

# The Neo4j Operations Manual
## v4.4

# Table of Contents


1. Introduction 2
   1.1. Neo4j editions 2
      1.1.1. Performance and scalability 3
   1.2. Versioning 4
2. Installation 5
   2.1. System requirements 5
      2.1.1. Supported platforms 5
      2.1.2. Hardware requirements 5
      2.1.3. Software requirements 6
      2.1.4. Filesystem 7
      2.1.5. Java 8
   2.2. Linux installation 8
      2.2.1. Debian-based distributions (.deb) 8
      2.2.2. Red Hat, CentOS, Fedora, and Amazon Linux (.rpm) 13
      2.2.3. Linux executable (.tar) 17
3. Uninstall Neo4j 21
   3.1. Neo4j system service 21
   3.2. macOS installation 22
      3.2.1. Unix console application 22
      3.2.2. macOS service 23
      3.2.3. macOS file descriptor limits 23
      3.2.4. Uninstall Neo4j 24
   3.3. Windows installation 24
      3.3.1. Windows console application 24
      3.3.2. Windows service 25
      3.3.3. Windows PowerShell module 26
4. Uninstall Neo4j 29
   4.1. Neo4j Desktop 29
5. Docker 30
   5.1. Get started with Neo4j in Docker 30
      5.1.1. Neo4j editions 30
      5.1.2. Using the Neo4j Docker image 31
      5.1.3. Using `NEO4J_AUTH` to set an initial password 31
      5.1.4. Persisting data using Volumes 31
      5.1.5. Running Neo4j as a non-root user 32
      5.1.6. More useful Docker Run options 32
      5.1.7. Offline installation of Neo4j Docker image 33
   5.2. Persist data with Docker volumes 33

5.2.1. Neo4j mount points and permissions 34
5.2.2. File permissions 35
5.3. Modify the default configuration 36
5.3.1. Environment variables 36
5.3.2. Mounting the */conf* volume 36
5.3.3. Customize a Neo4j Docker image 37
5.4. Plugins 38
5.4.1. Installing plugins 38
5.4.2. `NEO4J_PLUGINS` utility 38
5.4.3. Storing downloaded plugins 39
5.4.4. Installing plugin licenses 40
5.5. Deploy a Neo4j standalone server using Docker Compose 41
5.5.1. Deploy a Neo4j server using basic authentication mechanism 41
5.5.2. Deploy a Neo4j server with Docker secrets 41
5.6. Deploy a Neo4j cluster on Docker 42
5.6.1. Deploy a Causal Cluster with Docker Compose 43
5.6.2. Deploy a Causal Cluster using environment variables 46
5.7. Docker specific operations 48
5.7.1. Use Neo4j Admin 48
5.7.2. Use Neo4j Import 48
5.7.3. Use Neo4j Admin for memory recommendations 49
5.7.4. Use Cypher Shell 49
5.7.5. Install user-defined procedures 52
5.7.6. Configure Neo4j Labs plugins 52
5.8. Security 53
5.8.1. SSL Encryption 53
5.9. Docker maintenance operations 55
5.9.1. Dump and load a Neo4j database (offline) 56
5.9.2. Back up and restore a Neo4j database (online) 56
5.9.3. Upgrade Neo4j on Docker 60
5.9.4. Monitor Neo4j 60
5.10. Docker specific configuration settings 61
6. Kubernetes 73
6.1. Introduction 73
6.1.1. The Neo4j Helm chart repository 74
6.1.2. Using the Neo4j Helm chart repository 74
6.2. Configure the Neo4j Helm chart repository 74
6.2.1. Prerequisites 74
6.2.2. Configure the Neo4j Helm chart repository 74
6.2.3. Check for the available Neo4j Helm charts 75
6.3. Quickstart: Deploy a standalone instance 75

6.3.1. Neo4j Helm charts for standalone server deployments 75
6.3.2. Prerequisites 76
6.3.3. Create a Helm deployment *values.yaml* file 79
6.3.4. Install a Neo4j standalone instance 81
6.3.5. Verify the installation 82
6.3.6. Uninstall Neo4j and clean up the created resources 85
6.4. Quickstart: Deploy a cluster 86
6.4.1. Neo4j Helm charts for cluster deployments 87
6.4.2. Prerequisites 87
6.4.3. Create Helm deployment values files 90
6.4.4. Install Neo4j cluster cores 92
6.4.5. Verify the Neo4j cluster formation 94
6.4.6. Install the Neo4j read replicas 95
6.4.7. Verify the read replica has joined the cluster 96
6.4.8. Access the Neo4j cluster from inside Kubernetes 97
6.4.9. Access the Neo4j cluster from outside Kubernetes 101
6.4.10. Uninstall Neo4j cluster and clean up resources 103
6.5. Configure a Neo4j Helm deployment 104
6.5.1. Create a custom *values.yaml* file 104
6.5.2. Set Neo4j configuration 112
6.5.3. Set an initial password 113
6.5.4. Configure SSL 113
6.5.5. Configure SSO 121
6.5.6. Configure LDAP password through secret 122
6.5.7. Configure resource allocation 122
6.5.8. Configure a service account 124
6.5.9. Configure a custom container image 124
6.5.10. Configure and install APOC core only 124
6.5.11. Configure credentials for plugin's aliases using APOC-extended 125
6.5.12. Install Plugins 126
6.6. Volume mounts and persistent volumes with the Neo4j Helm charts 130
6.6.1. Volume mounts 131
6.6.2. Persistent volumes 131
6.6.3. Mapping volume mounts to persistent volumes 132
6.6.4. Provision persistent volumes with Neo4j Helm chart 135
6.7. Access a Neo4j standalone server 143
6.7.1. Supported Kubernetes services 143
6.7.2. Applications accessing Neo4j from inside Kubernetes 144
6.7.3. Applications accessing Neo4j from outside Kubernetes 145
6.7.4. Customizing Kubernetes Resources 146
6.7.5. Accessing Neo4j for DBMS administration and monitoring 146

6.8. Access a Neo4j cluster 147
6.8.1. Supported Kubernetes services 147
6.8.2. Applications accessing Neo4j from inside Kubernetes 148
6.8.3. Applications accessing Neo4j from outside Kubernetes 149
6.8.4. Customizing Kubernetes Resources 149
6.8.5. Accessing Neo4j for DBMS administration and monitoring 150
6.9. Accessing Neo4j using Kubernetes Ingress 151
6.9.1. Configuration options 151
6.9.2. Configure the Kubernetes Ingress 152
6.9.3. Install the Reverse proxy Helm chart 153
6.9.4. Access your data via Neo4j Browser 153
6.9.5. Access your data via Cypher Shell 154
6.10. Import Data 155
6.10.1. Importing data into Neo4j on Kubernetes 155
6.10.2. Configure the import volume mount 155
6.10.3. Copy files to the *import* volume using `kubectl cp` 156
6.10.4. Use `neo4j-admin import` 156
6.10.5. Alternative approach 156
6.11. Monitoring 157
6.11.1. Logging 157
6.11.2. Log collection 157
6.11.3. Metrics 157
6.12. Operations 158
6.12.1. Maintenance modes 158
6.12.2. Reset the `neo4j` user password 161
6.12.3. Dump and load databases (offline) 161
6.12.4. Back up and restore (online) 162
6.12.5. Upgrade Neo4j Community to Enterprise edition 172
6.12.6. Migrate Neo4j from Labs Helm to Neo4j Helm charts 172
6.12.7. Scale a Neo4j deployment 173
6.12.8. Use custom images from private registries 175
6.12.9. Assign Neo4j pods to specific nodes 176
6.13. Troubleshooting 177
6.13.1. Locate and investigate problems with the Neo4j Helm chart 177
6.13.2. Neo4j crashes or restarts unexpectedly 178
7. Configuration 182
7.1. The neo4j.conf file 182
7.1.1. Introduction 182
7.1.2. Syntax 182
7.1.3. JVM-specific configuration settings 183
7.1.4. List currently active settings 183

