`CREATE CONSTRAINT … IS UNIQUE`
`db.indexes()`	Yes	Yes	`SHOW INDEXES`
`db.indexDetails()`	Yes	Yes	`SHOW INDEXES YIELD*`
`db.index.fulltext.createNodeIndex()`	Yes	Yes	`CREATE FULLTEXT INDEX …`
`db.index.fulltext.createRelationshipIndex()`	Yes	Yes	`CREATE FULLTEXT INDEX …`
`db.index.fulltext.drop()`	Yes	Yes	`DROP INDEX …`

Name	Community Edition	Enterprise Edition	Replaced by
`db.schemaStatements()`	Yes	Yes	`SHOW INDEXES YIELD *` and `SHOW CONSTRAINTS YIELD *`
`dbms.cluster.overview()`	No	Yes	`SHOW SERVERS`
`dbms.cluster.quarantineDatabase()`	No	Yes	`dbms.quarantineDatabase()`
`dbms.cluster.role()`	No	Yes	`SHOW DATABASES`
`dbms.cluster.setDefaultDatabase()`	No	Yes	`dbms.setDefaultDatabase`
`dbms.database.state()`	Yes	Yes	`SHOW DATABASES`
`dbms.functions()`	Yes	Yes	`SHOW FUNCTIONS`
`dbms.killQueries()`	Yes	Yes	`TERMINATE TRANSACTIONS`
`dbms.killQuery()`	Yes	Yes	`TERMINATE TRANSACTIONS`
`dbms.killTransaction()`	Yes	Yes	`TERMINATE TRANSACTIONS`
`dbms.killTransactions()`	Yes	Yes	`TERMINATE TRANSACTIONS`
`dbms.listQueries()`	Yes	Yes	`SHOW TRANSACTIONS`
`dbms.listTransactions()`	Yes	Yes	`SHOW TRANSACTIONS`
`dbms.procedures()`	No	Yes	`SHOW PROCEDURES`
`dbms.security.activateUser()`	No	Yes	`ALTER USER`
`dbms.security.addRoleToUser()`	No	Yes	`GRANT ROLE TO USER`
`dbms.security.changePassword()`	Yes	Yes	`ALTER CURRENT USER SET PASSWORD`
`dbms.security.changeUserPassword()`	No	Yes	`ALTER USER`
`dbms.security.createRole()`	No	Yes	`CREATE ROLE`
`dbms.security.createUser()`	Yes	Yes	`CREATE USER`
`dbms.security.deleteRole()`	No	Yes	`DROP ROLE`
`dbms.security.deleteUser()`	Yes	Yes	`DROP USER`
`dbms.security.listRoles()`	Yes	Yes	`SHOW ROLES`
`dbms.security.listRolesForUser()`	No	Yes	`SHOW USERS`
`dbms.security.listUsers()`	Yes	Yes	`SHOW USERS`
`dbms.security.listUsersForRole()`	No	Yes	`SHOW ROLES WITH USERS`
`dbms.security.removeRoleFromUser()`	No	Yes	`REVOKE ROLE FROM USER`

Name	Community Edition	Enterprise Edition	Replaced by
`dbms.security.suspendUser()`	No	Yes	`ALTER USER`

Appendix A: Tutorials

The following step-by-step tutorials cover common operational tasks or otherwise exemplify working with Neo4j:

- Importing data — This tutorial provides detailed examples to illustrate the capabilities of importing data from CSV files with the command `neo4j-admin database import`.
- Setting up and using a composite database — This tutorial walks through the basics of setting up and using Composite databases.
- Fine-grained access control — This tutorial presents an example that illustrates various aspects of security and fine-grained access control.
- Configuring Neo4j Single Sign-On (SSO) — Examples and solutions to common problems when configuring SSO.
- Deploying a Neo4j cluster in a Docker container — This tutorial walks through setting up a Neo4j cluster on your local computer for testing purposes.

Importing data

This tutorial provides detailed examples to illustrate the capabilities of importing data from CSV files with the command `neo4j-admin database import`. The import command is used for loading large amounts of data from CSV files and supports full and incremental import into a running or stopped Neo4j DBMS.

Important:

The `neo4j-admin database import` command does not create a database, the command only imports data and makes it available for the database. The database must not exist before the `neo4j-admin database import` command has been executed, and the database should be created afterward. The command will exit with an error message if the database already exists.

Relationships are created by connecting node IDs, each node should have a unique ID to be able to be referenced when creating relationships between nodes. In the following examples, the node IDs are stored as properties on the nodes. If you do not want the IDs to persist as properties after the import completes, then do not specify a property name in the `:ID` field.

The examples show how to import data in a standalone Neo4j DBMS. They use:

- The Neo4j tarball (Unix console application).
- `$NEO4J_HOME` as the current working directory.
- The default database `neo4j`.
- The *import* directory of the Neo4j installation to store all the CSV files. However, the CSV files can be located in any directory of your file system.
- UNIX-styled paths.

- The `neo4j-admin database import` command.

> **Tip:**
>
> **Handy tips:**
>
> - The details of a CSV file header format can be found at CSV header format.
> - To show available databases, use the Cypher query `SHOW DATABASES`.
> - To remove a database, use the Cypher query `DROP DATABASE database_name`.
> - To create a database, use the Cypher query `CREATE DATABASE database_name`.

Import a small data set

In this example, you will import a small data set containing nodes and relationships. The data set is split into three CSV files, where each file has a header row describing the data.

The data

The data set contains information about movies, actors, and roles. Data for movies and actors are stored as nodes and the roles are stored as relationships.

The files you want to import data from are:

- `movies.csv`
- `actors.csv`
- `roles.csv`

Each movie in `movies.csv` has an `ID`, a `title`, and a `year`, stored as **properties** in the node. All the nodes in `movies.csv` also have the **label** `Movie`. A node can have several labels, as you can see in `movies.csv` there are nodes that also have the label `Sequel`. The node labels are optional, they are very useful for grouping nodes into sets where all nodes that have a certain label belong to the same set.

movies.csv

```
movieId:ID,title,year:int,:LABEL
tt0133093,"The Matrix",1999,Movie
tt0234215,"The Matrix Reloaded",2003,Movie;Sequel
tt0242653,"The Matrix Revolutions",2003,Movie;Sequel
```

The actors' data in `actors.csv` consist of an `ID` and a `name`, stored as **properties** in the node. The ID in this case is a shorthand for the actor's name. All the nodes in `actors.csv` have the label `Actor`.

actors.csv

```
personId:ID,name,:LABEL
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
```

The roles data in `roles.csv` have only one **property**, `role`. Roles are represented by relationship data that connects actor nodes with movie nodes.

There are three mandatory fields for relationship data:

1. `:START_ID` — ID referring to a node.
2. `:END_ID` — ID referring to a node.
3. `:TYPE` — The relationship type.

To create a relationship between two nodes, the IDs defined in `actors.csv` and `movies.csv` are used for the `:START_ID` and `:END_ID` fields. You also need to provide a relationship type (in this case `ACTED_IN`) for the `:TYPE` field.

roles.csv

```
:START_ID,role,:END_ID,:TYPE
keanu,"Neo",tt0133093,ACTED_IN
keanu,"Neo",tt0234215,ACTED_IN
keanu,"Neo",tt0242653,ACTED_IN
laurence,"Morpheus",tt0133093,ACTED_IN
laurence,"Morpheus",tt0234215,ACTED_IN
laurence,"Morpheus",tt0242653,ACTED_IN
carrieanne,"Trinity",tt0133093,ACTED_IN
carrieanne,"Trinity",tt0234215,ACTED_IN
carrieanne,"Trinity",tt0242653,ACTED_IN
```

Importing the data

- Paths to node data are defined with the `--nodes` option.
- Paths to relationship data is defined with the `--relationships` option.

The call to `neo4j-admin database import` would look like this:

shell

```
bin/neo4j-admin database import full neo4j --nodes=import/movies.csv --nodes=import/actors.csv
--relationships=import/roles.csv
```

Query the data

To query the data. Start Neo4j.

Note:

The default username and password are `neo4j` and `neo4j`.

shell

```
bin/neo4j start
```

To query the imported data in the graph, try a simple Cypher query.

shell

```
bin/cypher-shell --database=neo4j "MATCH (n) RETURN count(n) as nodes"
```

Stop Neo4j.

shell

```
bin/neo4j stop
```

CSV file delimiters

You can customize the configuration options that the import tool uses (see Options) if your data does not fit the default format.

The details of a CSV file header format can be found at CSV header format.

The data

The following CSV files have:

- `--delimiter=";"`
- `--array-delimiter="U+007C"` (`U+007C` is the Unicode code point for the character `|`)
- `--quote="'"`

movies2.csv

```
movieId:ID;title;year:int;:LABEL
tt0133093;'The Matrix';1999;Movie
tt0234215;'The Matrix Reloaded';2003;Movie|Sequel
tt0242653;'The Matrix Revolutions';2003;Movie|Sequel
```

actors2.csv

```
personId:ID;name;:LABEL
keanu;'Keanu Reeves';Actor
laurence;'Laurence Fishburne';Actor
carrieanne;'Carrie-Anne Moss';Actor
```

roles2.csv

```
:START_ID;role;:END_ID;:TYPE
keanu;'Neo';tt0133093;ACTED_IN
keanu;'Neo';tt0234215;ACTED_IN
keanu;'Neo';tt0242653;ACTED_IN
laurence;'Morpheus';tt0133093;ACTED_IN
laurence;'Morpheus';tt0234215;ACTED_IN
laurence;'Morpheus';tt0242653;ACTED_IN
carrieanne;'Trinity';tt0133093;ACTED_IN
carrieanne;'Trinity';tt0234215;ACTED_IN
carrieanne;'Trinity';tt0242653;ACTED_IN
```

Importing the data

The call to `neo4j-admin database import` would look like this:

shell

```
bin/neo4j-admin database import full neo4j --delimiter=";" --array-delimiter="U+007C" --quote="'"
--nodes=import/movies2.csv --nodes=import/actors2.csv --relationships=import/roles2.csv
```

Using separate header files

When dealing with very large CSV files, it is more convenient to have the header in a separate file. This makes it easier to edit the header as you avoid having to open a huge data file just to change it. The header file must be specified before the rest of the files in each file group.

The import tool can also process single-file compressed archives, for example:

- `--nodes=import/nodes.csv.gz`
- `--relationships=import/relationships.zip`

The data

You will use the same data set as in the previous example but with the headers in separate files.

movies3-header.csv

```
movieId:ID,title,year:int,:LABEL
```

movies3.csv

```
tt0133093,"The Matrix",1999,Movie
tt0234215,"The Matrix Reloaded",2003,Movie;Sequel
tt0242653,"The Matrix Revolutions",2003,Movie;Sequel
```

actors3-header.csv

```
personId:ID,name,:LABEL
```

actors3.csv

```
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
```

roles3-header.csv

```
:START_ID,role,:END_ID,:TYPE
```

roles3.csv

```
keanu,"Neo",tt0133093,ACTED_IN
keanu,"Neo",tt0234215,ACTED_IN
keanu,"Neo",tt0242653,ACTED_IN
laurence,"Morpheus",tt0133093,ACTED_IN
laurence,"Morpheus",tt0234215,ACTED_IN
laurence,"Morpheus",tt0242653,ACTED_IN
```

```
carrieanne,"Trinity",tt0133093,ACTED_IN
carrieanne,"Trinity",tt0234215,ACTED_IN
carrieanne,"Trinity",tt0242653,ACTED_IN
```

Importing the data

The call to `neo4j-admin database import` would look as follows:

> **Note:**
>
> The header line for a file group, whether it is the first line of a file in the group or a dedicated header file, must be the *first* line in the file group.

shell

```
bin/neo4j-admin database import full neo4j --nodes=import/movies3-header.csv,import/movies3.csv
--nodes=import/actors3-header.csv,import/actors3.csv --relationships=import/roles3
-header.csv,import/roles3.csv
```

Multiple input files

In addition to using a separate header file, you can also provide multiple node or relationship files. Files within such an input group can be specified with multiple match strings, delimited by `,`, where each matched string can be either the exact file name or a regular expression matching one or more files. Multiple matching files will be sorted according to their characters and their natural number sort order for file names containing numbers.

The data

movies4-header.csv

```
movieId:ID,title,year:int,:LABEL
```

movies4-part1.csv

```
tt0133093,"The Matrix",1999,Movie
tt0234215,"The Matrix Reloaded",2003,Movie;Sequel
```

movies4-part2.csv

```
tt0242653,"The Matrix Revolutions",2003,Movie;Sequel
```

actors4-header.csv

```
personId:ID,name,:LABEL
```

actors4-part1.csv

```
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
```

actors4-part2.csv

```
carrieanne,"Carrie-Anne Moss",Actor
```

roles4-header.csv

```
:START_ID,role,:END_ID,:TYPE
```

roles4-part1.csv

```
keanu,"Neo",tt0133093,ACTED_IN
keanu,"Neo",tt0234215,ACTED_IN
keanu,"Neo",tt0242653,ACTED_IN
laurence,"Morpheus",tt0133093,ACTED_IN
laurence,"Morpheus",tt0234215,ACTED_IN
```

roles4-part2.csv

```
laurence,"Morpheus",tt0242653,ACTED_IN
carrieanne,"Trinity",tt0133093,ACTED_IN
carrieanne,"Trinity",tt0234215,ACTED_IN
carrieanne,"Trinity",tt0242653,ACTED_IN
```

Importing the data

The call to `neo4j-admin database import` would look like this:

shell

```
bin/neo4j-admin database import full neo4j --nodes=import/movies4-header.csv,import/movies4
-part1.csv,import/movies4-part2.csv --nodes=import/actors4-header.csv,import/actors4
-part1.csv,import/actors4-part2.csv --relationships=import/roles4-header.csv,import/roles4
-part1.csv,import/roles4-part2.csv
```

Regular expressions

File names can be specified using regular expressions when there are many data source files. Each file name that matches the regular expression will be included.

If using separate header files, for the import to work correctly, the header file must be the first in the file group. When using regular expressions to specify the input files, the list of files will be sorted according to the names of the files that match the expression. The matching is aware of the numbers inside the file names and will sort them accordingly, without the need for padding with zeros.

Example 176. Match order

For example, let's assume that you have the following files:

- `movies4-header.csv`
- `movies4-data1.csv`
- `movies4-data2.csv`
- `movies4-data12.csv`

If you use the regular expression `movies4.*`, the sorting will place the header file last and the import will fail. A better alternative would be to name the header file explicitly and use a regular expression that only matches the names of the data files. For example: `--nodes "import/movies4-header.csv,movies-data.*"` will accomplish this.

Importing the data using regular expressions, the call to `neo4j-admin database import` can be simplified to:

shell

```
bin/neo4j-admin database import full neo4j --nodes="import/movies4-header.csv,import/movies4-part.*"
--nodes="import/actors4-header.csv,import/actors4-part.*" --relationships="import/roles4
-header.csv,import/roles4-part.*"
```

Note:

The use of regular expressions should not be confused with file globbing.

The expression `.*` means: "zero or more occurrences of any character except line break". Therefore, the regular expression `movies4.*` will list all files starting with `movies4`. Conversely, with file globbing, `ls movies4.*` will list all files starting with `movies4.`.

Another important difference to pay attention to is the sorting order. The result of a regular expression matching will place the file `movies4-part2.csv` before the file `movies4-part12.csv`. If doing `ls movies4-part*` in a directory containing the above-listed files, the file `movies4-part12.csv` will be listed before the file `movies4-part2.csv`.

Using the same label for every node

If you want to use the same node label(s) for every node in your nodes file you can do this by specifying the appropriate value as an option to `neo4j-admin database import`. There is then no need to specify the `:LABEL` column in the header file and each row (node) will apply the specified labels from the command line option.

Example 177. Specify node labels option

```
--nodes=LabelOne:LabelTwo=import/example-header.csv,import/example-data1.csv
```

Note:

It is possible to apply both the label provided in the file and the one provided on the command line to the node.

The data

In this example, you want to have the label `Movie` on every node specified in `movies5a.csv`, and you put the labels `Movie` and `Sequel` on the nodes specified in `sequels5a.csv`.

movies5a.csv

```
movieId:ID,title,year:int
tt0133093,"The Matrix",1999
```

sequels5a.csv

```
movieId:ID,title,year:int
tt0234215,"The Matrix Reloaded",2003
tt0242653,"The Matrix Revolutions",2003
```

actors5a.csv

```
personId:ID,name
keanu,"Keanu Reeves"
laurence,"Laurence Fishburne"
carrieanne,"Carrie-Anne Moss"
```

roles5a.csv

```
:START_ID,role,:END_ID,:TYPE
keanu,"Neo",tt0133093,ACTED_IN
keanu,"Neo",tt0234215,ACTED_IN
keanu,"Neo",tt0242653,ACTED_IN
laurence,"Morpheus",tt0133093,ACTED_IN
laurence,"Morpheus",tt0234215,ACTED_IN
laurence,"Morpheus",tt0242653,ACTED_IN
carrieanne,"Trinity",tt0133093,ACTED_IN
carrieanne,"Trinity",tt0234215,ACTED_IN
carrieanne,"Trinity",tt0242653,ACTED_IN
```

Importing the data

The call to `neo4j-admin database import` would look like this:

shell

```
bin/neo4j-admin database import full neo4j --nodes=Movie=import/movies5a.csv
--nodes=Movie:Sequel=import/sequels5a.csv --nodes=Actor=import/actors5a.csv
--relationships=import/roles5a.csv
```

Using the same relationship type for every relationship

If you want to use the same relationship type for every relationship in your relationships file this can be done by specifying the appropriate value as an option to `neo4j-admin database import`.

Example 178. Specify relationship type option

```
--relationships=TYPE=import/example-header.csv,import/example-data1.csv
```

Note:

If you provide a relationship type both on the command line and in the relationships file, the one in the file will be applied.

The data

In this example, you want the relationship type `ACTED_IN` to be applied on every relationship specified in `roles5b.csv`.

movies5b.csv

```
movieId:ID,title,year:int,:LABEL
tt0133093,"The Matrix",1999,Movie
tt0234215,"The Matrix Reloaded",2003,Movie;Sequel
tt0242653,"The Matrix Revolutions",2003,Movie;Sequel
```

actors5b.csv

```
personId:ID,name,:LABEL
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
```

roles5b.csv

```
:START_ID,role,:END_ID
keanu,"Neo",tt0133093
keanu,"Neo",tt0234215
keanu,"Neo",tt0242653
laurence,"Morpheus",tt0133093
laurence,"Morpheus",tt0234215
laurence,"Morpheus",tt0242653
carrieanne,"Trinity",tt0133093
carrieanne,"Trinity",tt0234215
carrieanne,"Trinity",tt0242653
```

Importing the data

The call to `neo4j-admin database import` would look like this:

shell

```
bin/neo4j-admin database import full neo4j --nodes=import/movies5b.csv --nodes=import/actors5b.csv
--relationships=ACTED_IN=import/roles5b.csv
```

Properties

Nodes and relationships can have properties. The property type is specified in the CSV header row, see CSV header format.

The data

The following example creates a small graph containing one actor and one movie connected by one relationship.

There is a `roles` property on the relationship which contains an array of the characters played by the actor in a movie:

movies6.csv

```
movieId:ID,title,year:int,:LABEL
tt0099892,"Joe Versus the Volcano",1990,Movie
```

actors6.csv

```
personId:ID,name,:LABEL
meg,"Meg Ryan",Actor
```

roles6.csv

```
:START_ID,roles:string[],:END_ID,:TYPE
meg,"DeDe;Angelica Graynamore;Patricia Graynamore",tt0099892,ACTED_IN
```

Importing the data

The call to `neo4j-admin database import` would look like this:

shell

```
bin/neo4j-admin database import full neo4j --nodes=import/movies6.csv --nodes=import/actors6.csv
--relationships=import/roles6.csv
```

ID space

The import tool assumes that identifiers are unique across node files. This may not be the case for data sets that use sequential, auto-incremented, or otherwise colliding identifiers. Those data sets can define ID spaces where identifiers are unique within their respective ID space.

In cases where the node ID is unique only within files, using ID spaces is a way to ensure uniqueness across all node files. See Using ID spaces.

Each node processed by `neo4j-admin database import` must provide an ID if it is to be connected in any relationships. The node ID is used to find the start node and end node when creating a relationship.

Note:

From Neo4j 5.3, a node header can also contain multiple `ID` columns, where the relationship data references the composite value of all those columns. This also implies using `string` as `id-type`. For each `ID` column, you can specify to store its values as different node properties. However, the composite value cannot be stored as a node property.

Example 179. ID space

To define an ID space `Movie-ID` for `movieId:ID`, use the syntax `movieId:ID(Movie-ID)`.

The data

For example, if movies and people both use sequential identifiers, then you would define `Movie` and `Actor` ID spaces.

movies7.csv

```
movieId:ID(Movie-ID),title,year:int,:LABEL
1,"The Matrix",1999,Movie
2,"The Matrix Reloaded",2003,Movie;Sequel
3,"The Matrix Revolutions",2003,Movie;Sequel
```

actors7.csv

```
personId:ID(Actor-ID),name,:LABEL
1,"Keanu Reeves",Actor
2,"Laurence Fishburne",Actor
3,"Carrie-Anne Moss",Actor
```

You also need to reference the appropriate ID space in your relationships file so it knows which nodes to connect.

roles7.csv

```
:START_ID(Actor-ID),role,:END_ID(Movie-ID)
1,"Neo",1
1,"Neo",2
1,"Neo",3
2,"Morpheus",1
2,"Morpheus",2
2,"Morpheus",3
3,"Trinity",1
3,"Trinity",2
3,"Trinity",3
```

Importing the data

The call to `neo4j-admin database import` would look like this:

shell

```
bin/neo4j-admin database import full neo4j --nodes=import/movies7.csv --nodes=import/actors7.csv
--relationships=ACTED_IN=import/roles7.csv
```

Skip relationships referring to missing nodes

The import tool has no tolerance for bad entities (relationships or nodes) and will fail the import on the first bad entity. You can specify explicitly that you want it to ignore rows that contain bad entities.

There are two different types of bad input:

1. Bad relationships.
2. Bad nodes.

Relationships that refer to missing node IDs, either for `:START_ID` or `:END_ID` are considered bad relationships. Whether or not such relationships are skipped is controlled with the `--skip-bad-relationships` flag, which can have the values `true` or `false` or no value, which means `true`. The default

is `false`, which means that any bad relationship is considered an error and will fail the import. For more information, see the `--skip-bad-relationships` option.

The data

In the following example, there is a missing `emil` node referenced in the roles file.

movies8a.csv

```
movieId:ID,title,year:int,:LABEL
tt0133093,"The Matrix",1999,Movie
tt0234215,"The Matrix Reloaded",2003,Movie;Sequel
tt0242653,"The Matrix Revolutions",2003,Movie;Sequel
```

actors8a.csv

```
personId:ID,name,:LABEL
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
```

roles8a.csv

```
:START_ID,role,:END_ID,:TYPE
keanu,"Neo",tt0133093,ACTED_IN
keanu,"Neo",tt0234215,ACTED_IN
keanu,"Neo",tt0242653,ACTED_IN
laurence,"Morpheus",tt0133093,ACTED_IN
laurence,"Morpheus",tt0234215,ACTED_IN
laurence,"Morpheus",tt0242653,ACTED_IN
carrieanne,"Trinity",tt0133093,ACTED_IN
carrieanne,"Trinity",tt0234215,ACTED_IN
carrieanne,"Trinity",tt0242653,ACTED_IN
emil,"Emil",tt0133093,ACTED_IN
```

Importing the data

The call to `neo4j-admin database import` would look like this:

shell

```
bin/neo4j-admin database import full neo4j --nodes=import/movies8a.csv --nodes=import/actors8a.csv
--relationships=import/roles8a.csv
```

Because there is a bad relationship in the input data, the import process will fail.

Let's see what happens if you append the `--skip-bad-relationships` flag:

shell

```
bin/neo4j-admin database import full neo4j --skip-bad-relationships --nodes=import/movies8a.csv
--nodes=import/actors8a.csv --relationships=import/roles8a.csv
```

The data files are successfully imported and the bad relationship is ignored. An entry is written to the `import.report` file.

ignore bad relationships

```
InputRelationship:
   source: roles8a.csv:11
   properties: [role, Emil]
   startNode: emil (global id space)
   endNode: tt0133093 (global id space)
   type: ACTED_IN
 referring to missing node emil
```

Skip nodes with the same ID

Nodes that specify `:ID`, which has already been specified within the ID space are considered bad nodes. Whether or not such nodes are skipped is controlled with `--skip-duplicate-nodes` flag which can have the values `true` or `false` or no value, which means `true`. The default is `false`, which means that any duplicate node is considered an error and will fail the import. For more information, see the `--skip-duplicate-nodes` option.

The data

In the following example there is a node ID, `laurence`, that is specified twice within the same ID space.

actors8b.csv

```
personId:ID,name,:LABEL
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
laurence,"Laurence Harvey",Actor
```

Importing the data

The call to `neo4j-admin database import` would look like this:

shell

```
bin/neo4j-admin database import full neo4j --database=neo4j --nodes=import/actors8b.csv
```

Because there is a bad node in the input data, the import process will fail.

Let's see what happens if you append the `--skip-duplicate-nodes` flag:

shell

```
bin/neo4j-admin database import full neo4j --skip-duplicate-nodes --nodes=import/actors8b.csv
```

The data files are successfully imported and the bad node is ignored. An entry is written to the `import.report` file.

ignore bad nodes

```
ID 'laurence' is defined more than once in global ID space, at least at actors8b.csv:3 and actors8b.csv:5
```

Setting up and using a composite database

Composite databases allow queries that access multiple graphs at once. This is a function that enables:

- **Data Federation**: the ability to access data available in distributed sources in the form of **disjoint graphs**.
- **Data Sharding**: the ability to access data available in distributed sources in the form of a **common graph partitioned on multiple databases**.

In this tutorial, you will learn how to:

- Model your data for Composite database use
- Create databases for the composite
- Import data to your databases
- Configure a Composite database
- Retrieve data with a single Cypher query

Model your data for Composite database use

The example data in this tutorial is based on the Northwind dataset, created by Microsoft. It contains the sales data of a fictitious small company called “Northwind Traders”. The data includes customers, products, customer orders, warehouse stock, shipping, suppliers, employees, and sales territories.

Note:

For more information on how Northwind (a relational dataset) is modeled into a graph, run `:play northwind` in Neo4j Browser to play the built-in guide Northwind Graph.

The Northwind graph model consists of the following data:

- Node labels
  - `:Product`
  - `:Category`
  - `:Supplier`
  - `:Order`
  - `:Customer`
- Relationship types
  - `:SUPPLIES`
  - `:PART_OF`
  - `:ORDERS`
  - `:PURCHASED`

Figure 37. The Northwind data model

In this scenario, assume that data privacy constraints require customers' data to be stored in their original region. For simplicity, there are two regions: the Americas (AME) and Europe (EU). The first step is to remodel the Northwind dataset, so that customer data can be separated from the Product catalog, which has no privacy constraints. You create two graphs: one for the Product catalog, which includes `:Product`, `:Category`, `:Supplier`, `:PART_OF`, `:SUPPLIES`, and one partitioned graph in two databases for the Customer orders in EU and AME, with `:Product`, `:Order`, `:Customer`, `:PURCHASED`, and `:ORDERS`.