7.2. Command expansion 184
7.2.1. How it works 184
7.2.2. Enabling 185
7.2.3. Logging 185
7.2.4. Error Handling 185
7.3. File locations 186
7.3.1. Default file locations 186
7.3.2. Customize your file locations 188
7.3.3. File permissions 189
7.4. Ports 190
7.4.1. Backup 191
7.4.2. HTTP 192
7.4.3. HTTPS 192
7.4.4. Bolt 193
7.4.5. Causal Cluster 193
7.4.6. Graphite monitoring 194
7.4.7. Prometheus monitoring 194
7.4.8. JMX monitoring 195
7.4.9. Remote debugging 195
7.5. Configure connectors 195
7.5.1. Available connectors 195
7.5.2. Configuration options 196
7.5.3. Options for Bolt thread pooling 197
7.5.4. Defaults for addresses 197
7.6. Set an initial password 199
7.7. Configure plugins 199
7.7.1. Install and configure plugins 200
7.8. Update dynamic settings 201
7.8.1. Introduction 201
7.8.2. Discover dynamic settings 201
7.8.3. Update dynamic settings 202
7.9. Transaction log 203
7.9.1. Transaction logging 204
7.9.2. Log location 204
7.9.3. Log rotation 204
7.9.4. Log retention 204
7.9.5. Log pruning 206
7.10. Configuration settings 207
7.10.1. Configuration setting group for the load balancing framework 207
7.10.2. Checkpoint settings 208
7.10.3. Cluster settings 209

7.10.4. Fabric settings 226
7.10.5. Connection settings 231
7.10.6. Cypher settings 239
7.10.7. Database settings 241
7.10.8. DBMS settings 243
7.10.9. Import settings 251
7.10.10. Index settings 252
7.10.11. Logging settings 254
7.10.12. Memory settings 264
7.10.13. Metrics settings 269
7.10.14. Neo4j Browser and client settings 278
7.10.15. Security settings 279
7.10.16. Transaction settings 295
7.10.17. Transaction log settings 297
8. Manage databases 299
8.1. Introduction 299
8.1.1. Concepts 299
8.1.2. The `system` database 300
8.1.3. The default and home database 301
8.1.4. Per-user home databases 302
8.2. Administration and configuration 302
8.2.1. Administrative commands 302
8.2.2. Configuration parameters 304
8.3. Queries 306
8.3.1. Show the status of a specific database 306
8.3.2. Show the status of all databases 307
8.3.3. Show the status of the default database 307
8.3.4. Create a database 308
8.3.5. Switch a database 309
8.3.6. Create or replace a database 310
8.3.7. Stop a database 311
8.3.8. Start a database 312
8.3.9. Drop or remove a database 313
8.4. Error handling 313
8.4.1. Observing errors 314
8.4.2. Database states 314
8.4.3. Retrying failed operations 315
8.4.4. Quarantined databases 316
8.5. Databases in a Causal Cluster 319
8.5.1. Change the default database 319
8.5.2. Run Cypher administrative commands from Cypher Shell on a Causal Cluster 319

8.6. Connecting remote databases 321
8.6.1. Setup example 322
8.6.2. Fabric vs remote alias database 322
8.6.3. Configuration of a remote DBMS (*Bob*) 323
8.6.4. Configuration of a DBMS with a remote database alias (*Alice*) 324
8.6.5. Managing remote database aliases 325
8.6.6. Changing the encryption key 325
8.6.7. User connection to remote database aliases 326
9. Clustering 327
9.1. Introduction 327
9.1.1. Overview 327
9.1.2. Operational view 328
9.1.3. Primary servers 328
9.1.4. Secondary servers 329
9.1.5. Causal consistency 330
9.2. Deploy a cluster 331
9.2.1. Introduction 331
9.2.2. Configure a cluster with Single and Read Replica instances 332
9.2.3. Configure a cluster with Core instances 334
9.2.4. Add a Core Server to an existing cluster 336
9.2.5. Add a Secondary server to an existing cluster 337
9.2.6. Detach a Secondary server from an existing cluster 338
9.3. Seed a cluster 339
9.3.1. Introduction 339
9.3.2. Seed a cluster from a database dump (offline) 340
9.3.3. Seed a cluster from a database backup (online) 341
9.3.4. Seed a cluster using the import tool 344
9.4. Discovery 344
9.4.1. Overview 344
9.5. Intra-cluster encryption 346
9.5.1. Introduction 346
9.5.2. Example deployment 347
9.6. Internals of clustering 350
9.6.1. Elections and leadership 350
9.6.2. Leadership balancing 350
9.6.3. Multi-database and the reconciler 351
9.6.4. Server-side routing 351
9.6.5. Store copy 353
9.6.6. On-disk state 355
9.7. Settings reference 355
9.7.1. Common server settings 355

9.7.2. Multi-data center settings 359
9.8. Clustering glossary 361
10. Fabric 365
10.1. Introduction 365
10.1.1. Overview 365
10.1.2. Fabric concepts 366
10.1.3. Deployment examples 366
10.2. Configuration 369
10.2.1. Fabric database setup 369
10.2.2. Authentication and authorization 373
10.2.3. Important settings 374
10.3. Queries 376
10.3.1. Query a single graph 376
10.3.2. Query multiple graphs 376
10.3.3. Query all graphs 377
10.3.4. Query result aggregation 377
10.3.5. Correlated subquery 377
10.3.6. Updating query 378
10.3.7. Mapping functions 378
10.3.8. Fabric built-in functions 379
10.4. Further considerations 379
10.5. Sharding data with the `copy` command 380
11. Backup and restore 382
11.1. Backup and restore planning 382
11.1.1. Backup and restore strategy 382
11.1.2. Backup and restore options 383
11.1.3. Databases to backup 385
11.1.4. Additional files to back up 386
11.1.5. Storage considerations 386
11.2. Backup modes 386
11.2.1. Full backup 386
11.2.2. Incremental backup 387
11.3. Back up an online database 387
11.3.1. Command 388
11.3.2. Online backup configurations 392
11.3.3. Examples 395
11.4. Prepare a database for restoring 395
11.4.1. Command 395
11.4.2. Example 396
11.5. Restore a database backup 396
11.5.1. Command 396

11.5.2. Example 398
11.6. Back up an offline database 399
11.6.1. Command 399
11.6.2. Example 399
11.7. Restore a database dump 400
11.7.1. Command 400
11.7.2. Example 401
11.8. Copy a database store 401
11.8.1. Command 402
11.8.2. Examples 405
12. Authentication and authorization 407
12.1. Introduction 407
12.2. Built-in roles 408
12.3. Recover admin user and password 412
12.3.1. Disable authentication 412
12.3.2. Recover a lost password 413
12.3.3. Recover an unassigned admin role 414
12.3.4. Recover the admin role 415
12.3.5. Post-recovery steps 416
12.4. Fine-grained access control 416
12.4.1. Healthcare use case 416
12.4.2. Security 418
12.4.3. Access control using built-in roles 419
12.4.4. Sub-graph access control using privileges 419
12.5. Integration with LDAP directory services 434
12.5.1. Introduction 434
12.5.2. LDAP dynamic configuration settings 435
12.5.3. Set Neo4j to use LDAP 435
12.5.4. Map the LDAP groups to the Neo4j roles 436
12.5.5. Configure Neo4j to use Active Directory 436
12.5.6. Configure Neo4j to use OpenLDAP 439
12.5.7. Verify the LDAP configuration 439
12.5.8. The auth cache 440
12.5.9. Available methods of encryption 441
12.5.10. Use a self-signed certificate (SSL) in a test environment 442
12.6. Integration with Single Sign-On Services 442
12.6.1. OIDC configuration settings 442
12.6.2. Configure Neo4j to use OpenID Connect 444
12.6.3. Map the Identity Provider Groups to the Neo4j Roles 444
12.6.4. Configure Neo4j to use an OpenID Connect Identity Provider 445
12.6.5. Use a self-signed certificate (SSL) in a test environment 447

12.6.6. Debug logging of JWT claims 447
12.7. Manage procedure and user-defined function permissions 447
12.7.1. Manage procedure permissions 448
12.7.2. Manage user-defined function permissions 449
12.7.3. Manage procedure and user-defined function permissions from config setting 450
12.8. Terminology 451
13. Security 452
13.1. Securing extensions 452
13.1.1. Allow listing 452
13.1.2. Unrestricting 453
13.2. SSL framework 453
13.2.1. SSL providers 453
13.2.2. Certificates 455
13.2.3. Connectors 456
13.2.4. Configuration 456
13.2.5. SSL logs 469
13.2.6. Terminology 470
13.2.7. Configure SSL for FIPS 140-2 compatibility 471
13.3. Browser credentials handling 476
13.4. Security checklist 477
14. Performance 479
14.1. Memory configuration 479
14.1.1. Considerations 481
14.1.2. Capacity planning 482
14.1.3. Limit transaction memory usage 483
14.2. Index configuration 484
14.2.1. B-tree indexes 485
14.2.2. Text indexes 488
14.2.3. Full-text indexes 488
14.2.4. Token lookup indexes 489
14.2.5. Future indexes 491
14.3. Tuning of the garbage collector 491
14.4. Bolt thread pool configuration 493
14.4.1. How thread pooling works 493
14.4.2. Configuration options 493
14.4.3. How to size your Bolt thread pool 494
14.5. Linux file system tuning 494
14.6. Disks, RAM and other tips 495
14.6.1. Storage 495
14.6.2. Page cache 495
14.6.3. Active page cache warmup 496