Figure 38. The new data model

Data Federation

This way, the Product and Customer data are in two **disjoint graphs**, with different labels and relationship types. This is called *Data Federation*.

To query across them, you have to federate the graphs, because relationships cannot span across them. This is done by using a *proxy node* modeling pattern: nodes with the `:Product` label must be present in both federated domains.

In the Product catalog graph, nodes with the `:Product` label contain all the data related to a product, while in the Customer graphs, the same label is associated to a proxy node which only contains `productID`. The `productID` property allows you to link data across the graphs in this federation.

Figure 39. Data Federation

Data Sharding

Since the Customer data is for two regions (EU and AME), you have to partition it into two databases. The resulting two graphs have the same model (same labels, same relationship types), but different data. This is called *Data Sharding*.

Figure 40. Data Sharding

In general, there are a couple of main use cases that require sharding. The most common is scalability, i.e. different shards can be deployed on different servers, splitting the load on different resources. Another reason could be data regulations: different shards can be deployed on servers, residing in different locations, and managed independently.

Create databases for the composite

For this tutorial, you will create the following databases:

- `db0` for the Product catalog.
- `db1` for the EU customer data.
- `db2` for the AME customers.

1. Start the Neo4j DBMS.

```
bin/neo4j start
```

2. Check all available databases.

```
ls -al /data/databases/
```

```
total 0
drwxr-xr-x@  5 username  staff   160  9 Jun 12:53 .
drwxr-xr-x@  5 username  staff   160  9 Jun 12:53 ..
drwxr-xr-x  37 username  staff  1184  9 Jun 12:53 neo4j
-rw-r--r--   1 username  staff     0  9 Jun 12:53 store_lock
drwxr-xr-x  38 username  staff  1216  9 Jun 12:53 system
```

3. Connect to the Neo4j DBMS using `cypher-shell` with the default credentials and change the password when prompted:

```
bin/cypher-shell -u neo4j -p neo4j
```

```
Password change required
new password: ********
Connected to Neo4j 5 at neo4j://localhost:7687 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
```

> **Note:**
>
> For more information about the Cypher Shell command-line interface (CLI) and how to use it, see Cypher Shell.

4. Run the command `SHOW DATABASES` to list all available databases:

```
SHOW DATABASES;
```

```
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| name     | type       | aliases | access       | address          | role      | writer | requestedStatus | currentStatus | statusMessage | default | home  | constituents |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| "neo4j"  | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"        | "online"      | ""            | TRUE    | TRUE  | []           |
| "system" | "system"   | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"        | "online"      | ""            | FALSE   | FALSE | []           |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------+
2 rows available after 102 ms, consumed after another 11 ms
```

5. Run the command `CREATE DATABASE <database-name>` to create the databases:

```
CREATE DATABASE db0;
```

```
0 rows available after 137 ms, consumed after another 0 ms
```

```
CREATE DATABASE db1;
```

```
0 rows available after 141 ms, consumed after another 0 ms
```

```
CREATE DATABASE db2;
```

```
0 rows available after 135 ms, consumed after another 0 ms
```

6. Run the command `SHOW DATABASES` again to verify that the new databases have been created and are `online`:

```
SHOW DATABASES;
```

```
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| name     | type       | aliases | access       | address          | role      | writer | requestedStatus | currentStatus | statusMessage | default | home  | constituents |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| "db0"    | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"        | "online"      | ""            | FALSE   | FALSE | []           |
| "db1"    | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
```

```
| "online"      | ""           | FALSE   | FALSE | []          |
| "db2"    | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""           | FALSE   | FALSE | []          |
| "neo4j"  | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""           | TRUE    | TRUE  | []          |
| "system" | "system"   | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""           | FALSE   | FALSE | []          |
+-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------+
5 rows available after 8 ms, consumed after another 7 ms
```

Import data to your databases

You can use the command `LOAD CSV WITH HEADERS FROM` to import data to the databases.

Load the Product catalog in db0

1. Run the following Cypher query to change the active database to `db0`, and add the Product data:

```
:use db0;

LOAD CSV WITH HEADERS FROM "https://data.neo4j.com/northwind/products.csv" AS row
CREATE (n:Product)
SET n = row,
n.unitPrice = toFloat(row.unitPrice),
n.unitsInStock = toInteger(row.unitsInStock), n.unitsOnOrder = toInteger(row.unitsOnOrder),
n.reorderLevel = toInteger(row.reorderLevel), n.discontinued = (row.discontinued <> "0");

LOAD CSV WITH HEADERS FROM "https://data.neo4j.com/northwind/categories.csv" AS row
CREATE (n:Category)
SET n = row;

LOAD CSV WITH HEADERS FROM "https://data.neo4j.com/northwind/suppliers.csv" AS row
CREATE (n:Supplier)
SET n = row;

CREATE INDEX FOR (p:Product) ON (p.productID);
CREATE INDEX FOR (c:Category) ON (c.categoryID);
CREATE INDEX FOR (s:Supplier) ON (s.supplierID);

MATCH (p:Product),(c:Category)
WHERE p.categoryID = c.categoryID
CREATE (p)-[:PART_OF]->(c);

MATCH (p:Product),(s:Supplier)
WHERE p.supplierID = s.supplierID
CREATE (s)-[:SUPPLIES]->(p);
```

2. Press Enter.
3. Verify that the product data is loaded in `db0`:

```
MATCH (s:Supplier)-[:SUPPLIES]->(p:Product)-[:PART_OF]->(c:Category)
RETURN s.companyName AS Supplier, p.productName AS Product, c.categoryName AS Category
LIMIT 5;
```

```
+-----------------------------------------------------------------------+
| Supplier                   | Product                   | Category    |
+-----------------------------------------------------------------------+
| "Bigfoot Breweries"        | "Sasquatch Ale"           | "Beverages" |
| "Pavlova"                  | "Outback Lager"           | "Beverages" |
| "Bigfoot Breweries"        | "Laughing Lumberjack Lager" | "Beverages" |
| "Bigfoot Breweries"        | "Steeleye Stout"          | "Beverages" |
| "Aux joyeux ecclésiastiques" | "Côte de Blaye"           | "Beverages" |
+-----------------------------------------------------------------------+
```

```
5 rows available after 202 ms, consumed after another 5 ms
```

Load EU customers and related orders in db1

1. Run the following Cypher query to change the active database to `db1`, and add the EU customers and orders:

```
:use db1;

:param europe => ['Germany', 'UK', 'Sweden', 'France', 'Spain', 'Switzerland', 'Austria', 'Italy',
'Portugal', 'Ireland', 'Belgium', 'Norway', 'Denmark', 'Finland'];

LOAD CSV WITH HEADERS FROM "https://data.neo4j.com/northwind/customers.csv" AS row
WITH row
WHERE row.country IN $europe
CREATE (n:Customer)
SET n = row;

CREATE INDEX FOR (c:Customer) ON (c.customerID);

LOAD CSV WITH HEADERS FROM "https://data.neo4j.com/northwind/orders.csv" AS row
WITH row
MATCH (c:Customer)
WHERE row.customerID = c.customerID
CREATE (o:Order)
SET o = row;

CREATE INDEX FOR (o:Order) ON (o.orderID);

MATCH (c:Customer),(o:Order)
WHERE c.customerID = o.customerID
CREATE (c)-[:PURCHASED]->(o);

LOAD CSV WITH HEADERS FROM "https://data.neo4j.com/northwind/products.csv" AS row
CREATE (n:Product)
SET n.productID = row.productID;

CREATE INDEX FOR (p:Product) ON (p.productID);

LOAD CSV WITH HEADERS FROM "https://data.neo4j.com/northwind/order-details.csv" AS row
MATCH (p:Product), (o:Order)
WHERE p.productID = row.productID AND o.orderID = row.orderID
CREATE (o)-[details:ORDERS]->(p)
SET details = row, details.quantity = toInteger(row.quantity);
```

2. Press Enter.
3. Verify that the EU Customer orders data is loaded in `db1`:

```
MATCH (c:Customer)-[:PURCHASED]->(o:Order)-[:ORDERS]->(p:Product)
RETURN c.companyName AS Customer, c.country AS CustomerCountry, o.orderID AS Order, p.productID AS
Product
LIMIT 5;
```

```
+-------------------------------------------------------------+
| Customer              | CustomerCountry | Order   | Product |
+-------------------------------------------------------------+
| "Alfreds Futterkiste" | "Germany"       | "10692" | "63"    |
| "Alfreds Futterkiste" | "Germany"       | "10835" | "77"    |
| "Alfreds Futterkiste" | "Germany"       | "10835" | "59"    |
| "Alfreds Futterkiste" | "Germany"       | "10702" | "76"    |
| "Alfreds Futterkiste" | "Germany"       | "10702" | "3"     |
+-------------------------------------------------------------+

5 rows available after 47 ms, consumed after another 2 ms
```

Load AME customers and related orders in db2

1. Run the following Cypher query to change the active database to `db2` and add the AME customers and orders:

```
:use db2;

:param americas => ['Mexico', 'Canada', 'Argentina', 'Brazil', 'USA', 'Venezuela'];

LOAD CSV WITH HEADERS FROM "https://data.neo4j.com/northwind/customers.csv" AS row
WITH row
WHERE row.country IN $americas
CREATE (n:Customer)
SET n = row;

CREATE INDEX FOR (c:Customer) ON (c.customerID);

LOAD CSV WITH HEADERS FROM "https://data.neo4j.com/northwind/orders.csv" AS row
WITH row
MATCH (c:Customer)
WHERE row.customerID = c.customerID
CREATE (o:Order)
SET o = row;

CREATE INDEX FOR (o:Order) ON (o.orderID);

MATCH (c:Customer),(o:Order)
WHERE c.customerID = o.customerID
CREATE (c)-[:PURCHASED]->(o);

LOAD CSV WITH HEADERS FROM "https://data.neo4j.com/northwind/products.csv" AS row
CREATE (n:Product)
SET n.productID = row.productID;

CREATE INDEX FOR (p:Product) ON (p.productID);

LOAD CSV WITH HEADERS FROM "https://data.neo4j.com/northwind/order-details.csv" AS row
MATCH (p:Product), (o:Order)
WHERE p.productID = row.productID AND o.orderID = row.orderID
CREATE (o)-[details:ORDERS]->(p)
SET details = row,
details.quantity = toInteger(row.quantity);
```

2. Press Enter.
3. Verify that the AME Customer orders data is loaded in `db2`:

```
MATCH (c:Customer)-[:PURCHASED]->(o:Order)-[:ORDERS]->(p:Product)
RETURN c.companyName AS Customer, c.country AS CustomerCountry, o.orderID AS Order, p.productID AS
Product
LIMIT 5;
```

```
+-------------------------------------------------------------------------+
| Customer                          | CustomerCountry | Order   | Product |
+-------------------------------------------------------------------------+
| "Ana Trujillo Emparedados y helados" | "Mexico"        | "10759" | "32"    |
| "Ana Trujillo Emparedados y helados" | "Mexico"        | "10926" | "72"    |
| "Ana Trujillo Emparedados y helados" | "Mexico"        | "10926" | "13"    |
| "Ana Trujillo Emparedados y helados" | "Mexico"        | "10926" | "19"    |
| "Ana Trujillo Emparedados y helados" | "Mexico"        | "10926" | "11"    |
+-------------------------------------------------------------------------+

5 rows available after 42 ms, consumed after another 1 ms
```

Configure a Composite database

Set up a Composite database with the `CREATE COMPOSITE DATABASE` Cypher command and add local database aliases as constituents to the Composite database. In this example, the Composite database is called `compositenw`.

1. Run the command `CREATE COMPOSITE DATABASE <composite-database-name>` to create the Composite database:

```
CREATE COMPOSITE DATABASE compositenw;
```

```
0 rows available after 137 ms, consumed after another 0 ms
```

2. Run the command `CREATE ALIAS <composite-database-name>.<alias-name> FOR DATABASE <database-name>` to create the constituent database aliases:

```
CREATE ALIAS compositenw.product FOR DATABASE db0;
```

```
0 rows available after 101 ms, consumed after another 0 ms
```

```
CREATE ALIAS compositenw.customerEU FOR DATABASE db1;
```

```
0 rows available after 107 ms, consumed after another 0 ms
```

```
CREATE ALIAS compositenw.customerAME FOR DATABASE db2;
```

```
0 rows available after 98 ms, consumed after another 0 ms
```

> **Note:**
>
> The constituent database aliases in this tutorial are local database aliases (targeting databases in the same Neo4j DBMS), but they can just as well be remote database aliases (targeting databases in another Neo4j DBMS).

3. Run the command `SHOW DATABASES` to verify that the Composite database has been configured and is `online`:

```
SHOW DATABASES;
```

```
+-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------+
| name          | type        | aliases                    | access       | address          | role
| writer | requestedStatus | currentStatus | statusMessage | default | home  | constituents
|
+-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------+
| "db0"         | "standard"  | ["compositenw.product"]    | "read-write" | "localhost:7687" |
```

```
"primary" | TRUE   | "online"        | "online"      | ""            | FALSE   | FALSE | []
|
| "db1"         | "standard"  | ["compositenw.customerEU"]  | "read-write" | "localhost:7687" |
"primary" | TRUE   | "online"        | "online"      | ""            | FALSE   | FALSE | []
|
| "db2"         | "standard"  | ["compositenw.customerAME"] | "read-write" | "localhost:7687" |
"primary" | TRUE   | "online"        | "online"      | ""            | FALSE   | FALSE | []
|
| "compositenw" | "composite" | []                          | "read-only"  | "localhost:7687" |
"primary" | FALSE  | "online"        | "online"      | ""            | FALSE   | FALSE |
["compositenw.customerAME", "compositenw.customerEU", "compositenw.product"] |
| "neo4j"       | "standard"  | []                          | "read-write" | "localhost:7687" |
"primary" | TRUE   | "online"        | "online"      | ""            | TRUE    | TRUE  | []
|
| "system"      | "system"    | []                          | "read-write" | "localhost:7687" |
"primary" | TRUE   | "online"        | "online"      | ""            | FALSE   | FALSE | []
|
+-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
-------------------------------------------------------------+
6 rows available after 242 ms, consumed after another 18 ms
```

4. Run the command `SHOW ALIASES FOR DATABASES` to verify that the database aliases have been configured:

```
SHOW ALIASES FOR DATABASES;
```

```
+-------------------------------------------------------------------------------+
| name                      | composite     | database  | location | url  | user |
+-------------------------------------------------------------------------------+
| "compositenw.product"     | "compositenw" | "db0"     | "local"  | null | null |
| "compositenw.customerEU"  | "compositenw" | "db1"     | "local"  | null | null |
| "compositenw.customerAME" | "compositenw" | "db2"     | "local"  | null | null |
+-------------------------------------------------------------------------------+
3 rows available after 203 ms, consumed after another 16 ms
```

Retrieve data with a single Cypher query

Query a single database

When connected to a Composite database you can retrieve data from a single database by using the Cypher clause `USE` and the name of an alias:

```
:use compositenw
```

```
USE compositenw.product
MATCH (p:Product)
RETURN p.productName AS product
LIMIT 5;
```

```
+--------------------------------+
| product                        |
+--------------------------------+
| "Chai"                         |
| "Chang"                        |
| "Aniseed Syrup"                |
| "Chef Anton's Cajun Seasoning" |
| "Chef Anton's Gumbo Mix"       |
+--------------------------------+
```

```
5 rows available after 6 ms, consumed after another 21 ms
```

Query across multiple shards

Use the Composite database to query both shards and get customers whose name starts with A:

```
:use compositenw
```

```
USE compositenw.customerAME
MATCH (c:Customer)
WHERE c.customerID STARTS WITH 'A'
RETURN c.customerID AS name, c.country AS country
  UNION
USE compositenw.customerEU
MATCH (c:Customer)
WHERE c.customerID STARTS WITH 'A'
RETURN c.customerID AS name, c.country AS country
LIMIT 5;
```

```
+---------------------+
| name    | country   |
+---------------------+
| "ANATR" | "Mexico"  |
| "ANTON" | "Mexico"  |
| "ALFKI" | "Germany" |
| "AROUT" | "UK"      |
+---------------------+

4 rows available after 25 ms, consumed after another 56 ms
```

Or, using a more common Composite database idiom:

```
:use compositenw
```

```
UNWIND ['compositenw.customerAME', 'compositenw.customerEU'] AS g
CALL {
  USE graph.byName(g)
  MATCH (c:Customer)
  WHERE c.customerID STARTS WITH 'A'
  RETURN c.customerID AS name, c.country AS country
}
RETURN name, country
LIMIT 5;
```

```
+---------------------+
| name    | country   |
+---------------------+
| "ANATR" | "Mexico"  |
| "ANTON" | "Mexico"  |
| "ALFKI" | "Germany" |
| "AROUT" | "UK"      |
+---------------------+

4 rows available after 61 ms, consumed after another 8 ms
```

Query across federation and shards

Here is a more complex query that uses all 3 databases to find all customers who have bought

discontinued products in the Meat/Poultry category:

```
:use compositenw
```

```
CALL {
  USE compositenw.product
  MATCH (p:Product)-[:PART_OF]->(c:Category)
  WHERE p.discontinued = true
    AND c.categoryName = 'Meat/Poultry'
  RETURN COLLECT(p.productID) AS pids
}
WITH *
UNWIND [g IN graph.names() WHERE g STARTS WITH 'compositenw.customer'] AS g
CALL {
  USE graph.byName(g)
  WITH pids
  UNWIND pids as pid
  MATCH (p:Product{productID:pid})<-[:ORDERS]-(:Order)<-[:PURCHASED]-(c:Customer)
  RETURN DISTINCT c.customerID AS customer, c.country AS country
}
RETURN customer, country
LIMIT 20;
```

```
+--------------------------+
| customer | country       |
+--------------------------+
| "RICSU"  | "Switzerland" |
| "PERIC"  | "Mexico"      |
| "WARTH"  | "Finland"     |
| "WELLI"  | "Brazil"      |
| "DRACD"  | "Germany"     |
| "RATTC"  | "USA"         |
| "HUNGO"  | "Ireland"     |
| "QUEDE"  | "Brazil"      |
| "SEVES"  | "UK"          |
| "ANTON"  | "Mexico"      |
| "BERGS"  | "Sweden"      |
| "SAVEA"  | "USA"         |
| "AROUT"  | "UK"          |
| "FAMIA"  | "Brazil"      |
| "WANDK"  | "Germany"     |
| "WHITC"  | "USA"         |
| "ISLAT"  | "UK"          |
| "LONEP"  | "USA"         |
| "QUICK"  | "Germany"     |
| "HILAA"  | "Venezuela"   |
+--------------------------+

20 rows available after 51 ms, consumed after another 2 ms
```

The way this query works is by `compositenw` calling database `db0` to retrieve all discontinued products in the Meat/Poultry category. Then, using the returned product IDs, it queries both `db1` and `db2` **in parallel** and gets the customers who have purchased these products and their country.

You have just learned how to store and retrieve data from multiple databases using a single Cypher query.

For more details on Composite databases, see Concepts.

Fine-grained access control

When creating a database, administrators may want to establish which users can access certain information.

As described in Built-in roles and privileges, Neo4j already offers preset roles configured to specific permissions (i.e. read, edit, or write). While these built-in roles cover many common daily scenarios, it is also possible to create custom roles for specific needs.

This tutorial walks you through a healthcare use case that illustrates various aspects of security and fine-grained access control.

Healthcare use case

To demonstrate the application of these tools, consider an example of a *healthcare* database which could be relevant in a medical clinic or hospital.

For simplicity reasons, only three labels are used to represent the following entities:

`(:Patient)`

Patients that visit the clinic because they have some symptoms. Information specific to patients can be captured in properties:

- `name`
- `ssn`
- `address`
- `dateOfBirth`

`(:Symptom)`

A set of symptoms found in a catalog of known illnesses. They can be described using the properties:

- `name`
- `description`

`(:Disease)`

Known illnesses mapped in a catalog found in the database. They can be described using the properties:

- `name`
- `description`

These entities are modelled as nodes, and connected by relationships of the following types:

`(:Patient)-[:HAS]→(:Symptom)`

When a patient reports to the clinic, they describe their symptoms to the nurse or the doctor. The nurse or doctor then enters this information into the database in the form of connections between the patient node and a graph of known symptoms. Possible properties of interest on this relationship could be:

- `date` - date when symptom was reported.

`(:Symptom)-[:OF]→(:Disease)`

Symptoms are a subgraph in the graph of known diseases. The relationship between a symptom and a disease can include a probability factor for how likely or common it is for people with that disease to express that symptom. This will make it easier for the doctor to make a diagnosis using statistical

queries.

- `probability` - probability of symptom matching disease.

`(:Patient)-[:DIAGNOSIS]→(:Disease)`

The doctor can use the graph of diseases and their symptoms to perform an initial investigation into the most likely diseases to match the patient. Based on this, and their own assessment of the patient, the doctor may make a diagnosis which they would persist to the graph through the addition of this relationship with appropriate properties:

- `by`: doctor's name
- `date`: date of diagnosis
- `description`: additional doctors' notes

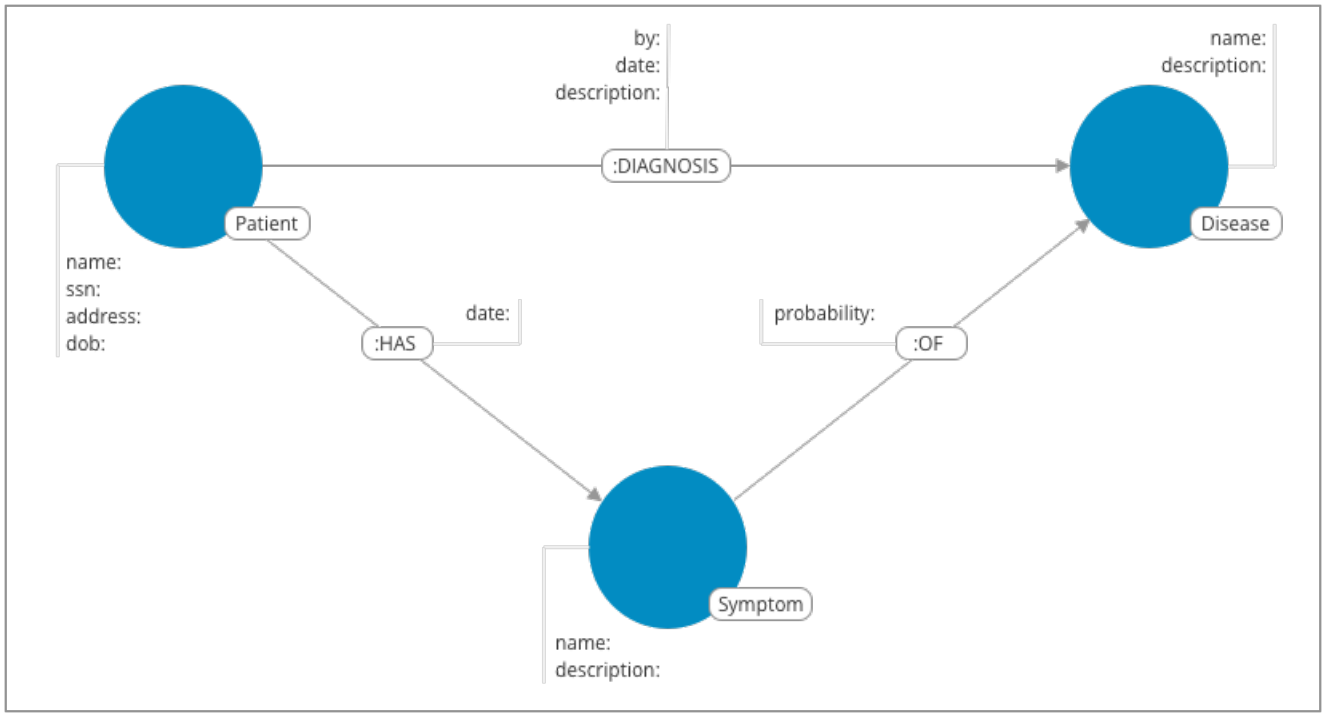


Figure 41. Healthcare use case

This same database would be used by a number of different users, each with different access needs:

- **Doctors** who need to diagnose patients.
- **Nurses** who need to treat patients.
- **Receptionists** who need to identify and record patient information.
- **Researchers** who need to perform statistical analysis of medical data.
- **IT administrators** who need to manage the database, to create and assign users for example.

To create the database, start by creating the nodes and relationships that represent the entities described above.

Create the `healthcare` database

The following steps assume that you have installed Neo4j Enterprise edition and it is running. For details,

see Installation.

1. Using Cypher Shell, log into the `system` database as the `neo4j` user:

```
cypher-shell -u neo4j -p my-password -d system
```

2. Create the `healthcare` database:

```
CREATE DATABASE healthcare;
```

3. Set the `healthcare` database as the default database:

```
CALL dbms.setDefaultDatabase("healthcare");
```

```
+------------------------------------------------------------------------+
| result                                                                 |
+------------------------------------------------------------------------+
| "Old default database unset. New default database set to healthcare" |
+------------------------------------------------------------------------+

1 row
ready to start consuming query after 9 ms, results consumed after another 4 ms
```

Import data into the `healthcare` database

Create the nodes and relationships that represent the entities described above in section Healthcare use case.