14.6.4. Checkpoint IOPS limit 497
14.7. Statistics and execution plans 497
14.7.1. Configure statistics collection 497
14.7.2. Configure the replanning of execution plans 499
14.8. Space reuse 500
14.8.1. ID files 500
14.8.2. Reclaim unused space 501
15. Monitoring 505
15.1. Logging 506
15.1.1. Log files 506
15.1.2. Log format 506
15.1.3. Log configurations 507
15.2. Metrics 521
15.2.1. Essential metrics 521
15.2.2. Enable metrics logging 524
15.2.3. Expose metrics 525
15.2.4. Metrics reference 526
16. Metrics specific to Causal Clustering 535
17. Java Virtual Machine Metrics 538
17.1. Manage queries 539
17.1.1. List all running queries 539
17.1.2. List all active locks for a query 539
17.1.3. Terminate queries 540
17.2. Manage transactions 541
17.2.1. Configure transaction timeout 541
17.2.2. Configure lock acquisition timeout 542
17.2.3. List all running transactions 542
17.3. Manage connections 542
17.3.1. List all network connections 542
17.3.2. Terminate multiple network connections 544
17.3.3. Terminate a single network connection 545
17.4. Manage background jobs 546
17.4.1. Listing active background jobs 547
17.4.2. Listing failed job executions 548
17.5. Monitor a Causal Cluster 549
17.5.1. Procedures for monitoring a Causal Cluster 549
17.5.2. Monitor cluster endpoints for status information 553
17.6. Monitor individual database states 558
17.6.1. Listing Databases 558
17.6.2. Listing a single database 562
18. Tools 565

18.1. Neo4j CLI tool 565
18.1.1. Syntax and commands 565
18.1.2. Environment variables 566
18.2. Neo4j Admin 566
18.2.1. Introduction 566
18.2.2. Syntax and commands 567
18.2.3. Environment variables 569
18.2.4. Environment variables 569
18.2.5. Exit codes 570
18.2.6. Consistency checker 570
18.2.7. Neo4j Admin report 572
18.2.8. Display store information 574
18.2.9. Memory recommendations 578
18.2.10. Import 579
18.2.11. Unbind a Neo4j cluster server 594
18.2.12. Upload to Neo4j AuraDB 596
18.3. Cypher Shell 599
18.3.1. About Cypher Shell CLI 599
18.3.2. Syntax 599
18.3.3. Running Cypher Shell within the Neo4j distribution 601
18.3.4. Running Cypher Shell from a different server 601
18.3.5. Available commands 602
18.3.6. Running Cypher statements 602
18.3.7. Query parameters 603
18.3.8. Transactions 604
18.3.9. Procedures 605
18.3.10. Supported operating systems 606
18.3.11. Keyboard shortcuts 606
Appendix 607
Appendix A: Reference 607
.A.1. Procedures 607
Appendix B: Tutorials 630
.B.1. Set up a local Causal Cluster 631
.B.2. Back up and restore a database in Causal Cluster 637
.B.3. Neo4j Admin import 643
.B.4. Set up and use Fabric 656
.B.5. Neo4j Single Sign-On (SSO) configuration 669
Appendix C: Advanced Causal Clustering 682
.C.1. Causal Clustering lifecycle 683
.C.2. Multi-data center 689
.C.3. Multi-data center design 690

.C.4. Multi-data center operations 698
.C.5. Multi-data center load balancing 704
.C.6. Data center disaster recovery 709
.1. Embedded usage 711
Appendix D: Deprecated security procedures 712
.D.1. Enterprise Edition 712
.D.2. Community Edition 724
Appendix E: Query routing decisions 727
.E.1. Introduction 727
.E.2. Routing decision tree 727
.E.3. Illustrated routing decision tree 728

© 2026

Documentation license: Creative Commons 4.0

This manual covers the following areas:

- Introduction — Introduction of Neo4j Community and Enterprise Editions.
- Installation — Instructions on how to install Neo4j in different deployment contexts.
- Docker — Instructions on how to use Neo4j on Docker.
- Kubernetes — Instructions on how to use Neo4j on Kuberenetes.
- Configuration — Instructions on how to configure certain parts of Neo4j.
- Manage databases — Instructions on how to manage multiple active databases with Neo4j.
- Clustering — Comprehensive descriptions of Neo4j Causal Clustering.
- Fabric — Instructions on how to configure and use Neo4j Fabric.
- Backup and restore — Instructions on how to back up and restore Neo4j deployments.
- Authentication and authorization — Instructions on user management and role-based access control.
- Security — Instructions on server security.
- Monitoring — Instructions on setting up Neo4j monitoring.
- Performance — Instructions on how to go about performance tuning for Neo4j.
- Tools — Description of Neo4j tools.
- Reference — Listings of all Neo4j configuration parameters.
- Tutorials — Step-by-step instructions on various scenarios for setting up Neo4j.
- Advanced Causal Clustering — Advanced concepts and actions for Neo4j Causal Clustering.
- Deprecated security procedures — Deprecated security procedures.
- Query routing decisions — Query routing decisions.



**Note:**

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

*Who should read this?*

This manual is written for:

- the engineer performing the Neo4j production deployment.
- the operations engineer supporting and maintaining the Neo4j production database.
- the enterprise architect investigating database options.
- the infrastructure architect planning the Neo4j production deployment.

# Chapter 1. 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.

## 1.1. 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 |
|---|---|---|
| Property graph model | ✔ | ✔ |
| Native graph processing & storage | ✔ | ✔ |
| ACID-compliant transactions | ✔ | ✔ |
| Cypher graph query language | ✔ | ✔ |
| Neo4j Browser with syntax highlighting | ✔ | ✔ |
| Bolt Protocol | ✔ | ✔ |
| Language drivers for C#, Go, Java, JavaScript & Python [1] | ✔ | ✔ |
| High-performance native API | ✔ | ✔ |

| Feature | Community Edition | Enterprise Edition |
| --- | --- | --- |
| High-performance caching | ✔ | ✔ |
| Cost-based query optimizer | ✔ | ✔ |
| Graph algorithms to support AI initiatives [1] | ✔ | ✔ |
| Fast writes via native label indexes | ✔ | ✔ |
| Composite indexes | ✔ | ✔ |
| Full-text node & relationship indexes | ✔ | ✔ |
| Store copy | ✘ | ✔ |
| Auto-reuse of space | ✔ | ✔ |
| Multiple databases (beyond the `system` and default databases) | ✘ | ✔ |
| *Slotted* and *Pipelined* Cypher runtimes | ✘ | ✔ |
| Property-existence constraints | ✘ | ✔ |
| Node Key constraints | ✘ | ✔ |
| Listing and terminating running queries | ✘ | ✔ |
| Role-based access control | ✘ | ✔ |
| Sub-graph access control | ✘ | ✔ |
| LDAP and Active Directory integration | ✘ | ✔ |
| Kerberos security option | ✘ | ✔ |

### 1.1.1. Performance and scalability

*Table 2. Performance and scalability features*

| Feature | Community Edition | Enterprise Edition |
| --- | --- | --- |
| Causal Clustering for global-scale applications | ✘ | ✔ |
| Intra-cluster encryption | ✘ | ✔ |
| Offline backups | ✔ | ✔ |
| Online backups | ✘ | ✔ |
| Encrypted backups | ✘ | ✔ |
| Rolling upgrades | ✘ | ✔ |
| Automatic cache warming | ✘ | ✔ |

| Feature | Community Edition | Enterprise Edition |
| --- | --- | --- |
| Routing and load balancing with Neo4j Drivers | ✗ | ✓ |
| Advanced monitoring | ✗ | ✓ |
| Graph size limitations | 34 billion nodes, 34 billion relationships, and 68 billion properties | No limit |
| Import command-line tool (`neo4j-admin import` command) | ✓ | ✓ |
| Import command-line tool (`neo4j-admin import` command), resumable | ✗ | ✓ |

## 1.2. 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 previous `MAJOR` version.
- `MINOR` version - functionality in a backwards 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.

# Chapter 2. Installation

The topics described are:

- 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.

## 2.1. System requirements

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



**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.

### 2.1.1. Supported platforms

Neo4j is supported on systems with x86_64 and ARM64 architectures, whether they are a physical, virtual, or containerized environments.

### 2.1.2. Hardware requirements

In terms of minimum hardware requirements, follow these guidelines:

*Table 3. 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:<br>• Neo4j workloads tend significantly toward random reads.<br>• Select media with low average seek time: SSD over spinning disks.<br>• Consult Disks, RAM and other tips for more details. |

For personal use and software development:

*Table 4. Hardware requirement guidelines for personal use and software development.*

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

For cloud environments:

*Table 5. Hardware requirement guidelines for cloud environments.*

| | |
|---|---|
| CPU | 2vCPU minimum, 16+ recommended, possibly Xeon processors. |
| Memory | 2GB minimum, size depends 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.<br>Storage size depends on the size of the databases. |

For server-based, on-premise environments:

*Table 6. Hardware requirement guidelines for server-based, on-premise environments.*

| | |
|---|---|
| CPU | Intel Xeon processors. |
| Memory | 8GB minimum, size depends on workloads; in some cases, it is recommended to use instances with memory that fits the size of the graph in use. |
| Storage | SATA i7.2K RPM 6Gbps Hard Drive minimum, NVMe SSD recommended.<br>Storage size depends on the size of the databases. |

## 2.1.3. Software requirements

For personal use and software development:

*Table 7. Software requirements for personal use and software development.*

| Operating System | Supported JDK |
| --- | --- |
| MacOS 10.14+ | ZuluJDK 11 |
| Ubuntu Desktop 18.04+ | OpenJDK 11, OracleJDK 11, and ZuluJDK 11 |
| Debian 10, 11, 12 | OpenJDK 11 (except Debian 12), OracleJDK 11, and ZuluJDK 11 |
| SuSE 15+ | Oracle JDK 11 |
| Windows 10 | OracleJDK 11 and ZuluJDK 11 |

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

*Table 8. Software requirements for cloud environments, and server-based, on-premises environments.*

| Operating System | Supported JDK |
| --- | --- |
| Amazon Linux AMI 2018.03+ | Amazon Corretto 11, OpenJDK 11, and OracleJDK 11 |
| CentOS Stream 8, 9 | OpenJDK 11 |
| Debian 11 | OpenJDK 11, OracleJDK 11 |
| Red Hat Enterprise Linux Server 8.6, 8.8, 9.0, 9.2, 9.4 | Red Hat OpenJDK 11, Oracle JDK 11, and ZuluJDK 11 |
| Ubuntu Server 18.04+ | OpenJDK 11, OracleJDK 11, and ZuluJDK 11 |
| Windows Server 2016, 2019, 2022 | OracleJDK 11 and ZuluJDK 11 |

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

### 2.1.4. 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 `dbms.jvm.additional=-Djava.io.tmpdir=/home/neo4j` in *conf/neo4j.conf*. Additionally, 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.

### 2.1.5. 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 9. Neo4j version and JVM requirements.*

| Neo4j Version | JVM compliance |
|---|---|
| 3.x | Java SE 8 Platform Specification |
| 4.x | Java SE 11 Platform Specification |

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.

## 2.2. Linux installation

This section describes how to install Neo4j on Linux using Debian or RPM packages, or from a Tar archive. You can also install Neo4j as a system service using systemd.

- Debian-based distributions (.deb)
- Red Hat, CentOS, Fedora, and Amazon Linux (.rpm)
- Linux executable (.tar)
- Neo4j system service

### 2.2.1. Debian-based distributions (.deb)

#### Installation

To install Neo4j on Debian you need to make sure of the following:

- An OpenJDK Java 11 runtime is installed or available through your package manager.
- The repository containing the Neo4j Debian package is known to the package manager.

##### Java Prerequisites (Oracle Java and Ubuntu 16.04+ only)

Neo4j 4.4 requires the Java 11 runtime. Java 11 is not included in Ubuntu 16.04 LTS and will have to be set up manually prior to installing or upgrading to Neo4j 4.4, as described below. Ubuntu 18.04 onwards already has the Openjdk Java 11 package available through `apt`.

Oracle Java and Debian

Neo4j is compatible with Oracle Java on Debian/Ubuntu Linux, but should be installed via tarball. The Debian installer may still be used, but it will install OpenJDK Java 11 in addition to any existing Java installations.

This is due to changes in Oracle's Debian package manifest between Java versions 8 and 11.

```
echo "deb https://httpredir.debian.org/debian stretch-backports main" | sudo tee -a
/etc/apt/sources.list.d/stretch-backports.list
sudo apt-get update
sudo apt-get install openjdk-11-jre
```

If you already had a different version of Java installed, see Dealing with multiple installed Java versions to make sure Java 11 is the default version. You are now ready to install Neo4j.

#### Java 11 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 11 automatically if it is not already installed. See Dealing with multiple installed Java versions to make sure you can start Neo4j after install.

#### Dealing with multiple installed Java versions

It is important that you configure your default Java version to point to Java 11, or Neo4j 4.4.48 will be unable to start. Do so with the `update-java-alternatives` command.

- First list all your installed version of Java with `update-java-alternatives --list`

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

  ```
  java-1.11.0-openjdk-amd64 1071 /usr/lib/jvm/java-1.11.0-openjdk-amd64
  java-1.8.0-openjdk-amd64 1069 /usr/lib/jvm/java-1.8.0-openjdk-amd64
  ```

- Identify your Java 11 version, in this case it is `java-1.11.0-openjdk-amd64`. Then set it as the default with (replacing `<java11name>` with the appropriate name from above)

  ```
  sudo update-java-alternatives --jre --set <java11name>
  ```

### Add the repository

The Debian package is available from https://debian.neo4j.com.

- To use the repository for generally available versions of Neo4j, run:

  ```
  wget -O - https://debian.neo4j.com/neotechnology.gpg.key | sudo apt-key add -
  echo 'deb https://debian.neo4j.com stable latest' | sudo tee -a /etc/apt/sources.list.d/neo4j.list
  sudo apt-get update
  ```

To avoid the risk of the `apt` package manager accidentally forcing a database upgrade, different major and minor releases of Neo4j are also available separately inside the repository. To install Neo4j this way, specify the major and minor version required, in place of `latest`.

We recommend the following method for production or business critical installations:

```
wget -O - https://debian.neo4j.com/neotechnology.gpg.key | sudo apt-key add -
echo 'deb https://debian.neo4j.com stable 4.4' | sudo tee -a /etc/apt/sources.list.d/neo4j.list
sudo apt-get update
```

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

```
apt list -a neo4j
```



**Note:**

In Ubuntu server installations you will 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 Community Edition:

```
sudo apt-get install neo4j=1:4.4.48
```

To install Neo4j Enterprise Edition:

```
sudo apt-get install neo4j-enterprise=1:4.4.48
```

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



**Note:**

Versions of Neo4j that are not yet generally available may differ in naming.

The naming structure of packages are normally composed as `neo4j-enterprise=1:<version>~<release>`. For example, Neo4j Enterprise Edition Milestone Release 3 would be: `neo4j-enterprise=1:4.0.0~beta03mr03`.

Refer to the download page for more information regarding the name of packages.

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`:

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

#### Offline installation

If you cannot reach `https://debian.neo4j.com`, perhaps due to a firewall, you will need to obtain Neo4j via an alternative machine which 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 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_4.4.48_all.deb
     ```

     **Note:**

     To list all files that the Debian software package ( `.deb` file) installs:

     ```
     dpkg --contents neo4j_4.4.48_all.deb
     ```

   - Neo4j Community Edition:

```
curl -O https://dist.neo4j.org/deb/neo4j_4.4.48_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, Neo4j is enabled to start automatically on system boot by default.

**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
---
```

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

### Install on Red Hat, CentOS, Fedora, or Amazon Linux

> **Note:**
>
> If you follow guidelines such as STIG RHEL 8 must mount /tmp with the noexec option, you may have `/tmp` mounted with the `noexec` option. This could cause a conflict, as the Neo4j Admin tool uses `zstd` to compress files. However, Java supports pointing to an alternative `temp` directory by either setting `dbms.jvm.additional=-Djava.io.tmpdir=/path/that/has/exec/permissions` in the *neo4j.config* file, or in the `JAVA_OPTS` environment variable.