1. Switch to the `healthcare` database:

```
:use healthcare;
```

2. Create some data for symptoms:

```
WITH ['Itchy','Scratchy','Sore','Swollen','Red','Inflamed','Angry','Sad','Pale','Dizzy'] AS symptoms
             UNWIND symptoms AS symptom
             MERGE (s:Symptom {name:symptom})
             ON CREATE SET s.description = 'Looks ' + toLower(symptom)
             RETURN s.name, s.description;
```

▼ *Result*

```
+------------------------------+
| s.name     | s.description   |
+------------------------------+
| "Itchy"    | "Looks itchy"   |
| "Scratchy" | "Looks scratchy" |
| "Sore"     | "Looks sore"    |
| "Swollen"  | "Looks swollen" |
| "Red"      | "Looks red"     |
| "Inflamed" | "Looks inflamed" |
| "Angry"    | "Looks angry"   |
| "Sad"      | "Looks sad"     |
| "Pale"     | "Looks pale"    |
| "Dizzy"    | "Looks dizzy"   |
+------------------------------+
```

```
10 rows
ready to start consuming query after 53 ms, results consumed after another 24 ms
Added 10 nodes, Set 20 properties, Added 10 labels
```

3. Create some data for diseases:

```
WITH
  ['Argitis','Whatitis','Otheritis','Someitis','Placeboitis','Yellowitis'] AS diseases,
  ['Chronic','Acute'] AS severity
UNWIND diseases AS disease
UNWIND severity as sev
MERGE (d:Disease {name:sev+' '+disease})
ON CREATE SET d.description = sev + ' ' + toLower(disease)
RETURN d.name, d.description;
```

▼ *Result*

```
+-----------------------------------------------+
| d.name                | d.description         |
+-----------------------------------------------+
| "Chronic Argitis"     | "Chronic argitis"     |
| "Acute Argitis"       | "Acute argitis"       |
| "Chronic Whatitis"    | "Chronic whatitis"    |
| "Acute Whatitis"      | "Acute whatitis"      |
| "Chronic Otheritis"   | "Chronic otheritis"   |
| "Acute Otheritis"     | "Acute otheritis"     |
| "Chronic Someitis"    | "Chronic someitis"    |
| "Acute Someitis"      | "Acute someitis"      |
| "Chronic Placeboitis" | "Chronic placeboitis" |
| "Acute Placeboitis"   | "Acute placeboitis"   |
| "Chronic Yellowitis"  | "Chronic yellowitis"  |
| "Acute Yellowitis"    | "Acute yellowitis"    |
+-----------------------------------------------+

12 rows
ready to start consuming query after 56 ms, results consumed after another 7 ms
Added 12 nodes, Set 24 properties, Added 12 labels
```

4. Create relationships between symptoms and diseases:

```
MATCH (s:Symptom) WITH collect(s) as symptoms
WITH symptoms, size(symptoms) / 2 as maxsym
MATCH (d:Disease)
UNWIND range(0,maxsym) as symi
WITH d, symi, symptoms, toInteger(size(symptoms) * rand()) as si, rand()/2 + 0.5 AS prob
WITH d, symptoms[si] AS s, prob
MERGE (s)-[o:OF]->(d)
ON CREATE SET o.probability = prob
RETURN d.name, o.probability, s.name;
```

▼ *Result*

```
+-----------------------------------------------------------+
| d.name                | o.probability      | s.name     |
+-----------------------------------------------------------+
| "Chronic Argitis"     | 0.5488344602870381 | "Scratchy" |
| "Chronic Argitis"     | 0.660404649462915  | "Itchy"    |
| "Chronic Argitis"     | 0.6905998399032373 | "Angry"    |
| "Chronic Argitis"     | 0.660404649462915  | "Itchy"    |
| "Chronic Argitis"     | 0.8740581222813869 | "Red"      |
| "Chronic Argitis"     | 0.7456909803542418 | "Sore"     |
| "Acute Argitis"       | 0.607200508350778  | "Pale"     |
| "Acute Argitis"       | 0.5772236253537283 | "Red"      |
| "Acute Argitis"       | 0.7268375663608245 | "Inflamed" |
| "Acute Argitis"       | 0.847011132303783  | "Itchy"    |
| "Acute Argitis"       | 0.8025327549974599 | "Sore"     |
```

```
| "Acute Argitis"       | 0.5772236253537283 | "Red"      |
| "Chronic Whatitis"    | 0.9185112224896539 | "Sore"     |
| "Chronic Whatitis"    | 0.8220811592705012 | "Dizzy"    |
| "Chronic Whatitis"    | 0.8220811592705012 | "Dizzy"    |
| "Chronic Whatitis"    | 0.9947532896439784 | "Scratchy" |
| "Chronic Whatitis"    | 0.5479749642339755 | "Red"      |
| "Chronic Whatitis"    | 0.9466973516593605 | "Inflamed" |
| "Acute Whatitis"      | 0.7217509679510017 | "Inflamed" |
| "Acute Whatitis"      | 0.7217509679510017 | "Inflamed" |
| "Acute Whatitis"      | 0.7073350047270233 | "Scratchy" |
| "Acute Whatitis"      | 0.7217509679510017 | "Inflamed" |
| "Acute Whatitis"      | 0.6800748332507602 | "Red"      |
| "Acute Whatitis"      | 0.6953854679660172 | "Itchy"    |
| "Chronic Otheritis"   | 0.5570795327063996 | "Scratchy" |
| "Chronic Otheritis"   | 0.7615506655612736 | "Swollen"  |
| "Chronic Otheritis"   | 0.7147549568270981 | "Angry"    |
| "Chronic Otheritis"   | 0.9309059023795485 | "Red"      |
| "Chronic Otheritis"   | 0.8339105187862091 | "Dizzy"    |
| "Chronic Otheritis"   | 0.7147549568270981 | "Angry"    |
| "Acute Otheritis"     | 0.7449502448640619 | "Red"      |
| "Acute Otheritis"     | 0.6635390850482914 | "Sad"      |
| "Acute Otheritis"     | 0.6488764428922569 | "Itchy"    |
| "Acute Otheritis"     | 0.7642990617862074 | "Pale"     |
| "Acute Otheritis"     | 0.5532690807468361 | "Scratchy" |
| "Acute Otheritis"     | 0.8062425062999423 | "Inflamed" |
| "Chronic Someitis"    | 0.580678012588533  | "Sore"     |
| "Chronic Someitis"    | 0.9569035040624002 | "Red"      |
| "Chronic Someitis"    | 0.9328323008783481 | "Inflamed" |
| "Chronic Someitis"    | 0.9569035040624002 | "Red"      |
| "Chronic Someitis"    | 0.5492540886308123 | "Pale"     |
| "Chronic Someitis"    | 0.9204301026117075 | "Swollen"  |
| "Acute Someitis"      | 0.9969140989164824 | "Itchy"    |
| "Acute Someitis"      | 0.8756876989165112 | "Swollen"  |
| "Acute Someitis"      | 0.9969140989164824 | "Itchy"    |
| "Acute Someitis"      | 0.6258855371986936 | "Red"      |
| "Acute Someitis"      | 0.9928922186427123 | "Angry"    |
| "Acute Someitis"      | 0.6258855371986936 | "Red"      |
| "Chronic Placeboitis" | 0.9837947935707738 | "Itchy"    |
| "Chronic Placeboitis" | 0.7795050137703664 | "Inflamed" |
| "Chronic Placeboitis" | 0.680595344835278  | "Sad"      |
| "Chronic Placeboitis" | 0.8383237671521345 | "Scratchy" |
| "Chronic Placeboitis" | 0.7054054618102132 | "Swollen"  |
| "Chronic Placeboitis" | 0.7795050137703664 | "Inflamed" |
| "Acute Placeboitis"   | 0.768802727874529  | "Dizzy"    |
| "Acute Placeboitis"   | 0.6645530219645431 | "Scratchy" |
| "Acute Placeboitis"   | 0.9192262998770437 | "Pale"     |
| "Acute Placeboitis"   | 0.7321327463249545 | "Itchy"    |
| "Acute Placeboitis"   | 0.5768920173860386 | "Sad"      |
| "Acute Placeboitis"   | 0.5467367430608921 | "Sore"     |
| "Chronic Yellowitis"  | 0.657149882924074  | "Dizzy"    |
| "Chronic Yellowitis"  | 0.5274096280530778 | "Swollen"  |
| "Chronic Yellowitis"  | 0.657149882924074  | "Dizzy"    |
| "Chronic Yellowitis"  | 0.9011165844619397 | "Scratchy" |
| "Chronic Yellowitis"  | 0.5274096280530778 | "Swollen"  |
| "Chronic Yellowitis"  | 0.7267736062002124 | "Sore"     |
| "Acute Yellowitis"    | 0.7764355480097833 | "Swollen"  |
| "Acute Yellowitis"    | 0.9776709262803641 | "Inflamed" |
| "Acute Yellowitis"    | 0.6495454012653183 | "Red"      |
| "Acute Yellowitis"    | 0.7764355480097833 | "Swollen"  |
| "Acute Yellowitis"    | 0.7395280933743617 | "Dizzy"    |
| "Acute Yellowitis"    | 0.6068906083054821 | "Itchy"    |
+-----------------------------------------------------------+

72 rows
ready to start consuming query after 339 ms, results consumed after another 28 ms
Created 59 relationships, Set 59 properties
```

5. Ensure that the as yet non-existent types can be used:

```
CALL db.createRelationshipType('DIAGNOSIS');
CALL db.createProperty('by');
CALL db.createProperty('date');
CALL db.createProperty('description');
CALL db.createProperty('created_at');
```

```
CALL db.createProperty('updated_at');
```

▼ *Result*

```
0 rows
ready to start consuming query after 22 ms, results consumed after another 0 ms
0 rows
ready to start consuming query after 17 ms, results consumed after another 0 ms
0 rows
ready to start consuming query after 7 ms, results consumed after another 0 ms
0 rows
ready to start consuming query after 7 ms, results consumed after another 0 ms
0 rows
ready to start consuming query after 8 ms, results consumed after another 0 ms
0 rows
ready to start consuming query after 7 ms, results consumed after another 0 ms
```

6. Create some data for patients:

```
WITH
  ['Jack','Mary','Sally','Mark','Joe','Jane','Bob','Ally'] AS firstnames,
  ['Anderson','Jackson','Svensson','Smith','Stone'] AS surnames,
  ['mymail.com','example.com','other.org','net.net'] AS domains
UNWIND range(0,100) AS uid
WITH 1234567+uid AS ssn,
  firstnames[uid%size(firstnames)] AS firstname,
  surnames[uid%size(surnames)] AS surname,
  domains[uid%size(domains)] AS domain
WITH ssn, firstname, surname,
  tolower(firstname + '.' + surname + '@' + domain) AS email,
  toInteger(1500000000000 * rand()) AS ts
MERGE (p:Patient {ssn:ssn})
ON CREATE SET p.name = firstname + ' ' + surname,
  p.email = email,
  p.address = '1 secret way, downtown',
  p.dateOfBirth = date(datetime({epochmillis:ts}))
RETURN count(p);
```

▼ *Result*

```
+----------+
| count(p) |
+----------+
| 101      |
+----------+

1 row
ready to start consuming query after 49 ms, results consumed after another 38 ms
Added 101 nodes, Set 505 properties, Added 101 labels
```

7. Create relationships between patients and symptoms:

```
MATCH (s:Symptom) WITH collect(s) as symptoms
WITH symptoms, size(symptoms) / 2 as maxsym, 1500000000000 AS base, 75477004177 AS diff
MATCH (p:Patient)
UNWIND range(0,maxsym) as symi
WITH p, symi, symptoms, toInteger(size(symptoms) * rand()) as si, rand()/2 + 0.5 AS prob, base +
toInteger(diff * rand()) AS ts
WITH p, symptoms[si] AS s, prob, ts
MERGE (p)-[h:HAS]->(s)
ON CREATE SET h.date = date(datetime({epochmillis:ts}))
RETURN p.name, p.dateOfBirth, h.date, s.name;
```

▼ *Result*

```
+-------------------------------------------------------------+
| p.name           | p.dateOfBirth | h.date     | s.name      |
+-------------------------------------------------------------+
| "Jack Anderson"  | 1981-01-10    | 2019-03-03 | "Angry"     |
| "Jack Anderson"  | 1981-01-10    | 2018-05-05 | "Sad"       |
| "Jack Anderson"  | 1981-01-10    | 2018-06-05 | "Sore"      |
| "Jack Anderson"  | 1981-01-10    | 2017-11-17 | "Itchy"     |
| "Jack Anderson"  | 1981-01-10    | 2017-10-02 | "Dizzy"     |
| "Jack Anderson"  | 1981-01-10    | 2019-11-01 | "Red"       |
| "Mary Jackson"   | 1983-05-24    | 2018-03-30 | "Scratchy"  |
| "Mary Jackson"   | 1983-05-24    | 2018-03-08 | "Pale"      |
| "Mary Jackson"   | 1983-05-24    | 2019-05-03 | "Dizzy"     |
| "Mary Jackson"   | 1983-05-24    | 2019-08-16 | "Red"       |
| "Mary Jackson"   | 1983-05-24    | 2018-07-07 | "Inflamed"  |
| "Mary Jackson"   | 1983-05-24    | 2018-07-07 | "Inflamed"  |
| "Sally Svensson" | 2011-04-03    | 2018-02-12 | "Sore"      |
| "Sally Svensson" | 2011-04-03    | 2019-07-23 | "Pale"      |
| "Sally Svensson" | 2011-04-03    | 2019-04-17 | "Sad"       |
| "Sally Svensson" | 2011-04-03    | 2017-09-25 | "Red"       |
| "Sally Svensson" | 2011-04-03    | 2017-08-14 | "Swollen"   |
| "Sally Svensson" | 2011-04-03    | 2017-08-14 | "Swollen"   |
| "Mark Smith"     | 1998-08-26    | 2018-08-25 | "Inflamed"  |
| "Mark Smith"     | 1998-08-26    | 2018-08-25 | "Inflamed"  |
| "Mark Smith"     | 1998-08-26    | 2018-05-14 | "Itchy"     |
| "Mark Smith"     | 1998-08-26    | 2019-07-02 | "Dizzy"     |
| "Mark Smith"     | 1998-08-26    | 2018-02-27 | "Sad"       |
| "Mark Smith"     | 1998-08-26    | 2018-10-07 | "Swollen"   |
| "Joe Stone"      | 1972-10-20    | 2019-05-16 | "Red"       |
| "Joe Stone"      | 1972-10-20    | 2018-08-31 | "Inflamed"  |
| "Joe Stone"      | 1972-10-20    | 2018-08-31 | "Inflamed"  |
| "Joe Stone"      | 1972-10-20    | 2017-07-21 | "Sore"      |
| "Joe Stone"      | 1972-10-20    | 2018-08-31 | "Inflamed"  |
| "Joe Stone"      | 1972-10-20    | 2017-10-21 | "Itchy"     |
| "Jane Anderson"  | 2001-10-18    | 2018-07-04 | "Scratchy"  |
| "Jane Anderson"  | 2001-10-18    | 2019-02-09 | "Dizzy"     |
| "Jane Anderson"  | 2001-10-18    | 2018-05-03 | "Pale"      |
| "Jane Anderson"  | 2001-10-18    | 2019-08-13 | "Angry"     |
| "Jane Anderson"  | 2001-10-18    | 2018-05-03 | "Pale"      |
| "Jane Anderson"  | 2001-10-18    | 2019-11-12 | "Swollen"   |
| "Bob Jackson"    | 1997-08-20    | 2019-05-03 | "Sad"       |
| "Bob Jackson"    | 1997-08-20    | 2019-03-14 | "Red"       |
| "Bob Jackson"    | 1997-08-20    | 2019-03-01 | "Angry"     |
| "Bob Jackson"    | 1997-08-20    | 2018-03-10 | "Sore"      |
| "Bob Jackson"    | 1997-08-20    | 2018-03-10 | "Sore"      |
| "Bob Jackson"    | 1997-08-20    | 2019-05-03 | "Sad"       |
| "Ally Svensson"  | 2008-05-25    | 2019-06-06 | "Sore"      |
| "Ally Svensson"  | 2008-05-25    | 2019-11-04 | "Sad"       |
| "Ally Svensson"  | 2008-05-25    | 2018-10-04 | "Scratchy"  |
| "Ally Svensson"  | 2008-05-25    | 2017-11-28 | "Inflamed"  |
| "Ally Svensson"  | 2008-05-25    | 2018-10-29 | "Itchy"     |
| "Ally Svensson"  | 2008-05-25    | 2019-08-08 | "Angry"     |
| "Jack Smith"     | 1974-07-02    | 2018-01-15 | "Itchy"     |
| "Jack Smith"     | 1974-07-02    | 2019-02-12 | "Angry"     |
| "Jack Smith"     | 1974-07-02    | 2017-10-16 | "Dizzy"     |
| "Jack Smith"     | 1974-07-02    | 2018-01-03 | "Red"       |
| "Jack Smith"     | 1974-07-02    | 2018-01-15 | "Itchy"     |
| "Jack Smith"     | 1974-07-02    | 2017-11-14 | "Pale"      |
| "Mary Stone"     | 1983-09-27    | 2018-01-14 | "Dizzy"     |
| "Mary Stone"     | 1983-09-27    | 2019-03-24 | "Swollen"   |
| "Mary Stone"     | 1983-09-27    | 2018-04-07 | "Angry"     |
| "Mary Stone"     | 1983-09-27    | 2019-03-24 | "Swollen"   |
| "Mary Stone"     | 1983-09-27    | 2018-01-14 | "Dizzy"     |
| "Mary Stone"     | 1983-09-27    | 2017-11-27 | "Sore"      |
| "Sally Anderson" | 2009-12-14    | 2019-08-21 | "Swollen"   |
| "Sally Anderson" | 2009-12-14    | 2018-02-23 | "Sore"      |
| "Sally Anderson" | 2009-12-14    | 2018-06-05 | "Scratchy"  |
| "Sally Anderson" | 2009-12-14    | 2018-02-23 | "Sore"      |
| "Sally Anderson" | 2009-12-14    | 2017-08-20 | "Pale"      |
| "Sally Anderson" | 2009-12-14    | 2019-05-25 | "Itchy"     |
| "Mark Jackson"   | 1970-11-29    | 2018-09-27 | "Sore"      |
| "Mark Jackson"   | 1970-11-29    | 2017-12-27 | "Angry"     |
| "Mark Jackson"   | 1970-11-29    | 2017-12-26 | "Swollen"   |
| "Mark Jackson"   | 1970-11-29    | 2018-09-27 | "Sore"      |
| "Mark Jackson"   | 1970-11-29    | 2018-02-01 | "Inflamed"  |
| "Mark Jackson"   | 1970-11-29    | 2018-12-19 | "Pale"      |
| "Joe Svensson"   | 1972-02-12    | 2017-07-27 | "Sad"       |
```

"Joe Svensson"	1972-02-12	2019-07-27	"Itchy"
"Joe Svensson"	1972-02-12	2019-06-20	"Sore"
"Joe Svensson"	1972-02-12	2019-11-09	"Inflamed"
"Joe Svensson"	1972-02-12	2019-07-27	"Itchy"
"Joe Svensson"	1972-02-12	2019-07-27	"Itchy"
"Jane Smith"	2013-04-09	2018-12-13	"Swollen"
"Jane Smith"	2013-04-09	2018-12-13	"Swollen"
"Jane Smith"	2013-04-09	2019-03-13	"Red"
"Jane Smith"	2013-04-09	2019-01-25	"Dizzy"
"Jane Smith"	2013-04-09	2017-07-25	"Angry"
"Jane Smith"	2013-04-09	2018-04-11	"Inflamed"
"Bob Stone"	2012-04-11	2018-09-04	"Inflamed"
"Bob Stone"	2012-04-11	2017-08-07	"Red"
"Bob Stone"	2012-04-11	2019-07-11	"Swollen"
"Bob Stone"	2012-04-11	2019-07-11	"Swollen"
"Bob Stone"	2012-04-11	2019-07-14	"Sore"
"Bob Stone"	2012-04-11	2017-11-18	"Itchy"
"Ally Anderson"	2000-05-24	2018-11-27	"Itchy"
"Ally Anderson"	2000-05-24	2018-02-10	"Pale"
"Ally Anderson"	2000-05-24	2017-07-25	"Red"
"Ally Anderson"	2000-05-24	2018-03-19	"Sad"
"Ally Anderson"	2000-05-24	2017-07-25	"Red"
"Ally Anderson"	2000-05-24	2017-07-25	"Red"
"Jack Jackson"	1988-03-13	2018-06-29	"Sad"
"Jack Jackson"	1988-03-13	2019-06-25	"Sore"
"Jack Jackson"	1988-03-13	2019-05-27	"Inflamed"
"Jack Jackson"	1988-03-13	2018-07-09	"Angry"
"Jack Jackson"	1988-03-13	2018-04-23	"Pale"
"Jack Jackson"	1988-03-13	2019-05-15	"Dizzy"
"Mary Svensson"	2008-01-19	2018-03-16	"Dizzy"
"Mary Svensson"	2008-01-19	2018-04-14	"Red"
"Mary Svensson"	2008-01-19	2018-10-25	"Pale"
"Mary Svensson"	2008-01-19	2019-07-15	"Swollen"
"Mary Svensson"	2008-01-19	2019-07-15	"Swollen"
"Mary Svensson"	2008-01-19	2018-04-14	"Red"
"Sally Smith"	1977-03-20	2019-02-23	"Dizzy"
"Sally Smith"	1977-03-20	2017-11-28	"Red"
"Sally Smith"	1977-03-20	2018-06-23	"Scratchy"
"Sally Smith"	1977-03-20	2017-10-28	"Sad"
"Sally Smith"	1977-03-20	2017-11-28	"Red"
"Sally Smith"	1977-03-20	2018-10-05	"Inflamed"
"Mark Stone"	1986-06-15	2019-11-24	"Pale"
"Mark Stone"	1986-06-15	2018-07-30	"Itchy"
"Mark Stone"	1986-06-15	2018-07-30	"Itchy"
"Mark Stone"	1986-06-15	2017-09-10	"Dizzy"
"Mark Stone"	1986-06-15	2018-07-18	"Red"
"Mark Stone"	1986-06-15	2019-08-15	"Sore"
"Joe Anderson"	1980-09-06	2019-06-19	"Dizzy"
"Joe Anderson"	1980-09-06	2017-11-28	"Red"
"Joe Anderson"	1980-09-06	2019-08-12	"Scratchy"
"Joe Anderson"	1980-09-06	2019-08-12	"Scratchy"
"Joe Anderson"	1980-09-06	2019-06-19	"Dizzy"
"Joe Anderson"	1980-09-06	2017-07-16	"Inflamed"
"Jane Jackson"	2016-02-20	2018-04-03	"Swollen"
"Jane Jackson"	2016-02-20	2018-02-21	"Pale"
"Jane Jackson"	2016-02-20	2018-07-17	"Angry"
"Jane Jackson"	2016-02-20	2018-01-22	"Sore"
"Jane Jackson"	2016-02-20	2018-01-22	"Sore"
"Jane Jackson"	2016-02-20	2017-09-28	"Dizzy"
"Bob Svensson"	1983-08-04	2019-02-02	"Pale"
"Bob Svensson"	1983-08-04	2018-12-01	"Dizzy"
"Bob Svensson"	1983-08-04	2019-08-07	"Sad"
"Bob Svensson"	1983-08-04	2018-11-18	"Swollen"
"Bob Svensson"	1983-08-04	2018-12-25	"Scratchy"
"Bob Svensson"	1983-08-04	2018-04-09	"Inflamed"
"Ally Smith"	2012-03-01	2018-03-28	"Inflamed"
"Ally Smith"	2012-03-01	2018-03-28	"Inflamed"
"Ally Smith"	2012-03-01	2018-06-09	"Scratchy"
"Ally Smith"	2012-03-01	2019-01-25	"Angry"
"Ally Smith"	2012-03-01	2018-09-06	"Pale"
"Ally Smith"	2012-03-01	2018-12-04	"Dizzy"
"Jack Stone"	2009-11-08	2019-01-18	"Pale"
"Jack Stone"	2009-11-08	2018-03-29	"Angry"
"Jack Stone"	2009-11-08	2019-10-22	"Inflamed"
"Jack Stone"	2009-11-08	2019-01-18	"Pale"
"Jack Stone"	2009-11-08	2017-12-09	"Itchy"
"Jack Stone"	2009-11-08	2018-10-27	"Red"

"Mary Anderson"	1991-11-25	2018-01-02	"Angry"
"Mary Anderson"	1991-11-25	2018-01-02	"Angry"
"Mary Anderson"	1991-11-25	2017-11-01	"Inflamed"
"Mary Anderson"	1991-11-25	2017-12-16	"Sore"
"Mary Anderson"	1991-11-25	2018-01-02	"Angry"
"Mary Anderson"	1991-11-25	2018-03-22	"Red"
"Sally Jackson"	2008-11-09	2019-07-02	"Inflamed"
"Sally Jackson"	2008-11-09	2018-02-24	"Red"
"Sally Jackson"	2008-11-09	2019-08-07	"Swollen"
"Sally Jackson"	2008-11-09	2019-04-05	"Sore"
"Sally Jackson"	2008-11-09	2019-07-02	"Inflamed"
"Sally Jackson"	2008-11-09	2019-02-23	"Scratchy"
"Mark Svensson"	1979-06-22	2019-08-09	"Itchy"
"Mark Svensson"	1979-06-22	2019-05-11	"Swollen"
"Mark Svensson"	1979-06-22	2018-08-11	"Inflamed"
"Mark Svensson"	1979-06-22	2019-08-09	"Itchy"
"Mark Svensson"	1979-06-22	2017-10-11	"Sad"
"Mark Svensson"	1979-06-22	2019-09-22	"Scratchy"
"Joe Smith"	2008-07-03	2017-08-24	"Sore"
"Joe Smith"	2008-07-03	2018-12-03	"Red"
"Joe Smith"	2008-07-03	2018-12-03	"Red"
"Joe Smith"	2008-07-03	2018-08-20	"Inflamed"
"Joe Smith"	2008-07-03	2018-12-03	"Red"
"Joe Smith"	2008-07-03	2019-04-17	"Angry"
"Jane Stone"	1977-11-25	2018-03-19	"Scratchy"
"Jane Stone"	1977-11-25	2017-08-18	"Dizzy"
"Jane Stone"	1977-11-25	2017-12-09	"Red"
"Jane Stone"	1977-11-25	2018-06-14	"Swollen"
"Jane Stone"	1977-11-25	2018-08-22	"Pale"
"Jane Stone"	1977-11-25	2018-08-22	"Pale"
"Bob Anderson"	1970-04-27	2019-10-17	"Scratchy"
"Bob Anderson"	1970-04-27	2018-06-16	"Red"
"Bob Anderson"	1970-04-27	2017-11-07	"Itchy"
"Bob Anderson"	1970-04-27	2018-12-11	"Pale"
"Bob Anderson"	1970-04-27	2017-11-07	"Itchy"
"Bob Anderson"	1970-04-27	2019-02-26	"Swollen"
"Ally Jackson"	1982-01-12	2019-06-15	"Sad"
"Ally Jackson"	1982-01-12	2018-01-12	"Sore"
"Ally Jackson"	1982-01-12	2019-06-15	"Sad"
"Ally Jackson"	1982-01-12	2018-01-12	"Sore"
"Ally Jackson"	1982-01-12	2018-04-19	"Itchy"
"Ally Jackson"	1982-01-12	2019-04-06	"Red"
"Jack Svensson"	2012-08-22	2017-12-10	"Scratchy"
"Jack Svensson"	2012-08-22	2018-08-25	"Pale"
"Jack Svensson"	2012-08-22	2017-12-10	"Scratchy"
"Jack Svensson"	2012-08-22	2018-12-07	"Swollen"
"Jack Svensson"	2012-08-22	2018-08-25	"Pale"
"Jack Svensson"	2012-08-22	2018-04-30	"Red"
"Mary Smith"	2002-11-27	2018-07-26	"Red"
"Mary Smith"	2002-11-27	2018-04-09	"Dizzy"
"Mary Smith"	2002-11-27	2018-08-08	"Pale"
"Mary Smith"	2002-11-27	2018-08-28	"Sore"
"Mary Smith"	2002-11-27	2018-07-26	"Red"
"Mary Smith"	2002-11-27	2019-09-16	"Itchy"
"Sally Stone"	2001-04-25	2018-02-13	"Sore"
"Sally Stone"	2001-04-25	2019-05-03	"Itchy"
"Sally Stone"	2001-04-25	2019-09-25	"Dizzy"
"Sally Stone"	2001-04-25	2018-05-10	"Inflamed"
"Sally Stone"	2001-04-25	2019-09-03	"Scratchy"
"Sally Stone"	2001-04-25	2018-05-10	"Inflamed"
"Mark Anderson"	2007-06-19	2019-04-22	"Angry"
"Mark Anderson"	2007-06-19	2018-09-23	"Scratchy"
"Mark Anderson"	2007-06-19	2019-03-10	"Pale"
"Mark Anderson"	2007-06-19	2019-03-10	"Pale"
"Mark Anderson"	2007-06-19	2018-09-23	"Scratchy"
"Mark Anderson"	2007-06-19	2017-11-16	"Sad"
"Joe Jackson"	1991-10-12	2018-06-27	"Red"
"Joe Jackson"	1991-10-12	2018-10-26	"Pale"
"Joe Jackson"	1991-10-12	2018-10-30	"Sore"
"Joe Jackson"	1991-10-12	2018-10-30	"Sore"
"Joe Jackson"	1991-10-12	2018-10-26	"Pale"
"Joe Jackson"	1991-10-12	2019-01-06	"Swollen"
"Jane Svensson"	1982-07-02	2019-11-29	"Red"
"Jane Svensson"	1982-07-02	2017-12-07	"Angry"
"Jane Svensson"	1982-07-02	2019-04-05	"Swollen"
"Jane Svensson"	1982-07-02	2019-04-05	"Swollen"
"Jane Svensson"	1982-07-02	2018-12-10	"Sad"