#### Standard installation

1. Add the repository.

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

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

   Create a `neo4j.repo` file by running the following:

   ```
   cat << EOF >  /etc/yum.repos.d/neo4j.repo
   [neo4j]
   name=Neo4j RPM Repository
   baseurl=https://yum.neo4j.com/stable/4.4
   enabled=1
   gpgcheck=1
   EOF
   ```

   > **Note:**
   >
   > If you are upgrading from an older version of Neo4j, you may need to clear the package manager cache before Neo4j packages become available: `yum clean dbcache`

2. Ensure the correct Java version.

   Neo4j 4.4 requires the Java 11 runtime. Most of our supported RPM Linux distributions have Java 11 available by default. There is some minor setup required for Amazon Linux, and for compatibility with Oracle Java 11:

   - Java 11 on Amazon Linux:

     To enable OpenJDK 11 on Amazon Linux run the shell command:

     ```
     amazon-linux-extras enable java-openjdk11
     ```

     You are now ready to install Neo4j 4.4.48, which will install Java 11 automatically if it is not already installed.

   - Oracle Java 11:

     Oracle and OpenJDK provide incompatible RPM packages for Java 11. We provide an adapter for Oracle Java 11 which must be installed before Neo4j. The adapter contains no code, but will stop the package manger from installing OpenJDK 11 as a dependency despite an existing Java 11 installation.

     This step assumes that you have performed the previous step to set up the yum repository.

     a. Download and install the Oracle Java 11 JDK from the Oracle website.

     b. Install the adapter:

        ```
        sudo yum install https://dist.neo4j.org/neo4j-java11-adapter.noarch.rpm
        ```

        The SHA-256 of the adapter package can be verified against https://dist.neo4j.org/neo4j-java11-adapter.noarch.rpm.sha256.

   You are now ready to install Neo4j 4.4.48.

3. Install Neo4j.

   - To install Neo4j Community Edition as `root`:

     ```
     yum install neo4j-4.4.48
     ```

   - To install Neo4j Enterprise Edition as `root`:

     ```
     yum install neo4j-enterprise-4.4.48
     ```

     When installing Neo4j Enterprise Edition, you will be required to accept the license agreement before the interactive installation is allowed to complete.

4. Run the following to return the version and edition of Neo4j that has been installed:

```
rpm -qa | grep neo
```

### Non-interactive installation of Neo4j Enterprise Edition

For a non-interactive installation, you can set the environment variable `NEO4J_ACCEPT_LICENSE_AGREEMENT` to `yes` to indicate that you have read and accepted the license agreement. 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-4.4.48
```

## 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 neo4j-repository
   ```

2. Install Neo4j.
   - To install Neo4j Community Edition as `root`:

     ```
     zypper install neo4j-4.4.48
     ```

   - To install Neo4j Enterprise Edition as `root`:

     ```
     zypper install neo4j-enterprise-4.4.48
     ```

## Offline installation

If you cannot reach `https://yum.neo4j.com/stable` to install Neo4j using RPM, perhaps due to a firewall, you will 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 will mean that the offline machine will not receive the dependencies that are normally downloaded and installed automatically when using `yum` for installing Neo4j; Neo4j Cypher Shell and Java.

For information on supported versions of Java, see System requirements.

### Downloading the RPM installers

The Cypher Shell RPM package can be downloaded from Neo4j Deployment Center.

1. Run the following to obtain the required Neo4j RPM package:
   - Neo4j Enterprise Edition:

     ```
     curl -O https://dist.neo4j.org/rpm/neo4j-enterprise-4.4.48-1.noarch.rpm
     ```

   - Neo4j Community Edition:

     ```
     curl -O https://dist.neo4j.org/rpm/neo4j-4.4.48-1.noarch.rpm
     ```

2. Manually move the downloaded RPM packages to the offline machine.

If using Oracle Java 11, the same dependency issues apply as with the standard installation. You will need to additionally download and install the Java adaptor described in that section:

- To install Neo4j Enterprise Edition as `root`:

  ```
  curl -O https://dist.neo4j.org/neo4j-java11-adapter.noarch.rpm
  ```

### Performing an offline installation

#### Offline upgrade from 4.0.0 or later

- Neo4j 4.0.0 and onwards already require Java 11, so there should be no additional Java setup required.
- Neo4j Cypher Shell must be installed *before* Neo4j, because it is a dependency.
- Run the following on the offline machine to install Neo4j Cypher Shell and Neo4j simultaneously:

  ```
  rpm -U <Cypher Shell RPM file name> <Neo4j RPM file name>
  ```

  This must be one single command, and Neo4j Cypher Shell must be the first package in the command.

#### Offline upgrade from 3.5 or earlier

- Due to dependency conflicts with older versions, for offline upgrades from 3.5 or earlier, Neo4j Cypher Shell and Neo4j must be upgraded simultaneously.
- Before you begin, you will need to have Java 11 pre-installed. For Oracle Java 11 only, you must install the Oracle Java adapter before trying to install Neo4j.
- Run the following on the offline machine to install Neo4j Cypher Shell and Neo4j simultaneously:

  ```
  rpm -U <Cypher Shell RPM file name> <Neo4j RPM file name>
  ```

  This must be one single command, and Neo4j Cypher Shell must be the first package in the command.

## Starting the 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
---
```

## 2.2.3. 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 11 or Oracle Java 11.
2. Download the latest Neo4j tarball from Neo4j Deployment Center and unpack it:

```
tar zxf neo4j-enterprise-4.4.48-unix.tar.gz
```

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

```
mv neo4j-enterprise-4.4.48 /opt/
ln -s /opt/neo4j-enterprise-4.4.48 /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-4.4.48
     ```

   - **RedHat**

     ```
     chown -R neo4j /opt/neo4j-enterprise-4.4.48
     ```

6. 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.

7. Start Neo4j:
   a. To run Neo4j as a console application, use: `<NEO4J_HOME>/bin/neo4j console`.
   b. To run Neo4j in a background process, use: `<NEO4J_HOME>/bin/neo4j start`.
8. Open *http://localhost:7474* in your web browser.
9. Connect using the username `neo4j` with your password or with the default password `neo4j`. In the latter case you will be prompted to change the password.
10. 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.

```
user@localhost:~$ ulimit -n
1024
```

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.

```
user@localhost:~$ sudo systemctl edit neo4j.service
```

2. Append the `[Service]` section to the *neo4j.service* file.

```
[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, for example, Vim.

```
user@localhost:~$ 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.

```
user@localhost:~$ sudo vi /etc/security/limits.conf
```

4. Define the following values:

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

5. Reload the `systemd` settings.

```
user@localhost:~$ sudo systemctl daemon-reload
```

6. Reboot your machine.

# Chapter 3. 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
---
```

## 3.1. Neo4j system service



**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 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
---
```

## 3.2. macOS installation

### 3.2.1. Unix console application

1. If it is not already installed, get OpenJDK 11 or Oracle Java 11.

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-4.4.48-unix.tar.gz`.
5. Place the extracted files in a permanent home on your server. The top level directory is referred to as `NEO4J_HOME`.
6. 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.

7. Start Neo4j:
   a. To run Neo4j as a console application, use: `<NEO4J_HOME>/bin/neo4j console`.
   b. To run Neo4j in a background process, use: `<NEO4J_HOME>/bin/neo4j start`.
8. Visit *http://localhost:7474* in your web browser.
9. Connect using the username 'neo4j' with your password of with the default password 'neo4j'. In the latter case, you will be prompted to change the password.
10. Stop the server by typing `Ctrl-C` in the console.

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

## 3.2.2. macOS service

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

## 3.2.3. 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:

```
dbms.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.

### 3.2.4. 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
---
```

## 3.3. Windows installation

### 3.3.1. Windows console application

1. If it is not already installed, get OpenJDK 11 or Oracle Java 11.
2. Download the latest release from Neo4j Deployment Center.

   Select the appropriate ZIP distribution.