"Jane Svensson"	1982-07-02	2019-11-09	"Inflamed"
"Bob Smith"	1981-10-29	2018-07-21	"Sad"
"Bob Smith"	1981-10-29	2019-09-15	"Itchy"
"Bob Smith"	1981-10-29	2019-04-18	"Scratchy"
"Bob Smith"	1981-10-29	2019-05-12	"Swollen"
"Bob Smith"	1981-10-29	2018-07-21	"Sad"
"Bob Smith"	1981-10-29	2019-02-04	"Pale"
"Ally Stone"	1980-12-13	2018-08-02	"Red"
"Ally Stone"	1980-12-13	2017-09-04	"Dizzy"
"Ally Stone"	1980-12-13	2017-09-04	"Dizzy"
"Ally Stone"	1980-12-13	2017-09-13	"Pale"
"Ally Stone"	1980-12-13	2018-01-21	"Sad"
"Ally Stone"	1980-12-13	2017-09-04	"Dizzy"
"Jack Anderson"	1998-11-09	2019-01-22	"Swollen"
"Jack Anderson"	1998-11-09	2019-07-14	"Red"
"Jack Anderson"	1998-11-09	2019-05-21	"Inflamed"
"Jack Anderson"	1998-11-09	2019-05-21	"Inflamed"
"Jack Anderson"	1998-11-09	2019-06-18	"Itchy"
"Jack Anderson"	1998-11-09	2019-01-22	"Swollen"
"Mary Jackson"	1974-09-25	2018-12-10	"Itchy"
"Mary Jackson"	1974-09-25	2017-10-13	"Swollen"
"Mary Jackson"	1974-09-25	2018-02-26	"Red"
"Mary Jackson"	1974-09-25	2018-01-25	"Sad"
"Mary Jackson"	1974-09-25	2017-08-05	"Inflamed"
"Mary Jackson"	1974-09-25	2018-09-22	"Scratchy"
"Sally Svensson"	1987-06-05	2018-06-23	"Red"
"Sally Svensson"	1987-06-05	2017-12-31	"Sad"
"Sally Svensson"	1987-06-05	2017-12-25	"Sore"
"Sally Svensson"	1987-06-05	2018-08-10	"Dizzy"
"Sally Svensson"	1987-06-05	2017-12-31	"Sad"
"Sally Svensson"	1987-06-05	2019-10-31	"Angry"
"Mark Smith"	1991-08-30	2019-07-28	"Swollen"
"Mark Smith"	1991-08-30	2019-01-14	"Itchy"
"Mark Smith"	1991-08-30	2018-11-09	"Sad"
"Mark Smith"	1991-08-30	2019-07-28	"Swollen"
"Mark Smith"	1991-08-30	2019-06-09	"Red"
"Mark Smith"	1991-08-30	2017-10-09	"Scratchy"
"Joe Stone"	1999-08-23	2017-09-15	"Itchy"
"Joe Stone"	1999-08-23	2019-08-12	"Dizzy"
"Joe Stone"	1999-08-23	2018-12-06	"Sore"
"Joe Stone"	1999-08-23	2018-06-04	"Swollen"
"Joe Stone"	1999-08-23	2019-11-14	"Inflamed"
"Joe Stone"	1999-08-23	2019-05-19	"Scratchy"
"Jane Anderson"	1988-07-16	2019-08-15	"Red"
"Jane Anderson"	1988-07-16	2018-09-26	"Sore"
"Jane Anderson"	1988-07-16	2018-10-22	"Pale"
"Jane Anderson"	1988-07-16	2018-03-20	"Inflamed"
"Jane Anderson"	1988-07-16	2019-05-13	"Dizzy"
"Jane Anderson"	1988-07-16	2019-05-13	"Dizzy"
"Bob Jackson"	1974-09-23	2019-01-07	"Sore"
"Bob Jackson"	1974-09-23	2017-10-13	"Scratchy"
"Bob Jackson"	1974-09-23	2019-07-20	"Swollen"
"Bob Jackson"	1974-09-23	2017-11-23	"Red"
"Bob Jackson"	1974-09-23	2019-04-07	"Sad"
"Bob Jackson"	1974-09-23	2019-08-23	"Itchy"
"Ally Svensson"	2006-11-13	2018-07-22	"Pale"
"Ally Svensson"	2006-11-13	2018-10-13	"Itchy"
"Ally Svensson"	2006-11-13	2017-10-07	"Sad"
"Ally Svensson"	2006-11-13	2018-10-13	"Itchy"
"Ally Svensson"	2006-11-13	2018-06-20	"Dizzy"
"Ally Svensson"	2006-11-13	2019-10-08	"Scratchy"
"Jack Smith"	2017-05-17	2018-03-20	"Red"
"Jack Smith"	2017-05-17	2019-01-13	"Swollen"
"Jack Smith"	2017-05-17	2018-08-06	"Itchy"
"Jack Smith"	2017-05-17	2018-07-18	"Scratchy"
"Jack Smith"	2017-05-17	2018-06-10	"Sore"
"Jack Smith"	2017-05-17	2018-03-20	"Red"
"Mary Stone"	2011-06-20	2019-02-07	"Pale"
"Mary Stone"	2011-06-20	2018-12-07	"Itchy"
"Mary Stone"	2011-06-20	2019-09-17	"Scratchy"
"Mary Stone"	2011-06-20	2017-08-02	"Sore"
"Mary Stone"	2011-06-20	2019-09-17	"Scratchy"
"Mary Stone"	2011-06-20	2019-02-07	"Pale"
"Sally Anderson"	1970-12-02	2018-10-20	"Swollen"
"Sally Anderson"	1970-12-02	2019-02-05	"Scratchy"
"Sally Anderson"	1970-12-02	2019-11-12	"Pale"
"Sally Anderson"	1970-12-02	2018-03-21	"Angry"

"Sally Anderson"	1970-12-02	2019-07-21	"Inflamed"
"Sally Anderson"	1970-12-02	2019-07-21	"Inflamed"
"Mark Jackson"	2003-07-09	2018-12-20	"Sore"
"Mark Jackson"	2003-07-09	2018-04-13	"Itchy"
"Mark Jackson"	2003-07-09	2018-11-08	"Inflamed"
"Mark Jackson"	2003-07-09	2019-09-17	"Swollen"
"Mark Jackson"	2003-07-09	2018-04-11	"Dizzy"
"Mark Jackson"	2003-07-09	2018-12-20	"Sore"
"Joe Svensson"	2000-03-07	2019-01-31	"Angry"
"Joe Svensson"	2000-03-07	2018-03-29	"Sore"
"Joe Svensson"	2000-03-07	2019-10-26	"Pale"
"Joe Svensson"	2000-03-07	2019-01-31	"Angry"
"Joe Svensson"	2000-03-07	2018-01-01	"Scratchy"
"Joe Svensson"	2000-03-07	2018-03-29	"Sore"
"Jane Smith"	2012-05-14	2019-03-18	"Pale"
"Jane Smith"	2012-05-14	2018-08-15	"Swollen"
"Jane Smith"	2012-05-14	2018-01-16	"Sore"
"Jane Smith"	2012-05-14	2018-03-14	"Scratchy"
"Jane Smith"	2012-05-14	2018-05-23	"Inflamed"
"Jane Smith"	2012-05-14	2019-07-06	"Red"
"Bob Stone"	2011-06-07	2018-03-12	"Itchy"
"Bob Stone"	2011-06-07	2018-05-20	"Sad"
"Bob Stone"	2011-06-07	2017-08-12	"Red"
"Bob Stone"	2011-06-07	2018-01-13	"Swollen"
"Bob Stone"	2011-06-07	2019-01-13	"Angry"
"Bob Stone"	2011-06-07	2018-03-12	"Itchy"
"Ally Anderson"	1972-05-20	2018-09-27	"Pale"
"Ally Anderson"	1972-05-20	2017-08-11	"Inflamed"
"Ally Anderson"	1972-05-20	2017-08-23	"Sad"
"Ally Anderson"	1972-05-20	2019-06-09	"Dizzy"
"Ally Anderson"	1972-05-20	2018-10-08	"Scratchy"
"Ally Anderson"	1972-05-20	2018-06-13	"Swollen"
"Jack Jackson"	1985-09-11	2019-01-06	"Red"
"Jack Jackson"	1985-09-11	2018-02-05	"Sore"
"Jack Jackson"	1985-09-11	2018-09-10	"Scratchy"
"Jack Jackson"	1985-09-11	2019-10-17	"Dizzy"
"Jack Jackson"	1985-09-11	2018-07-07	"Angry"
"Jack Jackson"	1985-09-11	2018-02-05	"Sore"
"Mary Svensson"	1987-09-18	2018-08-06	"Red"
"Mary Svensson"	1987-09-18	2018-03-03	"Scratchy"
"Mary Svensson"	1987-09-18	2018-10-13	"Sad"
"Mary Svensson"	1987-09-18	2019-02-03	"Sore"
"Mary Svensson"	1987-09-18	2018-08-06	"Red"
"Mary Svensson"	1987-09-18	2018-08-06	"Red"
"Sally Smith"	2005-07-05	2018-05-26	"Dizzy"
"Sally Smith"	2005-07-05	2018-11-02	"Sad"
"Sally Smith"	2005-07-05	2018-05-26	"Dizzy"
"Sally Smith"	2005-07-05	2017-10-26	"Inflamed"
"Sally Smith"	2005-07-05	2018-07-24	"Angry"
"Sally Smith"	2005-07-05	2018-05-26	"Dizzy"
"Mark Stone"	2011-01-01	2019-01-24	"Red"
"Mark Stone"	2011-01-01	2018-02-26	"Scratchy"
"Mark Stone"	2011-01-01	2018-11-11	"Swollen"
"Mark Stone"	2011-01-01	2017-12-16	"Sore"
"Mark Stone"	2011-01-01	2018-02-26	"Scratchy"
"Mark Stone"	2011-01-01	2019-09-13	"Sad"
"Joe Anderson"	1981-12-16	2017-11-29	"Pale"
"Joe Anderson"	1981-12-16	2018-12-13	"Dizzy"
"Joe Anderson"	1981-12-16	2018-06-05	"Swollen"
"Joe Anderson"	1981-12-16	2018-09-27	"Sad"
"Joe Anderson"	1981-12-16	2017-09-12	"Inflamed"
"Joe Anderson"	1981-12-16	2019-10-10	"Sore"
"Jane Jackson"	1989-10-16	2019-04-22	"Dizzy"
"Jane Jackson"	1989-10-16	2019-04-30	"Swollen"
"Jane Jackson"	1989-10-16	2018-04-19	"Red"
"Jane Jackson"	1989-10-16	2018-09-28	"Inflamed"
"Jane Jackson"	1989-10-16	2019-07-19	"Scratchy"
"Jane Jackson"	1989-10-16	2018-05-19	"Sad"
"Bob Svensson"	2003-05-06	2019-11-05	"Sore"
"Bob Svensson"	2003-05-06	2018-08-09	"Scratchy"
"Bob Svensson"	2003-05-06	2018-11-22	"Inflamed"
"Bob Svensson"	2003-05-06	2018-02-14	"Angry"
"Bob Svensson"	2003-05-06	2018-11-22	"Inflamed"
"Bob Svensson"	2003-05-06	2018-02-25	"Itchy"
"Ally Smith"	1979-08-06	2019-10-25	"Pale"
"Ally Smith"	1979-08-06	2019-11-25	"Sore"
"Ally Smith"	1979-08-06	2019-10-19	"Dizzy"

"Ally Smith"	1979-08-06	2018-01-06	"Sad"
"Ally Smith"	1979-08-06	2019-03-12	"Red"
"Ally Smith"	1979-08-06	2019-05-25	"Itchy"
"Jack Stone"	2003-12-08	2019-04-29	"Swollen"
"Jack Stone"	2003-12-08	2018-09-02	"Scratchy"
"Jack Stone"	2003-12-08	2019-07-06	"Itchy"
"Jack Stone"	2003-12-08	2019-07-06	"Itchy"
"Jack Stone"	2003-12-08	2018-04-16	"Pale"
"Jack Stone"	2003-12-08	2018-02-10	"Sore"
"Mary Anderson"	1974-07-22	2018-06-08	"Dizzy"
"Mary Anderson"	1974-07-22	2018-06-08	"Dizzy"
"Mary Anderson"	1974-07-22	2019-12-02	"Pale"
"Mary Anderson"	1974-07-22	2018-09-08	"Angry"
"Mary Anderson"	1974-07-22	2018-07-05	"Swollen"
"Mary Anderson"	1974-07-22	2018-03-08	"Itchy"
"Sally Jackson"	1994-02-20	2019-07-19	"Dizzy"
"Sally Jackson"	1994-02-20	2019-06-29	"Pale"
"Sally Jackson"	1994-02-20	2019-06-14	"Angry"
"Sally Jackson"	1994-02-20	2018-07-27	"Red"
"Sally Jackson"	1994-02-20	2019-01-21	"Sad"
"Sally Jackson"	1994-02-20	2018-10-25	"Swollen"
"Mark Svensson"	1985-09-07	2018-01-06	"Inflamed"
"Mark Svensson"	1985-09-07	2018-01-06	"Inflamed"
"Mark Svensson"	1985-09-07	2019-04-04	"Red"
"Mark Svensson"	1985-09-07	2018-04-02	"Angry"
"Mark Svensson"	1985-09-07	2018-11-12	"Itchy"
"Mark Svensson"	1985-09-07	2018-11-12	"Itchy"
"Joe Smith"	1980-06-17	2018-10-23	"Sore"
"Joe Smith"	1980-06-17	2018-03-19	"Angry"
"Joe Smith"	1980-06-17	2018-03-19	"Angry"
"Joe Smith"	1980-06-17	2019-02-12	"Itchy"
"Joe Smith"	1980-06-17	2019-02-12	"Itchy"
"Joe Smith"	1980-06-17	2019-09-17	"Pale"
"Jane Stone"	2015-11-26	2019-04-27	"Scratchy"
"Jane Stone"	2015-11-26	2017-09-21	"Itchy"
"Jane Stone"	2015-11-26	2017-07-18	"Inflamed"
"Jane Stone"	2015-11-26	2018-04-05	"Pale"
"Jane Stone"	2015-11-26	2017-07-18	"Inflamed"
"Jane Stone"	2015-11-26	2019-01-12	"Swollen"
"Bob Anderson"	2007-02-06	2018-08-16	"Itchy"
"Bob Anderson"	2007-02-06	2019-08-23	"Inflamed"
"Bob Anderson"	2007-02-06	2018-08-31	"Dizzy"
"Bob Anderson"	2007-02-06	2019-01-16	"Sore"
"Bob Anderson"	2007-02-06	2018-08-31	"Dizzy"
"Bob Anderson"	2007-02-06	2018-02-14	"Angry"
"Ally Jackson"	1997-12-29	2018-03-10	"Pale"
"Ally Jackson"	1997-12-29	2019-11-21	"Red"
"Ally Jackson"	1997-12-29	2018-03-10	"Pale"
"Ally Jackson"	1997-12-29	2019-11-21	"Red"
"Ally Jackson"	1997-12-29	2018-09-20	"Dizzy"
"Ally Jackson"	1997-12-29	2018-03-12	"Itchy"
"Jack Svensson"	1974-04-26	2018-07-04	"Sore"
"Jack Svensson"	1974-04-26	2019-06-15	"Angry"
"Jack Svensson"	1974-04-26	2019-09-04	"Inflamed"
"Jack Svensson"	1974-04-26	2017-08-12	"Swollen"
"Jack Svensson"	1974-04-26	2018-07-04	"Sore"
"Jack Svensson"	1974-04-26	2019-11-06	"Itchy"
"Mary Smith"	2007-06-30	2018-06-13	"Sad"
"Mary Smith"	2007-06-30	2019-06-21	"Itchy"
"Mary Smith"	2007-06-30	2019-02-04	"Dizzy"
"Mary Smith"	2007-06-30	2018-03-15	"Angry"
"Mary Smith"	2007-06-30	2018-03-15	"Angry"
"Mary Smith"	2007-06-30	2018-12-07	"Sore"
"Sally Stone"	1999-06-21	2018-01-05	"Sore"
"Sally Stone"	1999-06-21	2018-02-19	"Angry"
"Sally Stone"	1999-06-21	2018-01-05	"Sore"
"Sally Stone"	1999-06-21	2018-01-05	"Sore"
"Sally Stone"	1999-06-21	2018-04-08	"Scratchy"
"Sally Stone"	1999-06-21	2018-01-18	"Dizzy"
"Mark Anderson"	1995-09-26	2018-03-04	"Scratchy"
"Mark Anderson"	1995-09-26	2018-10-14	"Sad"
"Mark Anderson"	1995-09-26	2019-05-16	"Pale"
"Mark Anderson"	1995-09-26	2017-12-09	"Swollen"
"Mark Anderson"	1995-09-26	2019-04-17	"Inflamed"
"Mark Anderson"	1995-09-26	2019-05-16	"Pale"
"Joe Jackson"	1973-06-05	2019-01-03	"Scratchy"
"Joe Jackson"	1973-06-05	2018-08-29	"Itchy"

"Joe Jackson"	1973-06-05	2019-07-06	"Angry"
"Joe Jackson"	1973-06-05	2018-08-29	"Itchy"
"Joe Jackson"	1973-06-05	2018-04-21	"Dizzy"
"Joe Jackson"	1973-06-05	2018-06-21	"Sore"
"Jane Svensson"	1970-05-24	2018-05-03	"Sore"
"Jane Svensson"	1970-05-24	2019-02-12	"Inflamed"
"Jane Svensson"	1970-05-24	2018-04-18	"Angry"
"Jane Svensson"	1970-05-24	2019-04-12	"Swollen"
"Jane Svensson"	1970-05-24	2018-12-08	"Red"
"Jane Svensson"	1970-05-24	2017-11-17	"Itchy"
"Bob Smith"	2014-07-07	2018-04-05	"Dizzy"
"Bob Smith"	2014-07-07	2018-01-21	"Red"
"Bob Smith"	2014-07-07	2018-04-05	"Dizzy"
"Bob Smith"	2014-07-07	2018-04-05	"Dizzy"
"Bob Smith"	2014-07-07	2018-02-04	"Sad"
"Bob Smith"	2014-07-07	2019-05-01	"Pale"
"Ally Stone"	1994-08-11	2017-10-17	"Inflamed"
"Ally Stone"	1994-08-11	2017-08-20	"Red"
"Ally Stone"	1994-08-11	2017-10-17	"Inflamed"
"Ally Stone"	1994-08-11	2019-03-15	"Angry"
"Ally Stone"	1994-08-11	2019-03-15	"Angry"
"Ally Stone"	1994-08-11	2018-08-26	"Swollen"
"Jack Anderson"	1994-08-22	2017-09-25	"Inflamed"
"Jack Anderson"	1994-08-22	2019-10-18	"Sad"
"Jack Anderson"	1994-08-22	2018-11-12	"Swollen"
"Jack Anderson"	1994-08-22	2019-10-18	"Sad"
"Jack Anderson"	1994-08-22	2018-11-12	"Swollen"
"Jack Anderson"	1994-08-22	2018-09-29	"Sore"
"Mary Jackson"	1993-02-17	2017-07-18	"Scratchy"
"Mary Jackson"	1993-02-17	2019-09-15	"Red"
"Mary Jackson"	1993-02-17	2018-05-28	"Itchy"
"Mary Jackson"	1993-02-17	2018-05-28	"Itchy"
"Mary Jackson"	1993-02-17	2018-09-28	"Inflamed"
"Mary Jackson"	1993-02-17	2017-08-19	"Sad"
"Sally Svensson"	2015-04-19	2017-11-15	"Sad"
"Sally Svensson"	2015-04-19	2018-07-30	"Sore"
"Sally Svensson"	2015-04-19	2017-11-15	"Sad"
"Sally Svensson"	2015-04-19	2018-10-05	"Pale"
"Sally Svensson"	2015-04-19	2019-06-14	"Dizzy"
"Sally Svensson"	2015-04-19	2018-03-09	"Scratchy"
"Mark Smith"	2012-08-01	2018-09-27	"Swollen"
"Mark Smith"	2012-08-01	2018-06-25	"Angry"
"Mark Smith"	2012-08-01	2019-01-08	"Sore"
"Mark Smith"	2012-08-01	2018-09-20	"Pale"
"Mark Smith"	2012-08-01	2019-09-30	"Scratchy"
"Mark Smith"	2012-08-01	2018-09-20	"Pale"
"Joe Stone"	2003-12-31	2018-01-29	"Sore"
"Joe Stone"	2003-12-31	2017-10-01	"Angry"
"Joe Stone"	2003-12-31	2019-07-16	"Scratchy"
"Joe Stone"	2003-12-31	2018-03-03	"Red"
"Joe Stone"	2003-12-31	2017-10-14	"Dizzy"
"Joe Stone"	2003-12-31	2017-08-18	"Pale"
"Jane Anderson"	2010-06-04	2019-12-02	"Sore"
"Jane Anderson"	2010-06-04	2018-03-24	"Scratchy"
"Jane Anderson"	2010-06-04	2018-07-20	"Sad"
"Jane Anderson"	2010-06-04	2019-06-17	"Swollen"
"Jane Anderson"	2010-06-04	2018-12-21	"Red"
"Jane Anderson"	2010-06-04	2019-12-02	"Sore"
"Bob Jackson"	1979-11-07	2018-05-03	"Sore"
"Bob Jackson"	1979-11-07	2018-04-20	"Angry"
"Bob Jackson"	1979-11-07	2018-05-10	"Pale"
"Bob Jackson"	1979-11-07	2018-02-09	"Swollen"
"Bob Jackson"	1979-11-07	2019-10-14	"Scratchy"
"Bob Jackson"	1979-11-07	2018-04-23	"Dizzy"
"Ally Svensson"	2004-12-14	2019-03-13	"Scratchy"
"Ally Svensson"	2004-12-14	2019-03-13	"Scratchy"
"Ally Svensson"	2004-12-14	2018-02-22	"Inflamed"
"Ally Svensson"	2004-12-14	2018-05-16	"Dizzy"
"Ally Svensson"	2004-12-14	2018-07-09	"Pale"
"Ally Svensson"	2004-12-14	2019-04-30	"Itchy"
"Jack Smith"	2010-09-23	2018-11-27	"Angry"
"Jack Smith"	2010-09-23	2018-04-18	"Pale"
"Jack Smith"	2010-09-23	2019-03-05	"Itchy"
"Jack Smith"	2010-09-23	2018-09-29	"Sore"
"Jack Smith"	2010-09-23	2019-02-08	"Red"
"Jack Smith"	2010-09-23	2019-02-22	"Sad"
"Mary Stone"	2009-07-14	2017-11-22	"Pale"

```
| "Mary Stone"     | 2009-07-14    | 2018-09-21 | "Scratchy" |
| "Mary Stone"     | 2009-07-14    | 2018-09-30 | "Angry"    |
| "Mary Stone"     | 2009-07-14    | 2019-06-17 | "Itchy"    |
| "Mary Stone"     | 2009-07-14    | 2017-11-22 | "Pale"     |
| "Mary Stone"     | 2009-07-14    | 2019-08-14 | "Sore"     |
| "Sally Anderson" | 1972-01-23    | 2017-09-13 | "Dizzy"    |
| "Sally Anderson" | 1972-01-23    | 2018-09-09 | "Itchy"    |
| "Sally Anderson" | 1972-01-23    | 2018-06-23 | "Sad"      |
| "Sally Anderson" | 1972-01-23    | 2018-06-23 | "Sad"      |
| "Sally Anderson" | 1972-01-23    | 2019-12-03 | "Sore"     |
| "Sally Anderson" | 1972-01-23    | 2019-12-03 | "Sore"     |
| "Mark Jackson"   | 2006-12-17    | 2018-08-02 | "Scratchy" |
| "Mark Jackson"   | 2006-12-17    | 2018-01-28 | "Sore"     |
| "Mark Jackson"   | 2006-12-17    | 2018-08-02 | "Scratchy" |
| "Mark Jackson"   | 2006-12-17    | 2018-05-26 | "Itchy"    |
| "Mark Jackson"   | 2006-12-17    | 2017-11-27 | "Sad"      |
| "Mark Jackson"   | 2006-12-17    | 2017-07-17 | "Red"      |
| "Joe Svensson"   | 2001-01-14    | 2019-02-20 | "Dizzy"    |
| "Joe Svensson"   | 2001-01-14    | 2018-07-12 | "Angry"    |
| "Joe Svensson"   | 2001-01-14    | 2018-08-08 | "Itchy"    |
| "Joe Svensson"   | 2001-01-14    | 2018-03-30 | "Pale"     |
| "Joe Svensson"   | 2001-01-14    | 2019-02-20 | "Dizzy"    |
| "Joe Svensson"   | 2001-01-14    | 2018-03-30 | "Pale"     |
| "Jane Smith"     | 1992-03-12    | 2017-10-07 | "Dizzy"    |
| "Jane Smith"     | 1992-03-12    | 2019-03-07 | "Inflamed" |
| "Jane Smith"     | 1992-03-12    | 2019-05-30 | "Red"      |
| "Jane Smith"     | 1992-03-12    | 2017-07-15 | "Pale"     |
| "Jane Smith"     | 1992-03-12    | 2019-04-14 | "Sore"     |
| "Jane Smith"     | 1992-03-12    | 2019-09-30 | "Angry"    |
| "Bob Stone"      | 2015-09-07    | 2018-09-11 | "Sore"     |
| "Bob Stone"      | 2015-09-07    | 2018-03-19 | "Dizzy"    |
| "Bob Stone"      | 2015-09-07    | 2019-01-01 | "Scratchy" |
| "Bob Stone"      | 2015-09-07    | 2019-01-01 | "Scratchy" |
| "Bob Stone"      | 2015-09-07    | 2017-12-27 | "Sad"      |
| "Bob Stone"      | 2015-09-07    | 2019-01-01 | "Scratchy" |
| "Ally Anderson"  | 1978-12-20    | 2018-08-20 | "Inflamed" |
| "Ally Anderson"  | 1978-12-20    | 2018-03-23 | "Sad"      |
| "Ally Anderson"  | 1978-12-20    | 2017-09-05 | "Itchy"    |
| "Ally Anderson"  | 1978-12-20    | 2018-01-07 | "Angry"    |
| "Ally Anderson"  | 1978-12-20    | 2018-02-08 | "Pale"     |
| "Ally Anderson"  | 1978-12-20    | 2018-02-08 | "Pale"     |
| "Jack Jackson"   | 1975-06-08    | 2018-10-10 | "Red"      |
| "Jack Jackson"   | 1975-06-08    | 2019-09-16 | "Pale"     |
| "Jack Jackson"   | 1975-06-08    | 2019-09-16 | "Pale"     |
| "Jack Jackson"   | 1975-06-08    | 2019-09-16 | "Pale"     |
| "Jack Jackson"   | 1975-06-08    | 2019-01-06 | "Angry"    |
| "Jack Jackson"   | 1975-06-08    | 2018-02-21 | "Scratchy" |
| "Mary Svensson"  | 2002-01-09    | 2018-09-24 | "Pale"     |
| "Mary Svensson"  | 2002-01-09    | 2018-03-03 | "Itchy"    |
| "Mary Svensson"  | 2002-01-09    | 2017-12-29 | "Swollen"  |
| "Mary Svensson"  | 2002-01-09    | 2019-11-01 | "Sad"      |
| "Mary Svensson"  | 2002-01-09    | 2018-03-03 | "Itchy"    |
| "Mary Svensson"  | 2002-01-09    | 2018-03-03 | "Itchy"    |
| "Sally Smith"    | 1973-12-03    | 2018-01-06 | "Itchy"    |
| "Sally Smith"    | 1973-12-03    | 2018-01-06 | "Itchy"    |
| "Sally Smith"    | 1973-12-03    | 2019-02-13 | "Dizzy"    |
| "Sally Smith"    | 1973-12-03    | 2019-10-21 | "Pale"     |
| "Sally Smith"    | 1973-12-03    | 2017-10-26 | "Scratchy" |
| "Sally Smith"    | 1973-12-03    | 2017-12-21 | "Red"      |
| "Mark Stone"     | 1988-06-07    | 2018-06-07 | "Swollen"  |
| "Mark Stone"     | 1988-06-07    | 2019-09-06 | "Scratchy" |
| "Mark Stone"     | 1988-06-07    | 2018-08-09 | "Itchy"    |
| "Mark Stone"     | 1988-06-07    | 2019-09-06 | "Scratchy" |
| "Mark Stone"     | 1988-06-07    | 2019-06-12 | "Dizzy"    |
| "Mark Stone"     | 1988-06-07    | 2019-09-06 | "Scratchy" |
| "Joe Anderson"   | 2010-10-05    | 2018-01-05 | "Pale"     |
| "Joe Anderson"   | 2010-10-05    | 2017-12-26 | "Scratchy" |
| "Joe Anderson"   | 2010-10-05    | 2018-01-05 | "Pale"     |
| "Joe Anderson"   | 2010-10-05    | 2019-04-11 | "Inflamed" |
| "Joe Anderson"   | 2010-10-05    | 2019-01-02 | "Sore"     |
| "Joe Anderson"   | 2010-10-05    | 2019-02-28 | "Dizzy"    |
+-------------------------------------------------------------+

606 rows
ready to start consuming query after 91 ms, results consumed after another 60 ms
Created 491 relationships, Set 491 properties
```