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. Right-click the downloaded file, click **Extract All**.
5. Place the extracted files in a permanent home on your server, for example `D:\neo4j\`. The top level directory is referred to as `NEO4J_HOME`.

6. 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.

7. Start Neo4j:
   a. To run Neo4j as a console application, use: `<NEO4J_HOME>\bin\neo4j console`.
   b. To install Neo4j as a service use: `<NEO4J_HOME>\bin\neo4j install-service`.
   c. For additional commands and to learn about the Windows PowerShell module included in the Zip file, see Windows PowerShell module.
8. Visit *http://localhost:7474* in your web browser.
9. Connect using the username 'neo4j' with your password or with the default password 'neo4j'. In the latter case, you'll be prompted to change the password.
10. Stop the server by typing `Ctrl-C` in the console.

## 3.3.2. Windows service

Neo4j can also be run as a Windows service. Install the service with `bin\neo4j install-service`, and start it with `bin\neo4j start`.

Don't forget to set your initial password.

The available commands for `bin\neo4j` are: `help`, `start`, `stop`, `restart`, `status`, `install-service`, `uninstall-service`, and `update-service`.



**Note:**

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

### Java options

When Neo4j is installed as a service, Java options are stored in the service configuration. Changes to these options after the service is installed will not take effect until the service configuration is updated. For example, changing the setting `dbms.memory.heap.max_size` in *neo4j.conf* will not take effect until the service is updated and restarted. To update the service, run `bin\neo4j update-service`. 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 install-service
```

2. Change memory configuration

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

3. Update service

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

4. Restart service

```
bin\neo4j restart
```

## 3.3.3. 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.

### Managing Neo4j on Windows

On Windows, it is sometimes necessary to *Unblock* a downloaded ZIP file before you can import its contents as a module. If you right-click on the ZIP file and choose "Properties" you will get a dialog which includes an "Unblock" button, which will enable you to import 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 will display a warning if it detects that you do not have administrative rights.

## How do I import the module?

The module file is located in the *bin* directory of your Neo4j installation, i.e. where you unzipped the downloaded file. For example, if Neo4j was installed in *C:\Neo4j* then the module would be imported like this:

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

This will add the module to the current session.

Once the module has been imported you can start an interactive console version of a Neo4j Server like this:

```
Invoke-Neo4j console
```

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

## How do I get help about the module?

Once the module is imported you can query the available commands like this:

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

The output should be similar to the following:

```
CommandType     Name                                Version    Source
-----------     ----                                -------    ------
Function        Invoke-Neo4j                        4.4.48      Neo4j-Management
Function        Invoke-Neo4jAdmin                   4.4.48      Neo4j-Management
Function        Invoke-Neo4jBackup                  4.4.48      Neo4j-Management
Function        Invoke-Neo4jImport                  4.4.48      Neo4j-Management
Function        Invoke-Neo4jShell                   4.4.48      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 install-service -Verbose
```

- Available commands for administrative tasks:

```
Invoke-Neo4jAdmin
```

## Common PowerShell parameters

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

# Chapter 4. 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
---
```

## 4.1. 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 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.

# Chapter 5. Docker

This chapter describes the following:

- Get started with Neo4j in Docker — Introduction to running Neo4j in a Docker container.
- Persist 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 Causal Clustering when using Docker.
- Docker specific operations - Descriptions of various `neo4j-admin` and `cypher-shell` operations that are specific to using Docker.
- Security - Information about using encryption with the Docker image.
- Docker maintenance operations How to maintain Neo4j when running in a Docker container.
- 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.

## 5.1. Get 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.

### 5.1.1. Neo4j editions

Tags are available for both Community Edition and Enterprise Edition. Version-specific Enterprise Edition tags have an `-enterprise` suffix, for example: `neo4j:4.4.48-enterprise`. Community Edition tags have no suffix, for example `neo4j:4.4.48`. The latest Neo4j Enterprise Edition release is available as `neo4j:enterprise`.

All supported tags can be found at https://hub.docker.com/_/neo4j/tags.

*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`.

© Network Engine for Objects in Lund AB. 2022. All Rights Reserved. Use of this Software without a proper commercial license with Neo4j, Inc. or its affiliates is prohibited.

Email inquiries can be directed to: licensing@neo4j.com

More information is also available at: https://neo4j.com/licensing/

### 5.1.2. 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:4.4.48
```

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.

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.

### 5.1.3. Using `NEO4J_AUTH` to set an initial password

When using Neo4j in a Docker container, you can set the initial password for the container 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:4.4.48
```

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

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

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

### 5.1.4. Persisting data using Volumes

The `--volume` option maps a local folder to the container, where you can persist data between restarts.

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

```
    --env NEO4J_AUTH=neo4j/your_password \
    --volume /path/to/your/data:/data \
    --volume= /path/to/your/logs:/logs \
    neo4j:4.4.48
```

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



**Note:**

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

## 5.1.5. Running Neo4j as a non-root user

For security reasons, Neo4j runs as the `neo4j` user inside the container. You can specify which user to run as by invoking docker with the `--user` argument. For example, the following runs Neo4j as your current user:

```
docker run \
    --publish=7474:7474 --publish=7687:7687 \
    --user="$(id -u):$(id -g)" \
    neo4j:4.4.48
```

## 5.1.6. More 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 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/test` |
| `--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`:

```
docker update --restart=no <containerID>
```

For more information on Docker restart policies, see The official Docker documentation.

### 5.1.7. 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:4.4.48` image to a `.tar` file:

```
docker save -o neo4j-4.4.48.tar neo4j:4.4.48
```

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

```
docker load --input neo4j-4.4.48.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.

## 5.2. Persist 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.

### 5.2.1. 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.<br>See Modify the default configuration . |
| `/plugins` | read[2] | Allows you to install plugins in containerized Neo4j.<br>See Plugins. |
| `/licenses` | read | Provide licenses for Neo4j and any plugins by mounting the license folder.<br>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.<br>See Security |
| `/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:4.4.48
```

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

```
docker volume create neo4jdata
docker run -it --rm \
   --volume neo4jdata:/data \
   neo4j:4.4.48
```

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.

## 5.2.2. 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.

## 5.3. Modify the default configuration

The default configuration provided by this 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_dbms_memory_pagecache_size=512M` and `NEO4J_dbms_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.

### 5.3.1. Environment variables

Pass environment variables to the container when you run it.

```
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:4.4.48
```

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, `dbms.tx_log.rotation.size` could be set by specifying the following argument to Docker:

```
--env NEO4J_dbms_tx__log_rotation_size
```

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

```
--env NEO4J_dbms_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"
```

### 5.3.2. 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:4.4.48
```

Any configuration files in the */conf* volume will override 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 will still override the values in configuration files in */conf* volume.



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

To dump an initial set of configuration files, run the image with the `dump-config` command.

```
docker run --rm \
    --volume=$HOME/neo4j/conf:/conf \
    neo4j:4.4.48 dump-config
```

## 5.3.3. 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:4.4.48-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:4.4.48-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/password --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes neo4j:4.4.48-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.

## 5.4. 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.

### 5.4.1. 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:4.4.48
```

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

### 5.4.2. `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
graph-data-science
graphql
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:4.4.48
```

*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:4.4.48
```

## 5.4.3. 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:4.4.48
```

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:4.4.48

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

## 5.4.4. 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:4.4.48
```

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:4.4.48
```

A folder containing license files.

# 5.5. Deploy a Neo4j standalone server using Docker Compose

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.

## 5.5.1. 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:4.4.48
       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 Persist 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.

## 5.5.2. 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:4.4.48
    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.

## 5.6. Deploy a Neo4j cluster on Docker

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

## 5.6.1. Deploy a Causal Cluster with Docker Compose

You can deploy a Causal 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 Causal Cluster members 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 core and replica 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.
dbms.memory.pagecache.size=100M

# Setting that specifies the initial JVM heap size.
dbms.memory.heap.initial_size=100M

# Strategy that the instance will use to determine the addresses of other members.
causal_clustering.discovery_type=DNS

# The network addresses of an initial set of Core cluster members that are available to bootstrap
this Core or Read Replica instance.
# If the DNS strategy is used, the addresses are fetch using the DNS A records.
causal_clustering.initial_discovery_members=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.
causal_clustering.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
Core cluster.
causal_clustering.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 catchup protocol.
causal_clustering.transaction_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.
dbms.routing.advertised_address=$(hostname)
```

2. Prepare your *docker-compose.yml* file using the following example. For more information, see the Docker Compose official Service configuration reference.