Manage authorization and access control

Unlike applications which often require users to be modeled within the application itself, databases provide user management resources such as roles and privileges. This allows users to be created entirely within the database security model, a strategy that allows the separation of access to the data and the data itself. For more information, see Authentication and authorization.

In this tutorial, consider five users of the *healthcare* database:

- Alice, the doctor.
- Daniel, the nurse.
- Bob, the receptionist.
- Charlie, the researcher.
- Tina, the IT administrator.

You can create these users by using the `CREATE USER` command (from the `system` database):

```
CREATE USER charlie SET PASSWORD 'secretpassword1' CHANGE NOT REQUIRED;
CREATE USER alice SET PASSWORD 'secretpassword2' CHANGE NOT REQUIRED;
CREATE USER daniel SET PASSWORD 'secretpassword3' CHANGE NOT REQUIRED;
CREATE USER bob SET PASSWORD 'secretpassword4' CHANGE NOT REQUIRED;
CREATE USER tina SET PASSWORD 'secretpassword5' CHANGE NOT REQUIRED;
```

At this point, the users cannot interact with the database, so these capabilities need to be granted by using roles. There are two different ways of doing this either by using built-in roles and privileges or by using more advanced resources with fine-grained privileges for sub-graph access control.

Access control using built-in roles

Neo4j comes with built-in roles that cover a number of common needs:

- `PUBLIC` - All users have this role. They can by default access the home database, load data, and run all procedures and user-defined functions.
- `reader` - Can read data from all databases.
- `editor` - Can read and update all databases, but not expand the schema with new labels, relationship types, or property names.
- `publisher` - Can read and edit, as well as add new labels, relationship types, and property names.
- `architect` - Has all the capabilities of the publisher as well as the ability to manage indexes and constraints.
- `admin` - Can perform architect actions as well as load data and manage databases, users, roles, and privileges.

Consider Charlie from the example of users. As a researcher, they do not need write access to the database, so they are assigned the `reader` role.

On the other hand, Alice (the doctor), Daniel (the nurse), and Bob (the receptionist) all need to update the database with new patient information but do not need to expand the schema with new labels,

relationship types, property names, or indexes. For this reason, they are all assigned the `editor` role.

Tina, the IT administrator who installs and manages the database, needs to be assigned the `admin` role.

Here is how to grant roles to the users (from the `system` database):

```
GRANT ROLE reader TO charlie;
GRANT ROLE editor TO alice;
GRANT ROLE editor TO daniel;
GRANT ROLE editor TO bob;
GRANT ROLE admin TO tina;
```

Sub-graph access control using privileges

A limitation of the previously described approach is that it does allow all users to see all the data on the database. In many real-world scenarios though, it would be preferable to establish some access restrictions.

For example, you may want to limit the researcher's access to the patients' personal information or restrict the receptionist from writing new labels on the database. While these restrictions could be coded into the application layer, it is possible and **more secure** to enforce fine-grained restrictions directly within the Neo4j security model by creating custom roles and assigning specific privileges to them.

Since new custom roles will be created, it is important to first revoke the current roles from the users assigned to them. Run the following command against the `system` database:

```
REVOKE ROLE reader FROM charlie;
REVOKE ROLE editor FROM alice;
REVOKE ROLE editor FROM daniel;
REVOKE ROLE editor FROM bob;
REVOKE ROLE admin FROM tina;
```

Now you can create custom roles based on the concept of *privileges*, which allows more control over what each user is capable of doing. To properly assign those privileges, start by identifying each type of user:

Doctor

Should be able to read and write most of the graph, but be prevented from reading the patients' address. Has the permission to save *diagnoses* to the database, but not expand the schema with new concepts.

Receptionist

Should be able to read and write all patient data, but not be able to see the symptoms, diseases, or diagnoses.

Researcher

Should be able to perform statistical analysis of all data, except patients' personal information, to which they should have restricted access. To illustrate two different ways of setting up the same effective privileges, two roles are created for comparison.

Nurse

Should be able to perform all tasks that both the doctor and the receptionist can do. Granting both roles

(doctor and receptionist) to the nurse does not work as expected. This is explained in the section dedicated to the creation of the `nurse` role.

Junior nurse

While the senior nurse is able to save diagnoses just as a doctor can, some (junior) nurses might not be allowed to do that. Creating another role from scratch is an option, but the same output can be achieved by combining the `nurse` role with a new `disableDiagnoses` role that specifically restricts that activity.

IT administrator

This role is very similar to the built-in `admin` role, except that it should not allow access to the patients' `SSN` or be able to save a diagnosis, a privilege restricted to medical professionals. To achieve this, the built-in `admin` role can be copied and modified accordingly.

User manager

This user should have similar access as the IT administrator, but with more restrictions. To achieve that, a new role can be created from scratch and only specific administrative capabilities can be assigned to it.

Before creating the new roles and assigning them to Alice, Bob, Daniel, Charlie, and Tina, it is important to define the privileges each role should have. Since all users need `ACCESS` privilege to the `healthcare` database, this can be set through the `PUBLIC` role instead of all the individual roles:

Run the following command against the `system` database:

```
GRANT ACCESS ON DATABASE healthcare TO PUBLIC;
```

Privileges of `itadmin`

This role can be created as a copy of the built-in `admin` role:

```
CREATE ROLE itadmin AS COPY OF admin;
```

Then you need to **deny** the two specific actions this role is not supposed to perform:

- Read any patients' social security number (`SSN`).
- Submit medical diagnoses.

As well as the ability for the `itadmin` to amend their own privileges.

```
DENY READ {ssn} ON GRAPH healthcare NODES Patient TO itadmin;
DENY CREATE ON GRAPH healthcare RELATIONSHIPS DIAGNOSIS TO itadmin;
DENY ROLE MANAGEMENT ON DBMS TO itadmin;
DENY PRIVILEGE MANAGEMENT ON DBMS TO itadmin;
```

The complete set of privileges available to users assigned the `itadmin` role can be viewed using the following command:

```
SHOW ROLE itadmin PRIVILEGES AS COMMANDS;
```

▼ *Result*

```
+-------------------------------------------------------------------------+
| command                                                                 |
+-------------------------------------------------------------------------+
| "DENY CREATE ON GRAPH `healthcare` RELATIONSHIP DIAGNOSIS TO `itadmin`" |
| "DENY PRIVILEGE MANAGEMENT ON DBMS TO `itadmin`"                        |
| "DENY READ {ssn} ON GRAPH `healthcare` NODE Patient TO `itadmin`"       |
| "DENY ROLE MANAGEMENT ON DBMS TO `itadmin`"                             |
| "GRANT ACCESS ON DATABASE * TO `itadmin`"                               |
| "GRANT ALL DBMS PRIVILEGES ON DBMS TO `itadmin`"                        |
| "GRANT CONSTRAINT MANAGEMENT ON DATABASE * TO `itadmin`"                |
| "GRANT INDEX MANAGEMENT ON DATABASE * TO `itadmin`"                     |
| "GRANT LOAD ON ALL DATA TO `itadmin`"                                   |
| "GRANT MATCH {*} ON GRAPH * NODE * TO `itadmin`"                        |
| "GRANT MATCH {*} ON GRAPH * RELATIONSHIP * TO `itadmin`"                |
| "GRANT NAME MANAGEMENT ON DATABASE * TO `itadmin`"                      |
| "GRANT SHOW CONSTRAINT ON DATABASE * TO `itadmin`"                      |
| "GRANT SHOW INDEX ON DATABASE * TO `itadmin`"                           |
| "GRANT START ON DATABASE * TO `itadmin`"                                |
| "GRANT STOP ON DATABASE * TO `itadmin`"                                 |
| "GRANT TRANSACTION MANAGEMENT (*) ON DATABASE * TO `itadmin`"           |
| "GRANT WRITE ON GRAPH * TO `itadmin`"                                   |
+-------------------------------------------------------------------------+
18 rows
ready to start consuming query after 29 ms, results consumed after another 1 ms
```

Note:

Privileges that were granted or denied earlier can be revoked using the `REVOKE` command.

To provide the IT administrator `tina` these privileges, they must be assigned the new role `itadmin`. Run the following command against the `system` database:

```
GRANT ROLE itadmin TO tina;
```

To demonstrate that Tina is not able to see the patients' `SSN`, you can log into Cypher Shell as `tina` and run the following query against the `healthcare` database:

```
MATCH (n:Patient)
 WHERE n.dateOfBirth < date('1972-06-12')
RETURN n.name, n.ssn, n.address, n.dateOfBirth;
```

Result

```
+-----------------------------------------------------------------+
| n.name           | n.ssn | n.address                | n.dateOfBirth |
+-----------------------------------------------------------------+
| "Mark Jackson"   | NULL  | "1 secret way, downtown" | 1970-11-29    |
| "Joe Svensson"   | NULL  | "1 secret way, downtown" | 1972-02-12    |
| "Bob Anderson"   | NULL  | "1 secret way, downtown" | 1970-04-27    |
| "Sally Anderson" | NULL  | "1 secret way, downtown" | 1970-12-02    |
| "Ally Anderson"  | NULL  | "1 secret way, downtown" | 1972-05-20    |
| "Jane Svensson"  | NULL  | "1 secret way, downtown" | 1970-05-24    |
| "Sally Anderson" | NULL  | "1 secret way, downtown" | 1972-01-23    |
+-----------------------------------------------------------------+

7 rows
ready to start consuming query after 49 ms, results consumed after another 2 ms
```

The results make it seem as if these nodes do not even have an `SSN` field. This is a key feature of the security model: users cannot tell the difference between data that does not exist and data that is hidden using fine-grained read privileges.

Now recall that the `itadmin` role was denied the ability to save diagnoses (as this is a critical medical function reserved for only doctors and senior medical staff), you can test this by trying to create `DIAGNOSIS` relationships:

```
MATCH (n:Patient), (d:Disease)
CREATE (n)-[:DIAGNOSIS]->(d);
```

Result

```
Create relationship with type 'DIAGNOSIS' on database 'healthcare' is not allowed for user 'tina' with
roles [PUBLIC, itadmin].
```

> **Note:**
>
> Restrictions to reading data do not result in errors, they only make it appear as if the data is not there. However, restrictions on updating the graph do output an appropriate error when the user attempts to perform an action they are not allowed to.

Privileges of `researcher`

The researcher Charlie was previously a read-only user. To assign them the desired permissions, you can do something similar to what was done with the `itadmin` role, this time copying and modifying the `reader` role.

Another way to do it is by creating a new role from scratch and then either granting or denying a list of privileges:

- **Denying privileges**:

  You can grant the role `researcher` the ability to find all nodes and read all properties (much like the `reader` role), but deny read access to the `Patient` properties. This way, the researcher is unable to see patients' information such as `name`, `SSN`, and `address`. This approach has a problem though: if more properties are added to the `Patient` nodes *after* the restrictions were assigned to the `researcher` role, these new properties will automatically be visible to the researcher — a possibly undesirable outcome.

  To avoid that, you can rather deny *specific* privileges by running the following commands against the `system` database. You must be logged in as a user with the `admin` role to be able to execute these commands:

  ```
  // First create the role
  CREATE ROLE researcherB;
  // Then grant access to everything
  GRANT MATCH {*}
      ON GRAPH healthcare
      TO researcherB;
  // And deny read on specific node properties
  DENY READ {name, address, ssn}
      ON GRAPH healthcare
  ```

```
    NODES Patient
    TO researcherB;
// And finally deny traversal of the doctors diagnosis
DENY TRAVERSE
    ON GRAPH healthcare
    RELATIONSHIPS DIAGNOSIS
    TO researcherB;
```

- Granting privileges:

Another alternative is to only provide specific access to the properties the researcher is allowed to see. This way, the addition of new properties (for instance, to a `Patient` node) does not automatically make them visible to users assigned with this role. In case you wish to make them visible though, you need to explicitly grant read access. Run the following commands against the `system` database. You must be logged in as a user with the `admin` role to be able to execute these commands:

```
// Create the role first
CREATE ROLE researcherW
// Allow the researcher to find all nodes
GRANT TRAVERSE
    ON GRAPH healthcare
    NODES *
    TO researcherW;
// Now only allow the researcher to traverse specific relationships
GRANT TRAVERSE
    ON GRAPH healthcare
    RELATIONSHIPS HAS, OF
    TO researcherW;
// Allow reading of all properties of medical metadata
GRANT READ {*}
    ON GRAPH healthcare
    NODES Symptom, Disease
    TO researcherW;
// Allow reading of all properties of the disease-symptom relationship
GRANT READ {*}
    ON GRAPH healthcare
    RELATIONSHIPS OF
    TO researcherW;
// Only allow reading dateOfBirth for research purposes
GRANT READ {dateOfBirth}
    ON GRAPH healthcare
    NODES Patient
    TO researcherW;
```

In order to test that the researcher Charlie now has the specified privileges, assign them the `researcherB` role (with specifically denied privileges):

```
GRANT ROLE researcherB TO charlie;
```

You can also use a version of the `SHOW PRIVILEGES` command to see Charlie's access rights, which are a combination of those assigned to the `researcherB` and `PUBLIC` roles:

```
SHOW USER charlie PRIVILEGES AS COMMANDS;
```

Result

```
+-------------------------------------------------------------------------+
| command                                                                 |
+-------------------------------------------------------------------------+
| "DENY READ {address} ON GRAPH `healthcare` NODE Patient TO $role"       |
| "DENY READ {name} ON GRAPH `healthcare` NODE Patient TO $role"          |
| "DENY READ {ssn} ON GRAPH `healthcare` NODE Patient TO $role"           |
```

```
| "DENY TRAVERSE ON GRAPH `healthcare` RELATIONSHIP DIAGNOSIS TO $role" |
| "GRANT ACCESS ON DATABASE `healthcare` TO $role"                      |
| "GRANT ACCESS ON HOME DATABASE TO $role"                              |
| "GRANT EXECUTE FUNCTION * ON DBMS TO $role"                           |
| "GRANT EXECUTE PROCEDURE * ON DBMS TO $role"                          |
| "GRANT LOAD ON ALL DATA TO $role"                                     |
| "GRANT MATCH {*} ON GRAPH `healthcare` NODE * TO $role"               |
| "GRANT MATCH {*} ON GRAPH `healthcare` RELATIONSHIP * TO $role"       |
+-----------------------------------------------------------------------+

11 rows
ready to start consuming query after 17 ms, results consumed after another 2 ms
```

Now when Charlie logs into Cypher Shell and tries to execute the following command against the `healthcare` database, even though the command is similar to the one previously used by the `itadmin`, they will see different results:

```
MATCH (n:Patient)
 WHERE n.dateOfBirth < date('1972-06-12')
RETURN n.name, n.ssn, n.address, n.dateOfBirth;
```

Result

```
+-------------------------------------------+
| n.name | n.ssn | n.address | n.dateOfBirth |
+-------------------------------------------+
| NULL   | NULL  | NULL      | 1970-11-29    |
| NULL   | NULL  | NULL      | 1972-02-12    |
| NULL   | NULL  | NULL      | 1970-04-27    |
| NULL   | NULL  | NULL      | 1970-12-02    |
| NULL   | NULL  | NULL      | 1972-05-20    |
| NULL   | NULL  | NULL      | 1970-05-24    |
| NULL   | NULL  | NULL      | 1972-01-23    |
+-------------------------------------------+

7 rows
ready to start consuming query after 5 ms, results consumed after another 4 ms
```

Only the date of birth is available, so that the researcher Charlie may perform statistical analysis, for example. Another query Charlie could try is to find the ten diseases a patient younger than 25 is most likely to be diagnosed with, listed by probability:

```
WITH datetime() - duration({years:25}) AS timeLimit
MATCH (n:Patient)
WHERE n.dateOfBirth > date(timeLimit)
MATCH (n)-[h:HAS]->(s:Symptom)-[o:OF]->(d:Disease)
WITH d.name AS disease, o.probability AS prob
RETURN disease, sum(prob) AS score ORDER BY score DESC LIMIT 10;
```

Result

```
+-----------------------------------------+
| disease               | score           |
+-----------------------------------------+
| "Acute Placeboitis"   | 98.08269474672981 |
| "Chronic Whatitis"    | 92.7601237335886  |
| "Acute Otheritis"     | 87.61578906815608 |
| "Chronic Someitis"    | 81.68350008637253 |
| "Chronic Placeboitis" | 81.18800771016768 |
| "Acute Argitis"       | 80.94323685188083 |
| "Chronic Argitis"     | 80.06685163653665 |
| "Chronic Otheritis"   | 76.06538667789484 |
| "Acute Yellowitis"    | 70.74589062185173 |
| "Acute Someitis"      | 70.3238679154795  |
+-----------------------------------------+
```

```
10 rows
ready to start consuming query after 171 ms, results consumed after another 23 ms
```

If the `researcherB` role is revoked to Charlie, but `researcherW` is granted, when re-running these queries, the same results will be obtained.

Note:

Privileges that were granted or denied earlier can be revoked using the `REVOKE` command.

Privileges of `doctor`

Doctors should be given the ability to read and write almost everything, except the patients' `address` property, for instance. This role can be built from scratch by assigning full read and write access, and then specifically denying access to the `address` property. Switch to the `system` database and run the following commands:

```
CREATE ROLE doctor;
GRANT TRAVERSE ON GRAPH healthcare TO doctor;
GRANT READ {*} ON GRAPH healthcare TO doctor;
GRANT WRITE ON GRAPH healthcare TO doctor;
DENY READ {address} ON GRAPH healthcare NODES Patient TO doctor;
DENY SET PROPERTY {address} ON GRAPH healthcare NODES Patient TO doctor;
```

To allow the doctor Alice to have these privileges, grant the user `alice` this new role:

```
GRANT ROLE doctor TO alice;
```

To demonstrate that Alice is not able to see patient addresses, log in as `alice` and run the following query against the `healthcare` database:

```
MATCH (n:Patient)
 WHERE n.dateOfBirth < date('1972-06-12')
RETURN n.name, n.ssn, n.address, n.dateOfBirth;
```

Result

```
+-----------------------------------------------------+
| n.name           | n.ssn   | n.address | n.dateOfBirth |
+-----------------------------------------------------+
| "Mark Jackson"   | 1234578 | NULL      | 1970-11-29    |
| "Joe Svensson"   | 1234579 | NULL      | 1972-02-12    |
| "Bob Anderson"   | 1234597 | NULL      | 1970-04-27    |
| "Sally Anderson" | 1234617 | NULL      | 1970-12-02    |
| "Ally Anderson"  | 1234622 | NULL      | 1972-05-20    |
| "Jane Svensson"  | 1234644 | NULL      | 1970-05-24    |
| "Sally Anderson" | 1234657 | NULL      | 1972-01-23    |
+-----------------------------------------------------+

7 rows
ready to start consuming query after 5 ms, results consumed after another 3 ms
```

As a result, the doctor has the expected privileges, including being able to see the patients' `SSN`, but not

their address.

The doctor is also able to see all other node types:

```
MATCH (n) WITH labels(n) AS labels
RETURN labels, count(*);
```

Result

```
+-----------------------+
| labels      | count(*) |
+-----------------------+
| ["Symptom"] | 10       |
| ["Disease"] | 12       |
| ["Patient"] | 101      |
+-----------------------+

3 rows
ready to start consuming query after 29 ms, results consumed after another 1 ms
```

In addition, the doctor can traverse the graph, finding symptoms and diseases connected to patients:

```
MATCH (n:Patient)-[:HAS]->(s:Symptom)-[:OF]->(d:Disease)
  WHERE n.ssn = 1234657
RETURN n.name, d.name, count(s) AS score ORDER BY score DESC;
```

The resulting table shows which are the most likely diagnoses based on symptoms. The doctor can use this table to facilitate further questioning and testing of the patient in order to decide on the final diagnosis.

Result

```
+-------------------------------------------------+
| n.name           | d.name                | score |
+-------------------------------------------------+
| "Sally Anderson" | "Acute Placeboitis"   | 4     |
| "Sally Anderson" | "Chronic Argitis"     | 2     |
| "Sally Anderson" | "Acute Argitis"       | 2     |
| "Sally Anderson" | "Acute Otheritis"     | 2     |
| "Sally Anderson" | "Chronic Placeboitis" | 2     |
| "Sally Anderson" | "Acute Yellowitis"    | 2     |
| "Sally Anderson" | "Chronic Whatitis"    | 2     |
| "Sally Anderson" | "Chronic Yellowitis"  | 2     |
| "Sally Anderson" | "Acute Whatitis"      | 1     |
| "Sally Anderson" | "Acute Someitis"      | 1     |
| "Sally Anderson" | "Chronic Someitis"    | 1     |
| "Sally Anderson" | "Chronic Otheritis"   | 1     |
+-------------------------------------------------+

12 rows
ready to start consuming query after 48 ms, results consumed after another 2 ms
```

Once the doctor has investigated further, they would be able to decide on the diagnosis and save that result to the database:

```
WITH datetime({epochmillis:timestamp()}) AS now
WITH now, date(now) as today
MATCH (p:Patient)
  WHERE p.ssn = 1234657
MATCH (d:Disease)
  WHERE d.name = "Chronic Placeboitis"
MERGE (p)-[i:DIAGNOSIS {by: 'Alice'}]->(d)
```

```
  ON CREATE SET i.created_at = now, i.updated_at = now, i.date = today
  ON MATCH SET i.updated_at = now
RETURN p.name, d.name, i.by, i.date, duration.between(i.created_at, i.updated_at) AS updated;
```

This allows the doctor to record their diagnosis as well as take note of previous diagnoses:

Result

```
+-------------------------------------------------------------------------+
| p.name           | d.name                | i.by    | i.date     | updated |
+-------------------------------------------------------------------------+
| "Sally Anderson" | "Chronic Placeboitis" | "Alice" | 2025-02-14 | PT0S    |
+-------------------------------------------------------------------------+

1 row
ready to start consuming query after 73 ms, results consumed after another 6 ms
Created 1 relationships, Set 4 properties
```

Note:

Creating the `DIAGNOSIS` relationship for the first time requires the privilege to create new types. This is also true for the property names `doctor`, `created_at`, and `updated_at`. It can be fixed by either granting the doctor `NAME MANAGEMENT` privileges or by pre-creating the missing types. The latter would be more precise and can be achieved by running, as an administrator, the procedures `db.createRelationshipType` and `db.createProperty` with appropriate arguments.

Privileges of `receptionist`

Receptionists should only be able to manage patient information. They are not allowed to find or read any other parts of the graph. In addition, they should be able to create and delete patients, but not any other nodes. Switch to the `system` database and run the following commands:

```
CREATE ROLE receptionist;
GRANT MATCH {*} ON GRAPH healthcare NODES Patient TO receptionist;
GRANT CREATE ON GRAPH healthcare NODES Patient TO receptionist;
GRANT DELETE ON GRAPH healthcare NODES Patient TO receptionist;
GRANT SET PROPERTY {*} ON GRAPH healthcare NODES Patient TO receptionist;
```

It would have been simpler to grant global `WRITE` privileges to the receptionist Bob. However, this would have the unfortunate side effect of allowing them the ability to create other nodes, like new `Symptom` nodes, even though they would subsequently be unable to find or read those same nodes. While there are use cases in which it is desirable to have roles able to create data they cannot read, that is not the case of this model.

With that in mind, grant the receptionist Bob their new `receptionist` role:

```
GRANT ROLE receptionist TO bob;
```

With these privileges, if Bob tries to read the entire database, he will still only see the patients:

```
MATCH (n) WITH labels(n) AS labels
RETURN labels, count(*);
```

Result

```
+-----------------------+
| labels      | count(*) |
+-----------------------+
| ["Patient"] | 101      |
+-----------------------+

1 row
ready to start consuming query after 2 ms, results consumed after another 3 ms
```

However, Bob is able to see all fields of the patients' records:

```
MATCH (n:Patient)
 WHERE n.dateOfBirth < date('1972-06-12')
RETURN n.name, n.ssn, n.address, n.dateOfBirth;
```

Result

```
+-----------------------------------------------------------------------+
| n.name           | n.ssn   | n.address                | n.dateOfBirth |
+-----------------------------------------------------------------------+
| "Mark Jackson"   | 1234578 | "1 secret way, downtown" | 1970-11-29    |
| "Joe Svensson"   | 1234579 | "1 secret way, downtown" | 1972-02-12    |
| "Bob Anderson"   | 1234597 | "1 secret way, downtown" | 1970-04-27    |
| "Sally Anderson" | 1234617 | "1 secret way, downtown" | 1970-12-02    |
| "Ally Anderson"  | 1234622 | "1 secret way, downtown" | 1972-05-20    |
| "Jane Svensson"  | 1234644 | "1 secret way, downtown" | 1970-05-24    |
| "Sally Anderson" | 1234657 | "1 secret way, downtown" | 1972-01-23    |
+-----------------------------------------------------------------------+

7 rows
ready to start consuming query after 2 ms, results consumed after another 1 ms
```

With the `receptionist` role, Bob can delete any new patient nodes they have just created, but they are not able to delete patients that have already received diagnoses since those are connected to parts of the graph that Bob cannot see. Here is a demonstration of both scenarios:

```
CREATE (n:Patient {
  ssn:87654321,
  name: 'Another Patient',
  email: 'another@example.com',
  address: '1 secret way, downtown',
  dateOfBirth: date('2001-01-20')
})
RETURN n.name, n.dateOfBirth;
```

Result

```
+-----------------------------------+
| n.name            | n.dateOfBirth |
+-----------------------------------+
| "Another Patient" | 2001-01-20    |
+-----------------------------------+

1 row
ready to start consuming query after 36 ms, results consumed after another 1 ms
Added 1 nodes, Set 5 properties, Added 1 labels
```

The receptionist is able to modify any patient record:

```
MATCH (n:Patient)
```

```
WHERE n.ssn = 87654321
SET n.address = '2 streets down, uptown'
RETURN n.name, n.dateOfBirth, n.address;
```

Result

```
+-------------------------------------------------------------+
| n.name            | n.dateOfBirth | n.address               |
+-------------------------------------------------------------+
| "Another Patient" | 2001-01-20    | "2 streets down, uptown" |
+-------------------------------------------------------------+

1 row
ready to start consuming query after 22 ms, results consumed after another 3 ms
Set 1 properties
```

The receptionist is also able to delete this recently created patient because it is not connected to any other records:

```
MATCH (n:Patient)
 WHERE n.ssn = 87654321
DETACH DELETE n;
```

Result

```
0 rows
ready to start consuming query after 17 ms, results consumed after another 0 ms
Deleted 1 nodes
```

However, if the receptionist attempts to delete a patient that has existing diagnoses, this will fail:

```
MATCH (n:Patient)
 WHERE n.ssn = 1234610
DETACH DELETE n;
```

Result

```
Cannot delete node<65>, because it still has relationships. To delete this node, you must first delete its
relationships.
```

The reason why this query fails is that, while Bob can find the `(:Patient)` node, he does not have sufficient traverse rights to find nor the rights to delete the outgoing relationships from it.

Either they need to ask Tina the `itadmin` for help for this task, or more privileges can be added to the `receptionist` role. Switch to the `system` database and run the following commands:

```
GRANT TRAVERSE ON GRAPH healthcare NODES Symptom, Disease TO receptionist;
GRANT TRAVERSE ON GRAPH healthcare RELATIONSHIPS HAS, DIAGNOSIS TO receptionist;
GRANT DELETE ON GRAPH healthcare RELATIONSHIPS HAS, DIAGNOSIS TO receptionist;
```

Note:

Privileges that were granted or denied earlier can be revoked using the `REVOKE` command.

Privileges of `nurse`

Nurses should have the capabilities of both doctors and receptionists, but assigning them both the `doctor` and `receptionist` roles might not have the expected effect. If those two roles were created with `GRANT` privileges only, combining them would be simply cumulative. But if the `doctor` role contains some `DENY` privileges, these always overrule `GRANT`. This means that the nurse will still have the same restrictions as a doctor, which is not what is intended here.

To demonstrate this, you can assign the `doctor` role to the nurse Daniel. Switch to the `system` database and run the following commands:

```
GRANT ROLE doctor, receptionist TO daniel;
```

Daniel should now have a combined set of privileges:

```
SHOW USER daniel PRIVILEGES AS COMMANDS;
```

Result

```
+-------------------------------------------------------------------------+
| command                                                                 |
+-------------------------------------------------------------------------+
| "DENY READ {address} ON GRAPH `healthcare` NODE Patient TO $role"       |
| "DENY SET PROPERTY {address} ON GRAPH `healthcare` NODE Patient TO $role" |
| "GRANT ACCESS ON DATABASE `healthcare` TO $role"                        |
| "GRANT ACCESS ON HOME DATABASE TO $role"                                |
| "GRANT CREATE ON GRAPH `healthcare` NODE Patient TO $role"              |
| "GRANT DELETE ON GRAPH `healthcare` NODE Patient TO $role"              |
| "GRANT DELETE ON GRAPH `healthcare` RELATIONSHIP DIAGNOSIS TO $role"    |
| "GRANT DELETE ON GRAPH `healthcare` RELATIONSHIP HAS TO $role"          |
| "GRANT EXECUTE FUNCTION * ON DBMS TO $role"                             |
| "GRANT EXECUTE PROCEDURE * ON DBMS TO $role"                            |
| "GRANT LOAD ON ALL DATA TO $role"                                       |
| "GRANT MATCH {*} ON GRAPH `healthcare` NODE Patient TO $role"           |
| "GRANT READ {*} ON GRAPH `healthcare` NODE * TO $role"                  |
| "GRANT READ {*} ON GRAPH `healthcare` RELATIONSHIP * TO $role"          |
| "GRANT SET PROPERTY {*} ON GRAPH `healthcare` NODE Patient TO $role"    |
| "GRANT TRAVERSE ON GRAPH `healthcare` NODE * TO $role"                  |
| "GRANT TRAVERSE ON GRAPH `healthcare` NODE Disease TO $role"            |
| "GRANT TRAVERSE ON GRAPH `healthcare` NODE Symptom TO $role"            |
| "GRANT TRAVERSE ON GRAPH `healthcare` RELATIONSHIP * TO $role"          |
| "GRANT TRAVERSE ON GRAPH `healthcare` RELATIONSHIP DIAGNOSIS TO $role"  |
| "GRANT TRAVERSE ON GRAPH `healthcare` RELATIONSHIP HAS TO $role"        |
| "GRANT WRITE ON GRAPH `healthcare` TO $role"                            |
+-------------------------------------------------------------------------+

22 rows
ready to start consuming query after 10 ms, results consumed after another 1 ms
```

Note:

Privileges that were granted or denied earlier can be revoked using the `REVOKE` command.

Now the intention is that a nurse can perform the actions of a receptionist, which means they should be able to read and write the `address` field of the `Patient` nodes. Log in as `daniel` and run the following query against the `healthcare` database:

```
MATCH (n:Patient)
```

```
 WHERE n.dateOfBirth < date('1972-06-12')
RETURN n.name, n.ssn, n.address, n.dateOfBirth;
```

Result

```
+-----------------------------------------------------+
| n.name           | n.ssn   | n.address | n.dateOfBirth |
+-----------------------------------------------------+
| "Mark Jackson"   | 1234578 | NULL      | 1970-11-29    |
| "Joe Svensson"   | 1234579 | NULL      | 1972-02-12    |
| "Bob Anderson"   | 1234597 | NULL      | 1970-04-27    |
| "Sally Anderson" | 1234617 | NULL      | 1970-12-02    |
| "Ally Anderson"  | 1234622 | NULL      | 1972-05-20    |
| "Jane Svensson"  | 1234644 | NULL      | 1970-05-24    |
| "Sally Anderson" | 1234657 | NULL      | 1972-01-23    |
+-----------------------------------------------------+

7 rows
ready to start consuming query after 4 ms, results consumed after another 2 ms
```

As expected, the `address` field is invisible to the nurse. This happens because, as previously described, `DENY` privileges *always* overrule `GRANT`. Since both roles `doctor` and `receptionist` were assigned to the nurse, the `DENIED` privileges of the `doctor` role are overruling the `GRANTED` privileges of the `receptionist`. Even if the nurse tries to write the address field, they would receive an error, and that is not what is desired here. To correct that, you can:

- Redefine the `doctor` role with only grants and define each `Patient` property the doctor should be able to read.
- Redefine the `nurse` role with the actual intended behavior.

The second option is simpler if you consider that the nurse is essentially the doctor without the `address` restrictions. In this case, you need to create a `nurse` role from scratch. Switch to the `system` database and run the following commands:

```
CREATE ROLE nurse;
GRANT TRAVERSE ON GRAPH healthcare TO nurse;
GRANT READ {*} ON GRAPH healthcare TO nurse;
GRANT WRITE ON GRAPH healthcare TO nurse;
```

Now you assign the `nurse` role to the nurse Daniel, but remember to revoke the `doctor` and the `receptionist` roles so there are no privileges being overridden:

```
REVOKE ROLE doctor FROM daniel;
REVOKE ROLE receptionist FROM daniel;
GRANT ROLE nurse TO daniel;
```

This time, when the nurse Daniel logs in and runs the following query against the `healthcare` database, they will see the `address` fields:

```
MATCH (n:Patient)
 WHERE n.dateOfBirth < date('1972-06-12')
RETURN n.name, n.ssn, n.address, n.dateOfBirth;
```

Result

```
+-----------------------------------------------------------------------+
```

```
| n.name           | n.ssn   | n.address                | n.dateOfBirth |
+-----------------------------------------------------------------------+
| "Mark Jackson"   | 1234578 | "1 secret way, downtown" | 1970-11-29    |
| "Joe Svensson"   | 1234579 | "1 secret way, downtown" | 1972-02-12    |
| "Bob Anderson"   | 1234597 | "1 secret way, downtown" | 1970-04-27    |
| "Sally Anderson" | 1234617 | "1 secret way, downtown" | 1970-12-02    |
| "Ally Anderson"  | 1234622 | "1 secret way, downtown" | 1972-05-20    |
| "Jane Svensson"  | 1234644 | "1 secret way, downtown" | 1970-05-24    |
| "Sally Anderson" | 1234657 | "1 secret way, downtown" | 1972-01-23    |
+-----------------------------------------------------------------------+

7 rows
ready to start consuming query after 4 ms, results consumed after another 2 ms
```

The other main action that the `nurse` role should be able to perform is the primary `doctor` action of saving a diagnosis to the database:

```
WITH date(datetime({epochmillis:timestamp()})) AS today
MATCH (p:Patient)
  WHERE p.ssn = 1234657
MATCH (d:Disease)
  WHERE d.name = "Chronic Placeboitis"
MERGE (p)-[i:DIAGNOSIS {by: 'Daniel'}]->(d)
  ON CREATE SET i.date = today
RETURN p.name, d.name, i.by, i.date;
```

Result

```
+-------------------------------------------------------------------+
| p.name           | d.name                | i.by     | i.date     |
+-------------------------------------------------------------------+
| "Sally Anderson" | "Chronic Placeboitis" | "Daniel" | 2025-02-14 |
+-------------------------------------------------------------------+

1 row
ready to start consuming query after 49 ms, results consumed after another 2 ms
Created 1 relationships, Set 2 properties
```

Performing this action, otherwise reserved for the `doctor` role, involves more responsibility for the `nurse`. There might be nurses that should not be entrusted with this option, which is why you can divide the `nurse` role into *senior* and *junior* nurses, for example. Currently, Daniel is a senior nurse.

Privileges of *junior* `nurse`

Previously, creating the `nurse` role by combining the `doctor` and `receptionist` roles led to an undesired scenario as the `DENIED` privileges of the `doctor` role overrode the `GRANTED` privileges of the `receptionist`. In that case, the objective was to enhance the permissions of the *senior* nurse, but when it comes to the *junior* nurse, they should be able to perform the same actions as the *senior*, except adding diagnoses to the database.

To achieve this, you can create a special role that contains specifically only the additional restrictions. Switch to the `system` database and run the following commands:

```
CREATE ROLE disableDiagnoses;
DENY CREATE ON GRAPH healthcare RELATIONSHIPS DIAGNOSIS TO disableDiagnoses;
```

And then assign this new role to the nurse Daniel, so you can test the behavior:

```
GRANT ROLE disableDiagnoses TO daniel;
```

If you check what privileges Daniel has now, it is the combination of the two roles `nurse` and `disableDiagnoses`:

```
SHOW USER daniel PRIVILEGES AS COMMANDS;
```

Result

```
+-------------------------------------------------------------------+
| command                                                           |
+-------------------------------------------------------------------+
| "DENY CREATE ON GRAPH `healthcare` RELATIONSHIP DIAGNOSIS TO $role" |
| "GRANT ACCESS ON DATABASE `healthcare` TO $role"                  |
| "GRANT ACCESS ON HOME DATABASE TO $role"                          |
| "GRANT EXECUTE FUNCTION * ON DBMS TO $role"                       |
| "GRANT EXECUTE PROCEDURE * ON DBMS TO $role"                      |
| "GRANT LOAD ON ALL DATA TO $role"                                 |
| "GRANT READ {*} ON GRAPH `healthcare` NODE * TO $role"            |
| "GRANT READ {*} ON GRAPH `healthcare` RELATIONSHIP * TO $role"    |
| "GRANT TRAVERSE ON GRAPH `healthcare` NODE * TO $role"            |
| "GRANT TRAVERSE ON GRAPH `healthcare` RELATIONSHIP * TO $role"    |
| "GRANT WRITE ON GRAPH `healthcare` TO $role"                      |
+-------------------------------------------------------------------+

11 rows
ready to start consuming query after 4 ms, results consumed after another 1 ms
```

Daniel can still see the address fields, and can even perform the diagnosis investigation that the `doctor` can perform:

```
MATCH (n:Patient)-[:HAS]->(s:Symptom)-[:OF]->(d:Disease)
WHERE n.ssn = 1234650
RETURN n.ssn, n.name, d.name, count(s) AS score ORDER BY score DESC;
```

Result

```
+-----------------------------------------------------+
| n.ssn   | n.name       | d.name                | score |
+-----------------------------------------------------+
| 1234650 | "Mark Smith" | "Chronic Argitis"     | 3     |
| 1234650 | "Mark Smith" | "Chronic Otheritis"   | 3     |
| 1234650 | "Mark Smith" | "Acute Placeboitis"   | 3     |
| 1234650 | "Mark Smith" | "Chronic Yellowitis"  | 3     |
| 1234650 | "Mark Smith" | "Chronic Someitis"    | 3     |
| 1234650 | "Mark Smith" | "Chronic Whatitis"    | 2     |
| 1234650 | "Mark Smith" | "Acute Otheritis"     | 2     |
| 1234650 | "Mark Smith" | "Chronic Placeboitis" | 2     |
| 1234650 | "Mark Smith" | "Acute Argitis"       | 2     |
| 1234650 | "Mark Smith" | "Acute Someitis"      | 2     |
| 1234650 | "Mark Smith" | "Acute Whatitis"      | 1     |
| 1234650 | "Mark Smith" | "Acute Yellowitis"    | 1     |
+-----------------------------------------------------+

12 rows
ready to start consuming query after 50 ms, results consumed after another 1 ms
```

But when they try to save a diagnosis to the database, they will be denied that action:

```
WITH date(datetime({epochmillis:timestamp()})) AS today
MATCH (p:Patient)
  WHERE p.ssn = 1234650
```

```
MATCH (d:Disease)
  WHERE d.name = "Chronic Placeboitis"
MERGE (p)-[i:DIAGNOSIS {by: 'Daniel'}]->(d)
  ON CREATE SET i.date = today
RETURN p.name, d.name, i.by, i.date;
```

Result

```
+---------------------------------+
| p.name | d.name | i.by | i.date |
+---------------------------------+
Create relationship with type 'DIAGNOSIS' on database 'healthcare' is not allowed for user 'daniel' with
roles [PUBLIC, disableDiagnoses, nurse].
```

To promote Daniel back to senior nurse, revoke the role that introduced the restriction:

```
REVOKE ROLE disableDiagnoses FROM daniel;
```

Building a custom administrator role

The `itadmin` role was originally created by copying the built-in `admin` role and adding restrictions. However, there might be cases in which having `DENY`s can be less convenient than only having `GRANT`s. Instead, you can build the administrator role from the ground up.

The IT administrator Tina is able to create new users and assign them to the product roles as an `itadmin`, but you can create a more restricted role called `userManager` and grant it only the appropriate privileges. Switch to the `system` database and run the following commands:

```
CREATE ROLE userManager;
GRANT USER MANAGEMENT ON DBMS TO userManager;
GRANT ROLE MANAGEMENT ON DBMS TO userManager;
GRANT SHOW PRIVILEGE ON DBMS TO userManager;
```

Test the new behavior by revoking the `itadmin` role from Tina and grant them the `userManager` role instead:

```
REVOKE ROLE itadmin FROM tina;
GRANT ROLE userManager TO tina;
```

These are the privileges granted to `userManager`:

- `USER MANAGEMENT` allows creating, updating, and dropping users.
- `ROLE MANAGEMENT` allows creating, updating, and dropping roles as well as assigning roles to users.
- `SHOW PRIVILEGE` allows listing the users' privileges.

Listing Tina's new privileges should now show a much shorter list than when they were a more powerful administrator with the `itadmin` role:

```
SHOW USER tina PRIVILEGES AS COMMANDS;
```

Result

```
+-------------------------------------------------+
| command                                         |
+-------------------------------------------------+
| "GRANT ACCESS ON DATABASE `healthcare` TO $role" |
| "GRANT ACCESS ON HOME DATABASE TO $role"         |
| "GRANT EXECUTE FUNCTION * ON DBMS TO $role"      |
| "GRANT EXECUTE PROCEDURE * ON DBMS TO $role"     |
| "GRANT LOAD ON ALL DATA TO $role"                |
| "GRANT ROLE MANAGEMENT ON DBMS TO $role"         |
| "GRANT SHOW PRIVILEGE ON DBMS TO $role"          |
| "GRANT USER MANAGEMENT ON DBMS TO $role"         |
+-------------------------------------------------+

8 rows
ready to start consuming query after 24 ms, results consumed after another 1 ms
```

> **Note:**
>
> No other privilege management privileges were granted here. How much power this role should have would depend on the requirements of the system. Refer to the section The `admin` role for a complete list of privileges to consider.

Now Tina should be able to create new users and assign them to roles (from the `system` database):

```
CREATE USER sally SET PASSWORD 'secretpassword' CHANGE REQUIRED;
GRANT ROLE receptionist TO sally;
SHOW USER sally PRIVILEGES AS COMMANDS;
```

Result

```
0 rows
ready to start consuming query after 45 ms, results consumed after another 0 ms
0 rows
ready to start consuming query after 4 ms, results consumed after another 0 ms
+--------------------------------------------------------------------------+
| command                                                                  |
+--------------------------------------------------------------------------+
| "GRANT ACCESS ON DATABASE `healthcare` TO $role"                         |
| "GRANT ACCESS ON HOME DATABASE TO $role"                                 |
| "GRANT CREATE ON GRAPH `healthcare` NODE Patient TO $role"               |
| "GRANT DELETE ON GRAPH `healthcare` NODE Patient TO $role"               |
| "GRANT DELETE ON GRAPH `healthcare` RELATIONSHIP DIAGNOSIS TO $role"     |
| "GRANT DELETE ON GRAPH `healthcare` RELATIONSHIP HAS TO $role"           |
| "GRANT EXECUTE FUNCTION * ON DBMS TO $role"                              |
| "GRANT EXECUTE PROCEDURE * ON DBMS TO $role"                             |
| "GRANT LOAD ON ALL DATA TO $role"                                        |
| "GRANT MATCH {*} ON GRAPH `healthcare` NODE Patient TO $role"            |
| "GRANT SET PROPERTY {*} ON GRAPH `healthcare` NODE Patient TO $role"     |
| "GRANT TRAVERSE ON GRAPH `healthcare` NODE Disease TO $role"             |
| "GRANT TRAVERSE ON GRAPH `healthcare` NODE Symptom TO $role"             |
| "GRANT TRAVERSE ON GRAPH `healthcare` RELATIONSHIP DIAGNOSIS TO $role"   |
| "GRANT TRAVERSE ON GRAPH `healthcare` RELATIONSHIP HAS TO $role"         |
+--------------------------------------------------------------------------+

15 rows
ready to start consuming query after 3 ms, results consumed after another 0 ms
```

Configuring Neo4j Single Sign-On (SSO)

Neo4j supports SSO authentication and authorization through identity providers implementing the OpenID

Connect (OIDC) standard. This page features detailed examples of how to configure Single Sign-On (SSO) for several identity providers. It also presents frequently asked questions and solutions to common problems encountered when configuring SSO.

> **Note:**
>
> The following configurations are crafted for a Neo4j Browser served on *http://localhost:7474/* (the default URL when starting the database on `localhost`).
>
> Therefore, when reproducing them in the identity providers, you must modify the redirect URI to include the URI serving your Neo4j Browser application. For example:
>
> ```
> http://localhost:7474/browser/?idp_id={provider}&auth_flow_step=redirect_uri
> ```

SSO works in the following way:

1. The server (Neo4j DBMS) contacts the identity provider (Okta, Entra ID, Google, etc.) and fetches the JSON Web Keys (JWKs) from the provider.
2. The client (e.g., Bloom, Neo4j Browser, etc.) asks the user for credentials and contacts the identity provider.
3. The identity provider responds with a JSON Web Token (JWT), a JSON file containing fields (claims) relative to the user (email, audience, groups, etc.).
4. The client provides the server with the JWT, and the server verifies its signature with the JWKs.
5. Introduced in 5.24 Optionally, you can control the authentication and authorization on a user level by setting `dbms.security.require_local_user` to `true` in the *neo4j.conf* file and the auth providers for the users, which authentication and authorization you want to control, using Cypher. This setting mandates that users with the relevant auth provider attached to them must exist in the database before they can authenticate and authorize with that auth provider. For information on how to modify or create users in this mode, see Manage users.
   This mode allows you the following using Cypher:
   a. Suspend SSO users.
   b. Set a home database for an SSO user.
   c. Set a user-friendly name for an SSO user (rather than relying on the external identifier).
   d. Set a password for an SSO user.
   e. Set a password change requirement for an SSO user.

For further information and examples, see Configure SSO at the user level using auth providers.

> **Important:**
>
> JWTs must **always** contain a value for `sub` even when using a different claim for `username`. It is the only claim guaranteed to be unique and stable. Other claims, such as `email` or `preferred_username` are less secure and may change over time. They should **not** be used for authentication. Neo4j may assign permissions to a user based on this username value in a

hybrid authorization configuration. Thus, changing the username claim from `sub` is not recommended.

Okta

The following examples show how to configure Okta for authentication and authorization using access tokens and ID tokens. It assumes that you are using Okta Developer Edition Service. For the complete guide on how to customize tokens returned from Okta with a groups claim, see the Okta official documentation.

Configure the client

1. From the left-hand side of the Okta dashboard, navigate to **Applications** and click **Create App Integration**.
2. Select **OIDC - OpenID Connect** for Sign-in method and **Single-Page Application** for Application type.
3. Click **Next**.
4. Configure the client with the appropriate redirect URI.
    a. Add a name for the app integration.
    b. Add the **Sign-in redirect URIs**, for example, *http://localhost:7474/browser/?idp_id=okta&auth_flow_step=redirect_uri*. This URI will accept returned token responses after successful authentication.
5. Add the **Sign-out redirect URIs**, for example, *http://localhost:7474/*.
6. In the **Assignments** section, select **Skip group assignment** for now.
7. Click **Save**.
8. Take note of the Client ID. You will need it later when configuring the Okta parameters and the Well-known OpenID Connect endpoint in the *neo4j.conf* file.

Assign Okta groups to the application

1. From the left-hand side of the Okta dashboard, navigate to **Dashboard → Directory → Groups**, and click **Add Group**.
2. Add a name for the group, for example, `engineers`, and click **Save**.
3. Click the group you just created and then click **Assign people**.
4. Add users to the group. Users can be added to a group either on user creation or by editing the group.
5. Assign the group to an application.
    a. Click **Applications** and then **Assign Applications**.
    b. Select the application you created earlier and click **Assign**.

Access token

This example shows how to configure Okta for authentication and authorization using access tokens and

how to configure Neo4j to use them.

Add a groups claim to access tokens

1. From the left-hand side of the Okta dashboard, navigate to **Security** → **API**.
2. Click the default authorization server (the one that shows `api://default` as audience) to return the `groups` claim in access tokens:
   a. On the **Claims** tab, click **Add Claim**.
   b. Add a claim with the name `groups`.
   c. From the **Value type** dropdown, select **Groups**.
   d. From the Filter dropdown, select **Matches regex** and the value `.*`.
   e. Click **Create**.

Configure Neo4j

1. Configure Neo4j to use Okta authentication by configuring the following settings in the *neo4j.conf* file:

```
dbms.security.authentication_providers=oidc-okta
dbms.security.authorization_providers=oidc-okta
dbms.security.oidc.okta.display_name=Okta
dbms.security.oidc.okta.auth_flow=pkce
dbms.security.oidc.okta.well_known_discovery_uri=https://dev-54101110.okta.com/oauth2/default/.well-known/oauth-authorization-server
dbms.security.oidc.okta.audience=api://default
dbms.security.oidc.okta.claims.username=sub
dbms.security.oidc.okta.claims.groups=groups
dbms.security.oidc.okta.params=client_id=0oao2rybx5hIERt5W5d7;response_type=code;scope=openid profile email
dbms.security.oidc.okta.authorization.group_to_role_mapping= "engineers" = admin; \
                                                             "collaborators" = reader
```

> **Note:**
>
> The `token_type_principal` and the `token_type_authentication` are omitted, meaning access tokens are used instead.

2. Log in with your Okta SSO credentials using the email of an `engineer` role user that results in an `admin` role in the database:

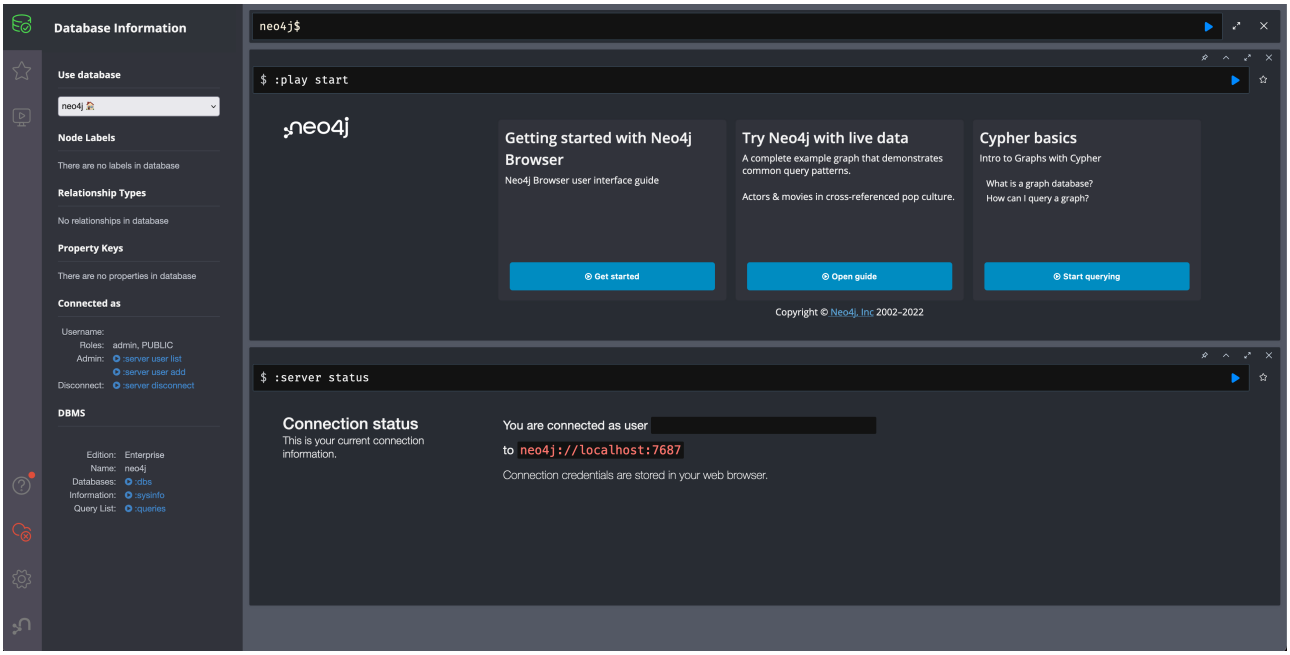


Figure 42. Okta OIDC successful login

ID token

This example shows how to configure Okta for authentication and authorization using ID tokens and the how to configure Neo4j to use them.

Add a groups claim to ID tokens

You can add a groups claim to ID tokens to configure authentication and authorization using ID tokens.

1. From the left-hand side of the Okta dashboard, navigate to **Security** → **API**.
2. Click the default authorization server (the one that shows `api://default` as audience) to return the `groups` claim in access tokens:
   a. On the **Claims** tab, click **Add Claim**.
   b. Add a claim with the name `groups`.
   c. From the **Include in token type** dropdown, select **ID Token**.
   d. From the **Value type** dropdown, select **Groups**.
   e. From the Filter dropdown, select **Matches regex** and the value `.*`.
   f. Click **Create**.
3. Add a claim with the name `userid` and the value type `User ID`.

> **Note:**
>
> The `userid` claim is not included in the ID token by default like the default `sub` claim for access tokens, thus you need to add it manually. The name you give to your claim needs to be also indicated in the configuration `dbms.security.oidc.okta.claims.username=userid` in the *neo4j.conf* file.

a. Click **Add Claim**.

b. Add a claim with the name `userid`.

c. From the **Include in token type** dropdown, select **ID Token**.

d. From the **Value type** dropdown, select **Expression**.

e. In the **Value** field, type `(appuser !=null) ? appuser.userName : app.clientId`.

f. Click **Create**.

Configure Neo4j

1. Configure Neo4j to use Okta authentication by configuring the following settings in the *neo4j.conf* file:

```
dbms.security.authentication_providers=oidc-okta, native
dbms.security.authorization_providers=oidc-okta
dbms.security.oidc.okta.display_name=Okta
dbms.security.oidc.okta.auth_flow=pkce
dbms.security.oidc.okta.well_known_discovery_uri=https://dev-54101110.okta.com/oauth2/default/.well-known/oauth-authorization-server
dbms.security.oidc.okta.audience=0oao2rybx5hIERt5W5d7
dbms.security.oidc.okta.claims.username=userid
dbms.security.oidc.okta.claims.groups=groups
dbms.security.oidc.okta.params=client_id=0oao2rybx5hIERt5W5d7;response_type=code;scope=openid profile email
dbms.security.oidc.okta.authorization.group_to_role_mapping="admin_group" = admin;
dbms.security.oidc.okta.config=token_type_principal=id_token;token_type_authentication=id_token
```

> **Tip:**
>
> You can find the audience parameter under **OpenID Connect ID Token** of your application on the **Sign On** tab.

2. Introduced in 5.24 (Optional) If you want control the authentication and authorization on a user level, configure `dbms.security.require_local_user` to `true` in the *neo4j.conf* file. This setting mandates that users with the relevant auth provider attached to them must exist in the database before they can authenticate and authorize with that auth provider. For information on how to create users in this mode, see Creating users.

   For example, to create the user `jake` who can authenticate using `native` or `okta`, and authorize using Okta (as configured in step 3), you can use the following Cypher query:

```
CREATE USER jake
SET HOME DATABASE 'jakesHomeDb'
SET AUTH 'oidc-okta' {SET ID 'jakesUniqueOktaUserId'} // `jakesUniqueOktaUserId` must match the value of the claim that you configured via dbms.security.oidc.okta.claims.username
SET AUTH 'native' {SET PASSWORD 'changeme' SET PASSWORD CHANGE REQUIRED}
```

   See Configure SSO at the user level using auth providers for further examples.

Microsoft Entra ID (formerly Azure Active Directory)

The following examples show how to configure Microsoft Entra ID for authentication and authorization using access tokens and ID tokens.

Register the application

1. Log in to the Azure portal.
2. Click **Microsoft Entra ID** and navigate to **Manage → App registrations**.
3. Click **New registration**.
4. Type a name for your application, for example, `Neo4j SSO`.
5. Under **Select the supported account types**, select `Accounts in this organizational directory only (Default Directory only - Single tenant)`.
6. Under **Redirect URI**, select `Single-page application (SPA)` and enter the redirect URI: *http://localhost:7474/browser/?idp_id=azure&auth_flow_step=redirect_uri* The redirect URI will accept the returned token responses after successful authentication.
7. Click **Register**.

Access token

This example shows how to configure Neo4j to use an Entra ID access token for authentication and authorization.

Configure Entra ID

1. From the **App registrations** page, select the app you just created.
2. From the left-hand side menu, navigate to **Manage → Token configuration**.
   a. Click **Add groups claim**.
   b. Select **Groups assigned to the application (recommended for large enterprise companies to avoid exceeding the limit on the number of groups a token can emit)** to include in your access token.
   c. Save your changes.
3. Navigate to **Expose an API** and click **Add a Scope**.

> **Note:**
>
> The first time you click the **Add a Scope** button, you see a new pane stating that you need to add an *Application ID URI* before proceeding. You can find it on your app **Overview** page. It is a GUID that looks like this: `api://<GUID>`. The GUID is a unique identifier for your application.

4. Click **Save and continue** after setting the *Application ID URI*.
5. Fill in all mandatory fields in the pane **Add a scope**.
   a. Enter a new **Scope name** (e.g., `access-token`), **Admin consent display name**, and **Admin consent description**.
   b. Make sure the **Enabled** scope state is selected.
   c. Select the **Add scope** button again to create a new scope. You can add all scopes supported by your API. Make a note of them for later.

Configure Neo4j

You can configure Neo4j to use Entra ID for authentication by configuring the following settings in the *neo4j.conf* file:

```
# Configure the access_token
dbms.security.oidc.azure.config=principal=unique_name;code_challenge_method=S256;token_type_principal=access_token;token_type_authentication=access_token
# Configure the OIDC token endpoint with the Directory (tenant) ID
dbms.security.oidc.azure.token_endpoint=https://login.microsoftonline.com/54e85725-ed2a-49a4-a19e-11c8d29f9a0f/oauth2/v2.0/token
# Configure the iss claim in the id token with the Directory (tenant) ID
# Make sure you add the trailing slash (`/`) at the end of the URL or this operation might fail.
dbms.security.oidc.azure.issuer=https://sts.windows.net/54e85725-ed2a-49a4-a19e-11c8d29f9a0f/
# Provide the Entra ID parameters, such as client_id, response_type, scope, etc.
dbms.security.oidc.azure.params=client_id=4376dc8b-b5af-424f-9ada-c1c1b2d416b9;response_type=code;scope=openid profile email api://4376dc8b-b5af-424f-9ada-c1c1b2d416b9/access-token
```

ID token

This example shows how to configure Entra ID for authentication and authorization using ID tokens.

Register the application

1. Log in to the Azure portal.
2. Navigate to **Microsoft Entra ID > Overview**.
3. From the **Add** dropdown menu, select **App registration** and fill in the following information to create your SSO application:

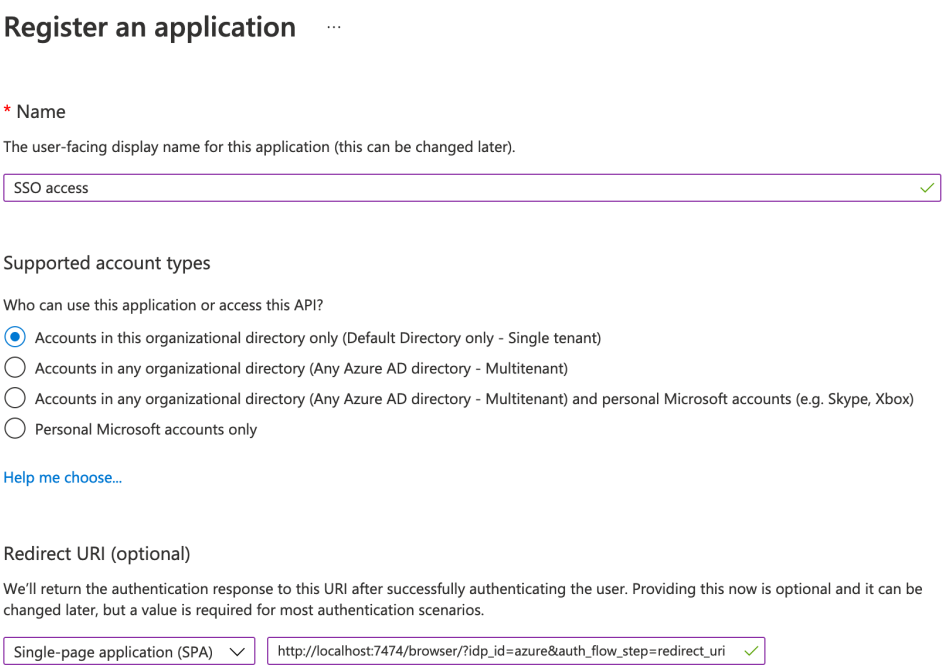


Figure 43. Entra OIDC client creation

The redirect URI *http://localhost:7474/browser/?idp_id=azure&auth_flow_step=redirect_uri* is the URI that will accept returned token responses after successful authentication.

4. Click **Register**.

Configure Neo4j

1. From the **App registrations** page, select the app you created in Register the application.
2. On the application **Overview** page, copy the Application (client) ID value and use it to configure the following properties in the *neo4j.conf* file:

```
dbms.security.oidc.azure.audience=4376dc8b-b5af-424f-9ada-c1c1b2d416b9
dbms.security.oidc.azure.params=client_id=4376dc8b-b5af-424f-9ada-
c1c1b2d416b9;response_type=code;scope=openid profile email
```

3. On the app's **Overview** page, click the **Endpoints** tab, and copy the **OpenID Connect metadata document** URI: Use it to configure the `well_known_discovery_uri` in the *neo4j.conf* file.

```
dbms.security.oidc.azure.well_known_discovery_uri=https://login.microsoftonline.com/54e85725-ed2a-
49a4-a19e-11c8d29f9a0f/v2.0/.well-known/openid-configuration
```

4. Configure Neo4j to use Entra ID authentication in the *neo4j.conf* file:

```
dbms.security.authentication_providers=oidc-azure
dbms.security.authorization_providers=oidc-azure
dbms.security.oidc.azure.display_name=Azure
dbms.security.oidc.azure.auth_flow=pkce
dbms.security.oidc.azure.config=token_type_principal=id_token;token_type_authentication=id_token
```

5. Configure which JWT claim should be used for usernames. Possible values are `sub`, `email`, or `preferred_username`.

> **Important:**
>
> `sub` is the only claim guaranteed to be unique and stable. For details, see #using-claims-to-reliably-identify-a-user-subject-and-object-id[Microsoft documentation] as well as the OpenId spec.

```
dbms.security.oidc.azure.claims.username=sub
```

Map Entra groups to Neo4j roles

Decide whether you want to use Entra groups directly or Entra App Roles.

Using Entra groups directly might be convenient if you already have users assigned to those groups and want to perform Group-to-Role mapping in the *neo4j.conf* file.

Entra App Roles allow a layer of separation between Neo4j roles and groups. When App Roles are used, only the roles relevant to Neo4j are sent in the JWT token. This prevents leaking permissions between applications. JWT tokens also have a limitation of 200 roles per token per user, which can be avoided by sending only the relevant App Roles.

Details about Entra ID App Roles can be found in the Microsoft documentation.

Using Entra groups directly

1. From the **App registrations** page, select your application.
2. From the left-hand side menu, navigate to **Manage → Manifest**.
3. Verify that the server is configured to return the Group Object IDs in the JWT identity tokens:

```
"groupMembershipClaims": "SecurityGroup, ApplicationGroup",
```

4. From the left-hand side menu, navigate to **Manage → Groups**.
5. Create groups and assign users to them. Take note of the **Object Id** column.
6. Configure a mapping from Entra Group Object Ids to Neo4j roles. For details, see Map the identity provider groups to the Neo4j roles.

```
dbms.security.oidc.azure.authorization.group_to_role_mapping= "e8b6ddfa-688d-4ace-987d-6cc5516af188" =
admin; \
                                                              "9e2a31e1-bdd1-47fe-844d-767502bd138d" =
reader
```

7. Configure Neo4j to use the `groups` field from the JWT token.

```
dbms.security.oidc.azure.claims.groups=groups
```

Using Entra ID App roles

1. From the left-hand side menu, navigate to **App roles** and add the Neo4j roles to the Microsoft Entra ID.
   a. Click **Create app role**.
   b. Fill in the fields:
      i. **Display name**: `admin`
      ii. **Allowed member types**: `Users/Groups`
      iii. **Value**: `admin`.
      The **Value** column must either correspond to the Neo4j roles or be mapped in the *neo4j.conf* file.
      iv. **Description**: `Neo4j admin role`
   c. Click **Apply**.
2. Repeat the previous step for the other roles you want to add.
3. Configure a mapping from Entra App Roles to Neo4j roles in the *neo4j.conf* file. For details, see Map the identity provider groups to the Neo4j roles.

```
dbms.security.oidc.azure.authorization.group_to_role_mapping= "managers" = admin; \
                                                              "engineers" = reader
```

4. Configure Neo4j to use the `roles` field from the JWT token.

```
dbms.security.oidc.azure.claims.groups=roles
```

5. Introduced in 5.24 (Optional) If you want control the authentication and authorization on a user level, configure `dbms.security.require_local_user` to `true` in the *neo4j.conf* file. This setting mandates that users with the relevant auth provider attached to them must exist in the database before they can authenticate and authorize with that auth provider. For information on how to create users in this mode, see Creating users.

   For example, to create a user `jake` who can authenticate and authorize using Azure, you can use the following Cypher query:

```
CREATE USER jake
SET HOME DATABASE 'jakesHomeDb'
SET AUTH 'oidc-azure' {SET ID 'jakesUniqueAzureUserId'} // `jakesUniqueAzureUserId` must match the
value of the claim that you configured via dbms.security.oidc.azure.claims.username
```

   See Configure SSO at the user level using auth providers for further examples.

Google

ID token

This example shows how to use Google OpenID Connect for authentication using ID tokens in conjunction with native authorization.

1. Configure the client and the redirect URI:

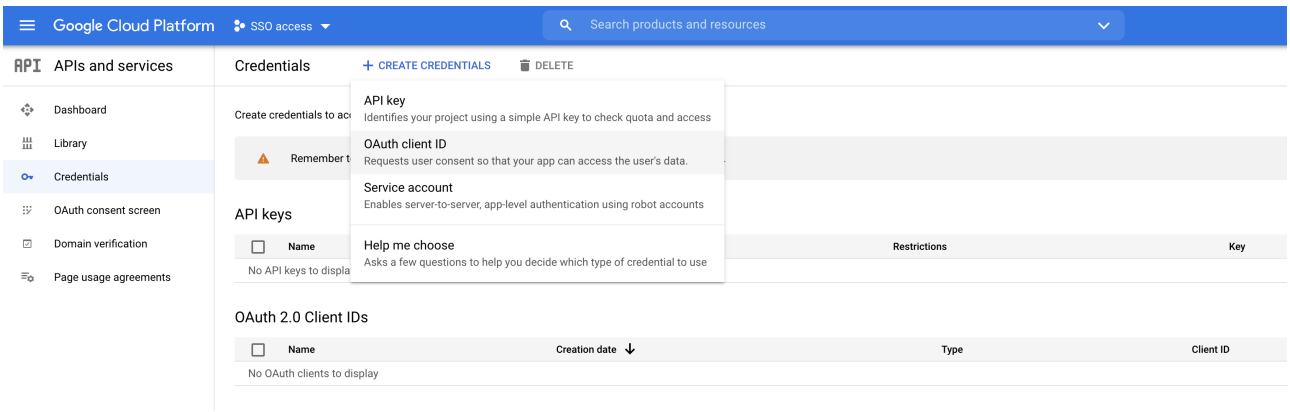


Figure 44. Google OIDC client creation

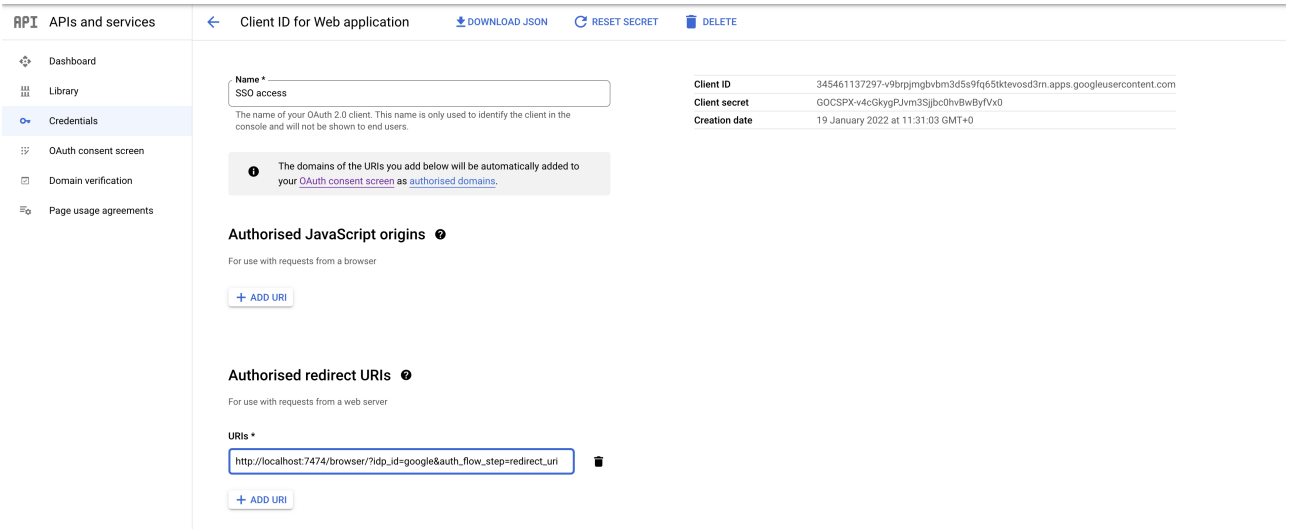


Figure 45. Google OIDC client configuration

> **Important:**
>
> SSO authorization does not work with Google, as the JWT returned by Google does not contain information about the groups that a user belongs to, and cannot be configured to. Therefore, it is recommended to use native (or another flavor) authorization by creating a native version of the user in Neo4j.

2. Configure Neo4j to use Google authentication by setting the following configurations in the *neo4j.conf* file:

```
dbms.security.authentication_providers=oidc-google
dbms.security.authorization_providers=native
dbms.security.oidc.google.display_name=Google
dbms.security.oidc.google.auth_flow=pkce
dbms.security.oidc.google.well_known_discovery_uri=https://accounts.google.com/.well-known/openid-configuration
dbms.security.oidc.google.audience=345461137297-v9brpjmgbvbm3d5s9fq65tktevosd3rn.apps.googleusercontent.com
dbms.security.oidc.google.claims.username=email
dbms.security.oidc.google.params=client_id=345461137297-v9brpjmgbvbm3d5s9fq65tktevosd3rn.apps.googleusercontent.com;response_type=code;scope=openid profile email
dbms.security.oidc.google.token_params=client_secret=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
dbms.security.oidc.google.config=token_type_principal=id_token;token_type_authentication=id_token
```

3. Using one of the following options, create a user in the database who can authenticate and authorize natively to be able to give the users roles from native authorization.

> Introduced in 5.24
>
> This approach relies on the existence of an admin user who can authenticate natively and then create less privileged users via auth providers, who can authenticate only using `oidc-google`, but will receive the roles granted to them using `native` authorization.
>
> > **Note:**

> An admin user with the name `neo4j` is created by default when the database is first started.

1. In the *neo4j.conf* file, temporarily enable native authentication for the `admin` user only and enable the user-level control of authentication and authorization:

   ```
   dbms.security.authentication_providers=oidc-google, native
   dbms.security.require_local_user=true
   ```

   This will switch to *user auth providers* mode whereby users can only authenticate and authorize if they have a corresponding auth provider in the database.

2. Create a user who can authenticate and authorize only using `oidc-google`:

   ```
   CREATE USER jake
   SET HOME DATABASE 'jakesHomeDb'
   SET AUTH 'oidc-google' {SET ID 'jakesUniqueGoogleUserId'}
   ```

   `jakesUniqueGoogleUserId` must match the value of the claim that you configured via `dbms.security.oidc.google.claims.username`.

3. Grant the user `jake` roles, for example, `reader`:

   ```
   GRANT ROLE reader TO jake
   ```

   The user implicitly receives `native` authorization because `native` is in the list of authorization providers and you have explicitly granted the user a role.

4. Once you have set up your users in this way, you can disable native authentication for the database completely. This will prevent all users, including the admin, from logging in with a username and password:

   ```
   dbms.security.authentication_providers=oidc-google
   ```

Alternatively, if you do not use auth providers, you can temporarily enable `native` authentication to create an SSO-authenticated admin user `alice` who can then create other users who can only authenticate using SSO.

1. Temporarily enable `native` authentication:

   ```
   dbms.security.authentication_providers=oidc-google, native
   ```

2. Create an SSO-authenticated `admin` user (in this example an equivalent of `alice@neo4j-test.com` must be set up in the Google SSO provider and their credentials must be known):

   ```
   CREATE USER `alice@neo4j-test.com` SET PASSWORD 'secretpassword';
   ```

```
GRANT ROLE admin to `alice@neo4j-test.com`;
```

3. Disable native authentication for the database to prevent users logging in with username and password:

```
dbms.security.authentication_providers=oidc-google
```

4. Log in via Google SSO as `alice@neo4j-test.com`, the `admin` user.
5. Create other users who can authenticate only using `oidc-google` and will receive the roles granted to them using `native` authorization.

```
CREATE USER jakesUniqueGoogleUserId
SET HOME DATABASE 'jakesHomeDb'
SET PASSWORD 'secretpassword' SET PASSWORD CHANGE NOT
REQUIRED
```

   `jakesUniqueGoogleUserId` must match the value of the claim that you configured via `dbms.security.oidc.google.claims.username`.

6. Grant the user roles using native authorization:

```
GRANT ROLE reader TO jakesUniqueGoogleUserId
```

FAQ

When should `pkce` be used as auth flow?

Assuming the client (Neo4j Browser or Bloom) can be accessed through the public internet, always use `pkce` auth-flow rather than `implicit` because the latter requires the client's secret to be available to the public client. In general, if both flows are available, it is recommended to opt for `pkce` because it is more secure than `implicit`.

Is Google authentication secure if it has a client secret listed in the config?

Yes. Google uses the pkce flow, but identity providers sometimes also use a client secret to ensure the client asking for a token is the one using it (pkce does not guarantee that). The client secret does not add any additional security as it is public but the `pkce` flow provides sufficient security.

Could not parse JWT of type "access_token"

When getting the message `Failed to get credentials: Could not parse JWT of type "access_token"` on Browser, it probably means the provider only accepts ID tokens.

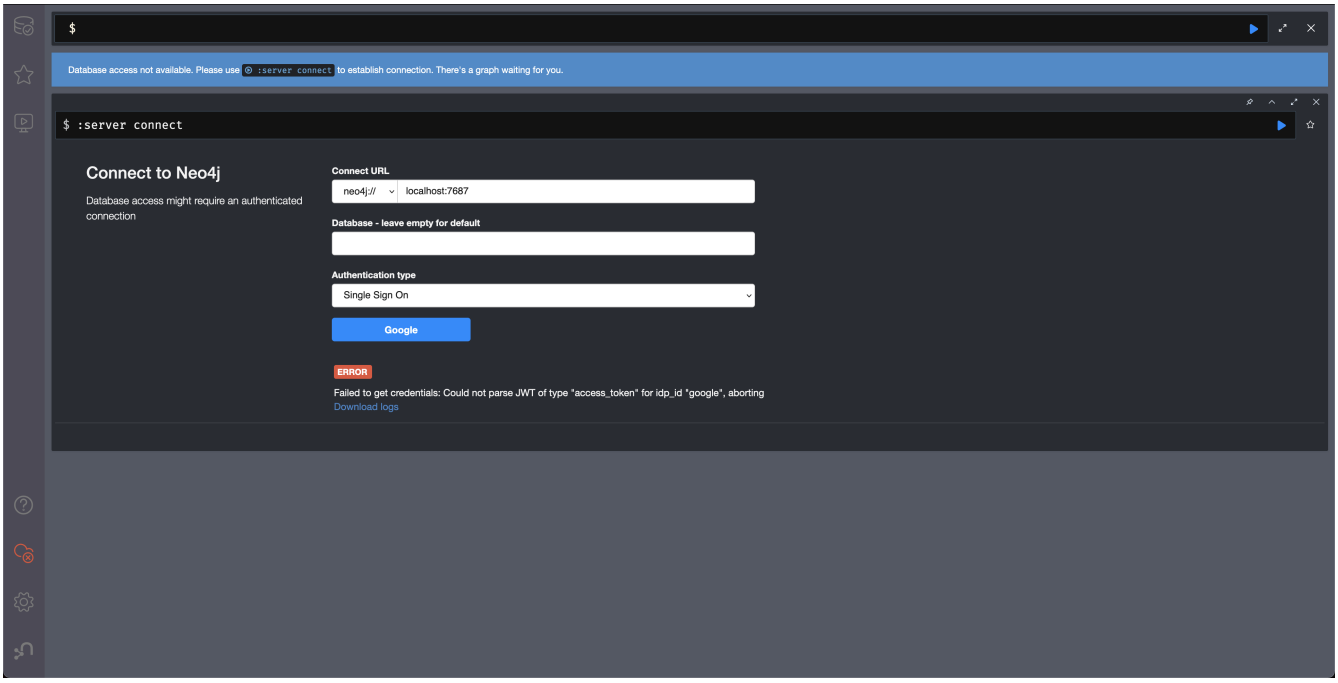


Figure 46. Failed to parse JWT of type access_token

Change to ID tokens in your *neo4j.conf*:

```
dbms.security.oidc.{{provider}}.config=token_type_principal=id_token;token_type_authentication=id_token
```

When should identity tokens vs. access tokens be used?

It is generally safer to use access tokens when possible due to being shorter-lived. If authorization permissions change on the identity provider, Neo4j will fail authorization. Neo4j Browser will try to reconnect and reflect the changed permissions faster than if ID tokens were used.

Debug logging of JWT claims

While setting up an OIDC integration, it is sometimes necessary to perform troubleshooting. In these cases, it can be useful to view the claims contained in the JWT supplied by the identity provider.

To enable the logging of these claims at `DEBUG` level in the security log, set dbms.security.logs.oidc.jwt_claims_at_debug_level_enabled to `true` and the security log level to `DEBUG`. You can do this in *<NEO4J_HOME>/conf/server-logs.xml*.

If you need more information on how to set up and manage the security log, see Configure the security log.

Warning:

Make sure to set dbms.security.logs.oidc.jwt_claims_at_debug_level_enabled back to `false` for production environments to avoid unwanted logging of potentially sensitive information. Also, bear in mind that the set of claims provided by an identity provider in the JWT can change over time.

How to debug further problems with the configuration

Apart from the logs available in *logs/debug.log* and *logs/security.log* in the Neo4j path, you can also use the web-development console in your web browser when doing the SSO authentication flow with Bloom or Neo4j Browser. This could reveal potential problems, such as the one presented below with an example identity provider and the Cross-Origin Request policy:

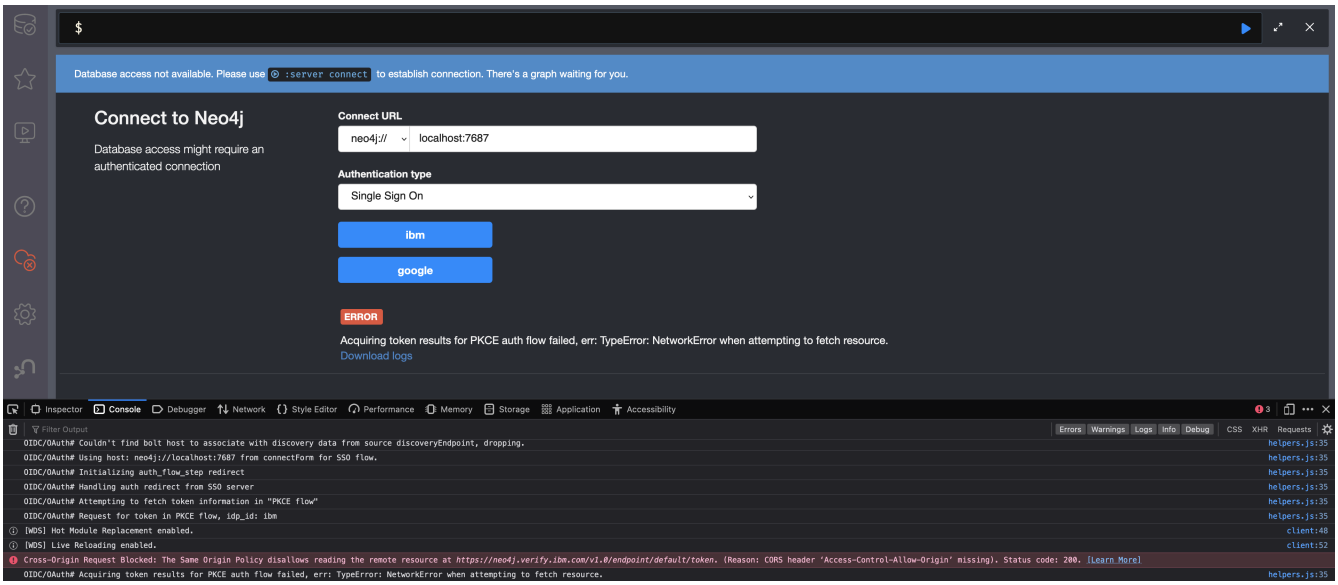


Figure 47. CORS error

The solution involves adding the redirect domain to the list of allowed domains in the provider (in this case, `localhost:8080`):

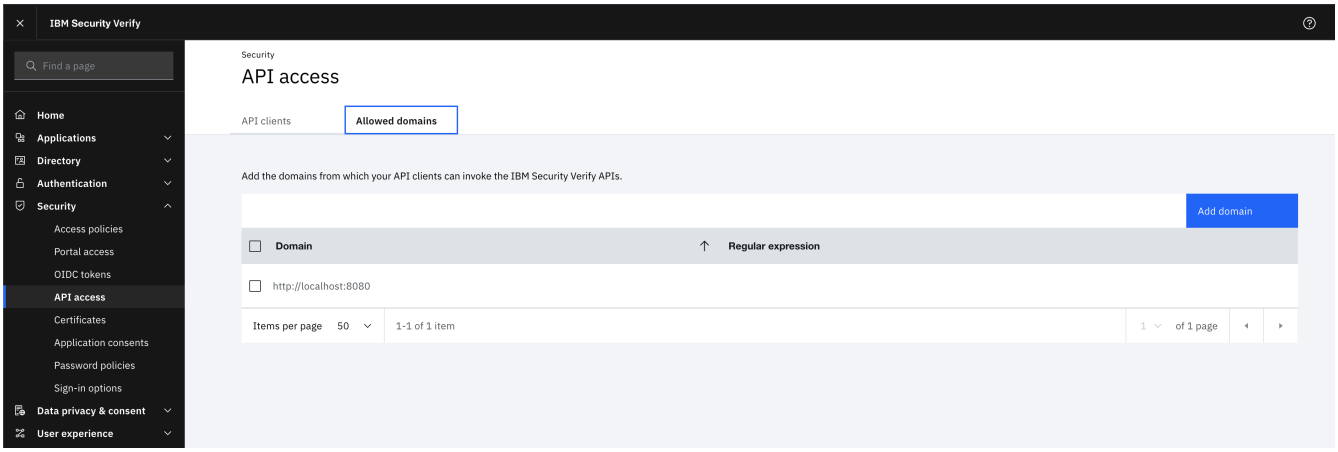


Figure 48. CORS error solution allowing the redirect domain on the provider

Deploying a Neo4j cluster in a Docker container

Neo4j supports clustering in a containerized environment without an orchestration tool. This tutorial walks through setting this up locally for testing purposes. For production deployment across multiple servers, see Deploy a Neo4j cluster on multiple Docker hosts.

Note:

The examples on this page make use of both command expansion and DNS discovery method. For more information, see:

- Command expansion
- Discovery using DNS with multiple records

Deploy a Neo4j cluster using Docker Compose

You can deploy a Neo4j cluster using Docker Compose. Docker Compose is a management tool for Docker containers. You use a YAML file to define the infrastructure of all your cluster servers in one file. Then, by running the single command `docker-compose up`, you create and start all the members without the need to invoke each of them individually. For more information about Docker Compose, see the Docker Compose official documentation.

Prerequisites

- Verify that you have installed Docker Compose. For more information, see the Install Docker Compose official documentation.

Procedure

1. Create a configuration file *neo4j.conf* which will be shared across cluster members and make it readable and writable for the user (eg., `chmod 640 neo4j.conf`)

```
# Setting that specifies how much memory Neo4j is allowed to use for the page cache.
server.memory.pagecache.size=100M

# Setting that specifies the initial JVM heap size.
server.memory.heap.initial_size=100M

# The behavior of the discovery service is determined by the parameters
`dbms.cluster.discovery.resolver_type`, `dbms.cluster.discovery.v2.endpoints`, and
`dbms.cluster.discovery.version`.
# The DNS strategy fetches the IP addresses of the cluster members using the DNS A records.
dbms.cluster.discovery.resolver_type=DNS

# The value of `dbms.cluster.discovery.version` must be set to `V2_ONLY` if you want to use
the discovery service v2.
# The discovery service v2 utilizes the port `6000`.
dbms.cluster.discovery.version=V2_ONLY

# The value of `dbms.cluster.discovery.v2.endpoints` should be set to a single domain name
and the port of the discovery service.
# The domain name returns an A record for every server in the cluster when a DNS lookup is
performed.
# Each A record returned by DNS should contain the IP address of the server in the cluster.
# The configured server uses all the IP addresses from the A records to join or form a
cluster.
# The discovery port must be the same on all servers when using this configuration.
dbms.cluster.discovery.v2.endpoints=neo4j-network:6000

# Address (the public hostname/IP address of the machine)
# and port setting that specifies where this instance advertises for discovery protocol
messages from other members of the cluster.
server.cluster.advertised_address=$(hostname -i)

# Address (the public hostname/IP address of the machine)
# and port setting that specifies where this instance advertises for Raft messages within
the cluster.
server.cluster.raft.advertised_address=$(hostname)

# Enable server-side routing
dbms.routing.enabled=true
```

```
# Use server-side routing for neo4j:// protocol connections.
dbms.routing.default_router=SERVER

# The advertised address for the intra-cluster routing connector.
server.routing.advertised_address=$(hostname)

# Automatically enable servers, rather than needing to explicitly do so for Free servers
initial.dbms.automatically_enable_free_servers=true
```

```
# Setting that specifies how much memory Neo4j is allowed to use for the page cache.
server.memory.pagecache.size=100M

# Setting that specifies the initial JVM heap size.
server.memory.heap.initial_size=100M

# The behavior of the initial discovery is determined by the parameters
`dbms.cluster.discovery.resolver_type` and `dbms.cluster.discovery.endpoints`.
# The DNS strategy fetches the IP addresses of the cluster members using the DNS A records.
dbms.cluster.discovery.resolver_type=DNS

# The value of `dbms.cluster.discovery.endpoints` should be set to a single domain name and
the port of the discovery service.
# The domain name returns an A record for every server in the cluster when a DNS lookup is
performed.
# Each A record returned by DNS should contain the IP address of the server in the cluster.
# The configured server uses all the IP addresses from the A records to join or form a
cluster.
# The discovery port must be the same on all servers when using this configuration.
dbms.cluster.discovery.endpoints=neo4j-network:5000

# Address (the public hostname/IP address of the machine)
# and port setting that specifies where this instance advertises for discovery protocol
messages from other members of the cluster.
server.discovery.advertised_address=$(hostname -i)

# Address (the public hostname/IP address of the machine)
# and port setting that specifies where this instance advertises for Raft messages within
the cluster.
server.cluster.raft.advertised_address=$(hostname)

 # Address (the public hostname/IP address of the machine)
 # and port setting that specifies where this instance advertises for requests for
transactions in the transaction-shipping catch-up protocol.
server.cluster.advertised_address=$(hostname)

# Enable server-side routing
dbms.routing.enabled=true

# Use server-side routing for neo4j:// protocol connections.
dbms.routing.default_router=SERVER

# The advertised address for the intra-cluster routing connector.
server.routing.advertised_address=$(hostname)
```

2. Prepare your *docker-compose.yml* file using the following example. For more information, see the Docker Compose official documentation.

Example 180. Example docker-compose.yml file

```
version: '3.8'

# Custom top-level network
networks:
  neo4j-internal:
```

```
services:

  server1:
    # Docker image to be used
    image: ${NEO4J_DOCKER_IMAGE}

    # Hostname
    hostname: server1

    # Service-level network, which specifies the networks, from the list of the top-level
networks (in this case only neo4j-internal), that the server will connect to.
    # Adds a network alias (used in neo4j.conf when configuring the discovery members)
    networks:
      neo4j-internal:
        aliases:
          - neo4j-network

    # The ports that will be accessible from outside the container - HTTP (7474) and Bolt (7687).
    ports:
      - "7474:7474"
      - "7687:7687"

    # Uncomment the volumes to be mounted to make them accessible from outside the container.
    volumes:
      - ./neo4j.conf:/conf/neo4j.conf # This is the main configuration file.
      - ./data/server1:/data
      - ./logs/server1:/logs
      - ./conf/server1:/conf
      - ./import/server1:/import
      #- ./metrics/server1:/metrics
      #- ./licenses/server1:/licenses
      #- ./ssl/server1:/ssl

    # Passes the following environment variables to the container
    environment:
      - NEO4J_ACCEPT_LICENSE_AGREEMENT
      - NEO4J_AUTH
      - EXTENDED_CONF
      - NEO4J_EDITION
      - NEO4J_initial_server_mode__constraint=PRIMARY

    # Simple check testing whether the port 7474 is opened.
    # If so, the instance running inside the container is considered as "healthy".
    # This status can be checked using the "docker ps" command.
    healthcheck:
      test: ["CMD-SHELL", "wget --no-verbose --tries=1 --spider localhost:7474 || exit 1"]

    # Set up the user
    user: ${USER_ID}:${GROUP_ID}

  server2:
    image: ${NEO4J_DOCKER_IMAGE}
    hostname: server2
    networks:
      neo4j-internal:
        aliases:
          - neo4j-network
    ports:
      - "7475:7474"
      - "7688:7687"
    volumes:
      - ./neo4j.conf:/conf/neo4j.conf
      - ./data/server2:/data
      - ./logs/server2:/logs
      - ./conf/server2:/conf
      - ./import/server2:/import
      #- ./metrics/server2:/metrics
      #- ./licenses/server2:/licenses
      #- ./ssl/server2:/ssl
    environment:
      - NEO4J_ACCEPT_LICENSE_AGREEMENT
      - NEO4J_AUTH
      - EXTENDED_CONF
      - NEO4J_EDITION
      - NEO4J_initial_server_mode__constraint=PRIMARY
    healthcheck:
      test: ["CMD-SHELL", "wget --no-verbose --tries=1 --spider localhost:7474 || exit 1"]
```

```
    user: ${USER_ID}:${GROUP_ID}

  server3:
    image: ${NEO4J_DOCKER_IMAGE}
    hostname: server3
    networks:
      neo4j-internal:
        aliases:
          - neo4j-network
    ports:
      - "7476:7474"
      - "7689:7687"
    volumes:
      - ./neo4j.conf:/conf/neo4j.conf
      - ./data/server3:/data
      - ./logs/server3:/logs
      - ./conf/server3:/conf
      - ./import/server3:/import
      #- ./metrics/server3:/metrics
      #- ./licenses/server3:/licenses
      #- ./ssl/server3:/ssl
    environment:
      - NEO4J_ACCEPT_LICENSE_AGREEMENT
      - NEO4J_AUTH
      - EXTENDED_CONF
      - NEO4J_EDITION
      - NEO4J_initial_server_mode__constraint=PRIMARY
    healthcheck:
      test: ["CMD-SHELL", "wget --no-verbose --tries=1 --spider localhost:7474 || exit 1"]
    user: ${USER_ID}:${GROUP_ID}

  server4:
    image: ${NEO4J_DOCKER_IMAGE}
    hostname: server4
    networks:
      neo4j-internal:
        aliases:
          - neo4j-network
    ports:
      - "7477:7474"
      - "7690:7687"
    volumes:
      - ./neo4j.conf:/conf/neo4j.conf
      - ./data/server4:/data
      - ./logs/server4:/logs
      - ./conf/server4:/conf
      - ./import/server4:/import
      #- ./metrics/server4:/metrics
      #- ./licenses/server4:/licenses
      #- ./ssl/server4:/ssl
    environment:
      - NEO4J_ACCEPT_LICENSE_AGREEMENT
      - NEO4J_AUTH
      - EXTENDED_CONF
      - NEO4J_EDITION
      - NEO4J_initial_server_mode__constraint=SECONDARY
    healthcheck:
      test: ["CMD-SHELL", "wget --no-verbose --tries=1 --spider localhost:7474 || exit 1"]
    user: ${USER_ID}:${GROUP_ID}
```

3. Set up the environment variables:
   - `export USER_ID="$(id -u)"`
   - `export GROUP_ID="$(id -g)"`
   - `export NEO4J_DOCKER_IMAGE=neo4j:enterprise`
   - `export NEO4J_EDITION=docker_compose`
   - `export EXTENDED_CONF=yes`
   - `export NEO4J_ACCEPT_LICENSE_AGREEMENT=yes`

- `export NEO4J_AUTH=neo4j/your_password`

4. Pre-build the folder structure by running the following command:

```
mkdir -p conf/{server1,server2,server3,server4} data/{server1,server2,server3,server4} import/
{server1,server2,server3,server4} logs/{server1,server2,server3,server4}
```

5. Deploy your Neo4j cluster by running `docker-compose up` from your project folder.
6. The instance will be available at the following addresses:
   - Neo4j instance *server1* will be available at *http://localhost:7474/*.
   - Neo4j instance *server2* will be available at *http://localhost:7475/*.
   - Neo4j instance *server3* will be available at *http://localhost:7476/*.
   - Neo4j instance *server4* will be available at *http://localhost:7477/*.
7. Authenticate with the default `neo4j/your_password` credentials.
8. Check the status of the cluster by running the following in Neo4j Browser:

```
SHOW SERVERS
```

Example output:

```
╒══════════════════════════════════════╤═══════════════════╤═══════════╤═══════════╤═══════════════════╕
│"name"                                │"address"          │"state"    │"health"   │"hosting"          │
╞══════════════════════════════════════╪═══════════════════╪═══════════╪═══════════╪═══════════════════╡
│"26c4ec22-958c-468d-9735-d74c43165cf4"│"localhost:7687"   │"Enabled"  │"Available"│["system","neo4j"] │
├──────────────────────────────────────┼───────────────────┼───────────┼───────────┼───────────────────┤
│"4cd6a447-8a16-4ec6-9714-a5b8970d8a80"│"localhost:7687"   │"Enabled"  │"Available"│["system"]         │
├──────────────────────────────────────┼───────────────────┼───────────┼───────────┼───────────────────┤
│"5d1971e9-a3a9-4945-93f4-34457d918708"│"localhost:7687"   │"Enabled"  │"Available"│["system"]         │
├──────────────────────────────────────┼───────────────────┼───────────┼───────────┼───────────────────┤
│"8d4c4941-241e-4fac-97b5-c9a9adbc50c3"│"localhost:7687"   │"Enabled"  │"Available"│["system"]         │
└──────────────────────────────────────┴───────────────────┴───────────┴───────────┴───────────────────┘
```

Deploy a Neo4j Cluster using environment variables

You can set up containers in a cluster to talk to each other using environment variables. Each container must have a network route to each of the others, the `NEO4J_initial_dbms_default__primaries__count`, `NEO4J_initial_dbms_default__secondaries__count`, and `NEO4J_dbms_cluster_discovery_endpoints` environment variables must be set for all servers.

Cluster environment variables

The following environment variables are specific to the Neo4j cluster, and are available in the Neo4j Enterprise Edition:

- `NEO4J_initial_server_mode__constraint`: the database mode, defaults to `NONE`, can be set to `PRIMARY` or `SECONDARY`.
- `NEO4J_dbms_cluster_discovery_endpoints`: a comma-separated list of endpoints, which a server should contact to discover other cluster servers. Deprecated in 5.23

- `NEO4J_dbms_cluster_discovery_v2_endpoints`: a comma-separated list of endpoints, which a server should contact to discover other cluster servers. Introduced in 5.23
- `NEO4J_server_discovery_advertised__address`: hostname/IP address and port to advertise for member discovery management communication. Deprecated in 5.23
- `NEO4J_server_cluster_advertised__address`: hostname/IP address and port to advertise for transaction handling and v2 discovery.
- `NEO4J_server_cluster_raft_advertised__address`: hostname/IP address and port to advertise for cluster communication.
- `NEO4J_dbms_cluster_discovery_version`: the discovery service version to use, defaults to `V1_ONLY`, can be set to `V1_OVER_V2`, `V2_OVER_V1`, or `V2_ONLY`. Introduced in 5.23

See Settings reference for more details of Neo4j cluster settings.

Set up a Neo4j Cluster on a single Docker host

Within a single Docker host, you can use the default ports for HTTP, HTTPS, and Bolt. For each container, these ports are mapped to a different set of ports on the Docker host.

Example of a `docker run` command for deploying a cluster with three servers:

```
docker network create --driver=bridge neo4j-cluster

docker run --name=server1 --detach --network=neo4j-cluster \
    --publish=7474:7474 --publish=7473:7473 --publish=7687:7687 \
    --hostname=server1 \
    --env NEO4J_initial_server_mode__constraint=PRIMARY \
    --env NEO4J_dbms_cluster_discovery_version=V2_ONLY \
    --env NEO4J_dbms_cluster_discovery_v2_endpoints=server1:6000,server2:6000,server3:6000 \
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
    --env NEO4J_server_bolt_advertised__address=localhost:7687 \
    --env NEO4J_server_http_advertised__address=localhost:7474 \
    --env NEO4J_AUTH=neo4j/your_password \
    neo4j:5.26.25-enterprise

docker run --name=server2 --detach --network=neo4j-cluster \
    --publish=8474:7474 --publish=8473:7473 --publish=8687:7687 \
    --hostname=server2 \
    --env NEO4J_initial_server_mode__constraint=PRIMARY \
    --env NEO4J_dbms_cluster_discovery_version=V2_ONLY \
    --env NEO4J_dbms_cluster_discovery_v2_endpoints=server1:6000,server2:6000,server3:6000 \
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
    --env NEO4J_server_bolt_advertised__address=localhost:8687 \
    --env NEO4J_server_http_advertised__address=localhost:8474 \
    --env NEO4J_AUTH=neo4j/your_password \
    neo4j:5.26.25-enterprise

docker run --name=server3 --detach --network=neo4j-cluster \
    --publish=9474:7474 --publish=9473:7473 --publish=9687:7687 \
    --hostname=server3 \
    --env NEO4J_initial_server_mode__constraint=PRIMARY \
    --env NEO4J_dbms_cluster_discovery_version=V2_ONLY \
    --env NEO4J_dbms_cluster_discovery_v2_endpoints=server1:6000,server2:6000,server3:6000 \
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
    --env NEO4J_server_bolt_advertised__address=localhost:9687 \
    --env NEO4J_server_http_advertised__address=localhost:9474 \
    --env NEO4J_AUTH=neo4j/your_password \
    neo4j:5.26.25-enterprise
```

```
docker network create --driver=bridge neo4j-cluster

docker run --name=server1 --detach --network=neo4j-cluster \
    --publish=7474:7474 --publish=7473:7473 --publish=7687:7687 \
    --hostname=server1 \
    --env NEO4J_initial_server_mode__constraint=PRIMARY \
    --env NEO4J_dbms_cluster_discovery_endpoints=server1:5000,server2:5000,server3:5000 \
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
    --env NEO4J_server_bolt_advertised__address=localhost:7687 \
    --env NEO4J_server_http_advertised__address=localhost:7474 \
    --env NEO4J_AUTH=neo4j/your_password \
    neo4j:5.26.25-enterprise

docker run --name=server2 --detach --network=neo4j-cluster \
    --publish=8474:7474 --publish=8473:7473 --publish=8687:7687 \
    --hostname=server2 \
    --env NEO4J_initial_server_mode__constraint=PRIMARY \
    --env NEO4J_dbms_cluster_discovery_endpoints=server1:5000,server2:5000,server3:5000 \
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
    --env NEO4J_server_bolt_advertised__address=localhost:8687 \
    --env NEO4J_server_http_advertised__address=localhost:8474 \
    --env NEO4J_AUTH=neo4j/your_password \
    neo4j:5.26.25-enterprise

docker run --name=server3 --detach --network=neo4j-cluster \
    --publish=9474:7474 --publish=9473:7473 --publish=9687:7687 \
    --hostname=server3 \
    --env NEO4J_initial_server_mode__constraint=PRIMARY \
    --env NEO4J_dbms_cluster_discovery_endpoints=server1:5000,server2:5000,server3:5000 \
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
    --env NEO4J_server_bolt_advertised__address=localhost:9687 \
    --env NEO4J_server_http_advertised__address=localhost:9474 \
    --env NEO4J_AUTH=neo4j/your_password \
    neo4j:5.26.25-enterprise
```

Additional servers can be added to the cluster in an ad-hoc fashion.

Example of a `docker run` command for adding a fourth server with a role `SECONDARY` to the cluster:

```
docker run --name=read-server4 --detach --network=neo4j-cluster \
    --publish=10474:7474 --publish=10473:7473 --publish=10687:7687 \
    --hostname=read-server4 \
    --env NEO4J_initial_server_mode__constraint=SECONDARY \
    --env NEO4J_dbms_cluster_discovery_version=V2_ONLY \
    --env NEO4J_dbms_cluster_discovery_v2_endpoints=server1:6000,server2:6000,server3:6000 \
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
    --env NEO4J_server_bolt_advertised__address=localhost:10687 \
    --env NEO4J_server_http_advertised__address=localhost:10474 \
    neo4j:5.26.25-enterprise
```

```
docker run --name=read-server4 --detach --network=neo4j-cluster \
    --publish=10474:7474 --publish=10473:7473 --publish=10687:7687 \
    --hostname=read-server4 \
    --env NEO4J_initial_server_mode__constraint=SECONDARY \
    --env NEO4J_dbms_cluster_discovery_endpoints=server1:5000,server2:5000,server3:5000 \
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
    --env NEO4J_server_bolt_advertised__address=localhost:10687 \
    --env NEO4J_server_http_advertised__address=localhost:10474 \
    neo4j:5.26.25-enterprise
```

License