*Example 8. Example docker-compose.yml file*

```
version: '3.8'

# Custom top-level network
networks:
  neo4j-internal:

services:

  core1:
    # Docker image to be used
    \image: ${NEO4J_DOCKER_IMAGE}

    # Hostname
    hostname: core1

    # 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/core1:/var/lib/neo4j/data
      - ./logs/core1:/var/lib/neo4j/logs
      - ./conf/core1:/var/lib/neo4j/conf
      - ./import/core1:/var/lib/neo4j/import
      #- ./metrics/core1:/var/lib/neo4j/metrics
      #- ./licenses/core1:/var/lib/neo4j/licenses
      #- ./ssl/core1:/var/lib/neo4j/ssl

    # Passes the following environment variables to the container
    environment:
      - NEO4J_ACCEPT_LICENSE_AGREEMENT
      - NEO4J_AUTH
      - EXTENDED_CONF
      - NEO4J_EDITION
      - NEO4J_dbms_mode=CORE

    # 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}

  core2:
    \image: ${NEO4J_DOCKER_IMAGE}
    hostname: core2
    networks:
      neo4j-internal:
```

```
      aliases:
        - neo4j-network
  ports:
    - "7475:7474"
    - "7688:7687"
  volumes:
    - ./neo4j.conf:/conf/neo4j.conf
    - ./data/core2:/var/lib/neo4j/data
    - ./logs/core2:/var/lib/neo4j/logs
    - ./conf/core2:/var/lib/neo4j/conf
    - ./import/core2:/var/lib/neo4j/import
    #- ./metrics/core2:/var/lib/neo4j/metrics
    #- ./licenses/core2:/var/lib/neo4j/licenses
    #- ./ssl/core2:/var/lib/neo4j/ssl
  environment:
    - NEO4J_ACCEPT_LICENSE_AGREEMENT
    - NEO4J_AUTH
    - EXTENDED_CONF
    - NEO4J_EDITION
    - NEO4J_dbms_mode=CORE
  healthcheck:
    test: ["CMD-SHELL", "wget --no-verbose --tries=1 --spider localhost:7474 || exit 1"]
  user: ${USER_ID}:${GROUP_ID}

core3:
  \image: ${NEO4J_DOCKER_IMAGE}
  hostname: core3
  networks:
    neo4j-internal:
      aliases:
        - neo4j-network
  ports:
    - "7476:7474"
    - "7689:7687"
  volumes:
    - ./neo4j.conf:/conf/neo4j.conf
    - ./data/core3:/var/lib/neo4j/data
    - ./logs/core3:/var/lib/neo4j/logs
    - ./conf/core3:/var/lib/neo4j/conf
    - ./import/core3:/var/lib/neo4j/import
    #- ./metrics/core3:/var/lib/neo4j/metrics
    #- ./licenses/core3:/var/lib/neo4j/licenses
    #- ./ssl/core3:/var/lib/neo4j/ssl
  environment:
    - NEO4J_ACCEPT_LICENSE_AGREEMENT
    - NEO4J_AUTH
    - EXTENDED_CONF
    - NEO4J_EDITION
    - NEO4J_dbms_mode=CORE
  healthcheck:
    test: ["CMD-SHELL", "wget --no-verbose --tries=1 --spider localhost:7474 || exit 1"]
  user: ${USER_ID}:${GROUP_ID}

readreplica1:
  \image: ${NEO4J_DOCKER_IMAGE}
  hostname: replica1
  networks:
    neo4j-internal:
      aliases:
        - neo4j-network
  ports:
    - "7477:7474"
    - "7690:7687"
  volumes:
    - ./neo4j.conf:/conf/neo4j.conf
    - ./data/replica1:/var/lib/neo4j/data
    - ./logs/replica1:/var/lib/neo4j/logs
    - ./conf/replica1:/var/lib/neo4j/conf
    - ./import/replica1:/var/lib/neo4j/import
    #- ./metrics/replica1:/var/lib/neo4j/metrics
    #- ./licenses/replica1:/var/lib/neo4j/licenses
    #- ./ssl/replica1:/var/lib/neo4j/ssl
  environment:
    - NEO4J_ACCEPT_LICENSE_AGREEMENT
    - NEO4J_AUTH
    - EXTENDED_CONF
    - NEO4J_EDITION
```

```
      - NEO4J_dbms_mode=READ_REPLICA
    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:4.4-enterprise`
   - `export NEO4J_EDITION=docker_compose`
   - `export EXTENDED_CONF=yes`
   - `export NEO4J_ACCEPT_LICENSE_AGREEMENT=yes`
   - `export NEO4J_AUTH=neo4j/your_password`
4. Deploy your Causal Cluster by running `docker-compose up` from your project folder.
5. Open *core1* at *http://core1-public-address:7474*.
6. Authenticate with the default `neo4j/your_password` credentials.
7. Check the status of the cluster by running the following in Neo4j Browser:

```
:sysinfo
```

## 5.6.2. Deploy a Causal 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, and the `NEO4J_causal__clustering_expected__core__cluster__size` and `NEO4J_causal__clustering_initial__discovery__members` environment variables must be set for Cores. Read Replicas only need to define `NEO4J_causal__clustering_initial__discovery__members`.

### Causal Cluster environment variables

The following environment variables are specific to Causal Clustering, and are available in the Neo4j Enterprise Edition:

- `NEO4J_dbms_mode`: the database mode, defaults to `SINGLE`, set to `CORE` or `READ_REPLICA` for Causal Clustering.
- `NEO4J_causal__clustering_expected__core__cluster__size`: the initial cluster size (number of Core instances) at startup.
- `NEO4J_causal__clustering_initial__discovery__members`: the network addresses of an initial set of Core cluster members.
- `NEO4J_causal__clustering_discovery__advertised__address`: hostname/IP address and port to advertise for member discovery management communication.
- `NEO4J_causal__clustering_transaction__advertised__address`: hostname/IP address and port to

advertise for transaction handling.

- `NEO4J_causal__clustering_raft__advertised__address`: hostname/IP address and port to advertise for cluster communication.

See Settings reference for more details of Neo4j Causal Clustering settings.

## Set up a Causal 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 3 COREs:

```
docker network create --driver=bridge cluster

docker run --name=core1 --detach --network=cluster \
    --publish=7474:7474 --publish=7473:7473 --publish=7687:7687 \
    --hostname=core1 \
    --env NEO4J_dbms_mode=CORE \
    --env NEO4J_causal__clustering_expected__core__cluster__size=3 \
    --env NEO4J_causal__clustering_initial__discovery__members=core1:5000,core2:5000,core3:5000 \
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
    --env NEO4J_dbms_connector_bolt_advertised__address=localhost:7687 \
    --env NEO4J_dbms_connector_http_advertised__address=localhost:7474 \
    neo4j:4.4.48-enterprise

docker run --name=core2 --detach --network=cluster \
    --publish=8474:7474 --publish=8473:7473 --publish=8687:7687 \
    --hostname=core2 \
    --env NEO4J_dbms_mode=CORE \
    --env NEO4J_causal__clustering_expected__core__cluster__size=3 \
    --env NEO4J_causal__clustering_initial__discovery__members=core1:5000,core2:5000,core3:5000 \
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
    --env NEO4J_dbms_connector_bolt_advertised__address=localhost:8687 \
    --env NEO4J_dbms_connector_http_advertised__address=localhost:8474 \
    neo4j:4.4.48-enterprise

docker run --name=core3 --detach --network=cluster \
    --publish=9474:7474 --publish=9473:7473 --publish=9687:7687 \
    --hostname=core3 \
    --env NEO4J_dbms_mode=CORE \
    --env NEO4J_causal__clustering_expected__core__cluster__size=3 \
    --env NEO4J_causal__clustering_initial__discovery__members=core1:5000,core2:5000,core3:5000 \
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
    --env NEO4J_dbms_connector_bolt_advertised__address=localhost:9687 \
    --env NEO4J_dbms_connector_http_advertised__address=localhost:9474 \
    neo4j:4.4.48-enterprise
```

Additional instances can be added to the cluster in an ad-hoc fashion.

Example of a `docker run` command for adding a Read Replica to the cluster:

```
docker run --name=read-replica1 --detach --network=cluster \
         --publish=10474:7474 --publish=10473:7473 --publish=10687:7687 \
         --hostname=read-replica1 \
         --env NEO4J_dbms_mode=READ_REPLICA \
         --env NEO4J_causal__clustering_initial__discovery__members=core1:5000,core2:5000,core3:5000 \
         --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
         --env NEO4J_dbms_connector_bolt_advertised__address=localhost:10687 \
         --env NEO4J_dbms_connector_http_advertised__address=localhost:10474 \
         neo4j:4.4.48-enterprise
```

### Set up a Causal Cluster on multiple Docker hosts

To get the Causal Cluster high-availability characteristics, however, it is more sensible to put the cluster nodes 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 the 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=neo4j-core --detach \
         --network=host \
         --publish=7474:7474 --publish=7687:7687 \
         --publish=5000:5000 --publish=6000:6000 --publish=7000:7000 \
         --hostname=public-address \
         --env NEO4J_dbms_mode=CORE \
         --env NEO4J_causal__clustering_expected__core__cluster__size=3 \
         --env NEO4J_causal__clustering_initial__discovery__members=core1-public-address:5000,core2-
public-address:5000,core3-public-address:5000 \
         --env NEO4J_causal__clustering_discovery__advertised__address=public-address:5000 \
         --env NEO4J_causal__clustering_transaction__advertised__address=public-address:6000 \
         --env NEO4J_causal__clustering_raft__advertised__address=public-address:7000 \
         --env NEO4J_dbms_connectors_default__advertised__address=public-address \
         --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
         --env NEO4J_dbms_connector_bolt_advertised__address=public-address:7687 \
         --env NEO4J_dbms_connector_http_advertised__address=public-address:7474 \
         neo4j:4.4.48-enterprise
```

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

> **Note:**
>
> Please note that if you are starting a Read Replica as above, you must publish the discovery port. For example, `--publish=5000:5000`.
>
> In versions prior to Neo4j 4.0, this was only necessary with Core servers.

# 5.7. Docker specific operations

## 5.7.1. 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 <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 tool.

## 5.7.2. Use Neo4j Import

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

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

For more information about the `neo4j-admin import` syntax and options, see Syntax and Options.

### 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 import` 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 \
    --user="$(id -u):$(id -g)" \
    neo4j:4.4.48 \
neo4j-admin import --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
```

## 5.7.3. Use Neo4j Admin for memory recommendations

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

*Example 9. Invoke* `neo4j-admin memrec --docker`

```
$neo4j-home> bin/neo4j-admin memrec --memory=16g --docker

...
...
...
# Based on the above, the following memory settings are recommended:
NEO4J_dbms_memory_heap_initial__size=5g
NEO4J_dbms_memory_heap_max__size=5g
NEO4J_dbms_memory_pagecache_size=7g
```

## 5.7.4. 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=$HOME/neo4j/data:/data \
       --user="$(id -u):$(id -g)" \
       neo4j:4.4.48
   ```

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/<password> \
neo4j:4.4.48

# 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/<password> \
neo4j:4.4.48

# 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.

## 5.7.5. Install user-defined procedures

To install user-defined procedures, mount the */plugins* volume containing the jars.

```
docker run --publish=7474:7474 --publish=7687:7687 --volume=$HOME/neo4j/plugins:/plugins neo4j:4.4.48
```

## 5.7.6. Configure Neo4j Labs plugins

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



**Note:**

This feature is intended to facilitate using Neo4j Labs plugins in development environments, but it is not recommended for use in production environments.

To use plugins in production with Neo4j Docker containers, see Install user-defined procedures.

The `NEO4JLABS_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 supported plugins.

For example, to install the APOC plugin (`apoc`), you can use the Docker argument;

```
--env NEO4JLABS_PLUGINS='["apoc"]'
```

and run the following command:

```
docker run -it --rm \
  --publish=7474:7474 --publish=7687:7687 \
  --user="$(id -u):$(id -g)" \
  -e NEO4J_AUTH=none \
  --env NEO4JLABS_PLUGINS='["apoc"]' \
```

```
neo4j:4.4.48
```

For example, to install the APOC plugin (`apoc`) and the Neo Semantics plugin (`n10s`), you can use the following Docker argument:

```
--env NEO4JLABS_PLUGINS='["apoc", "n10s"]'
```

*Table 12. Supported Neo4j Labs plugins*

| Name | Key | Further information |
|---|---|---|
| APOC | `apoc` | APOC Full |
| APOC Core | `apoc-core` | APOC Core |
| Bloom | `bloom` | Neo4j Bloom |
| Streams | `streams` | Neo4j Streaming Data Integrations User Guide |
| Graph Data Science | `graph-data-science` | Graph Data Science |
| Neo Semantics | `n10s` | https://neo4j.com/labs/nsmtx-rdf/ |



**Note:**

Running Bloom in a Docker container requires Neo4j Docker image 4.2.3-enterprise or later.

# 5.8. Security

## 5.8.1. SSL Encryption

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 you 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
dbms.connector.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 10. 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:4.4.48
```

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 11. 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:4.4.48
```

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 `dbms.connector.https.enabled` to `true`, to be able Neo4j 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.

## 5.9. Docker maintenance operations

### 5.9.1. Dump and load a Neo4j database (offline)

The `neo4j-admin dump` and `neo4j-admin 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 12. Invoke `neo4j-admin dump` to dump your database.*

```
docker run --interactive --tty --rm \
   --volume=$HOME/neo4j/data:/data \
   --volume=$HOME/neo4j/backups:/backups \
   neo4j/neo4j-admin:4.4.48 \
neo4j-admin dump --database=neo4j --to=/backups/<dump-name>.dump
```

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

The volume that will be used for the dumped database.

*Example 13. Invoke `neo4j-admin load` to load your data into the new database.*

```
docker run --interactive --tty --rm \
    --volume=$HOME/neo4j/data:/data \
    --volume=$HOME/neo4j/backups:/backups \
    neo4j/neo4j-admin:4.4.48 \
neo4j-admin load --database=neo4j --from=/backups/<dump-name>.dump
```

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.



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

### 5.9.2. 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 14. 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_dbms_backup_enabled=true \
    neo4j:4.4.48-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 states that you have accepted the Neo4j Enterprise Edition license agreement.

The environment variable that enables online backups.

*Example 15. Invoke `neo4j-admin backup` to back up an online database, using `docker exec`:*

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

For more information on the `neo4j-admin 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 node 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 16. 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_dbms_backup_enabled=true \
    --env NEO4J_dbms_backup_listen__address=0.0.0.0:6362 \
    neo4j:4.4.48-enterprise
```

The dbms.backup.listen_address port defined in 5.

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

The environment variable that states that you have accepted the Neo4j Enterprise Edition license agreement.

The environment variable that enables online backups.

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

Once you have a backup enabled cluster node, the `neo4j/neo4j-admin:4.4.48-enterprise` docker image can be used to backup the database.

*Example 17. Invoke `neo4j-admin` docker image to backup your database.*

```
docker run --interactive --tty --rm \
   --volume=$HOME/neo4j-enterprise/backups:/backups \
   --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
   neo4j/neo4j-admin:4.4.48-enterprise \
      neo4j-admin backup --database=<database name> \
         --backup-dir=/backups \
         --from=<backup node IP address>:6362
```

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

The environment variable that states that you have accepted the Neo4j Enterprise Edition license agreement.

The IP address of the backup cluster node and the dbms.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 18. 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:4.4.48-enterprise
```

The volume that contains all your databases.

The volume that contains the database backup.

The environment variable that states that you have accepted the Neo4j Enterprise Edition license agreement.

*Example 19. 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 20. Invoke `neo4j-admin restore` to restore a database backup.*

```
docker exec --interactive --tty <containerID/name> neo4j-admin restore --from=/backups/<database
backup name> --database=<database name>
```

## Restore a database using `neo4j-admin` image

The `neo4j-admin 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 21. 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:4.4.48-enterprise
```

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

The environment variable that states that you have accepted the Neo4j Enterprise Edition license agreement.

*Example 22. Stop the old database, then restore the backup database using `neo4j/neo4j-admin:4.4.48-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 restore \
         --database=<database name> \
         --from=/backups/<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` database container is using.

The volume that contains the database backup.

The environment variable that states that you have accepted the Neo4j Enterprise Edition license agreement.



**Note:**

For more information on the `neo4j-admin 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.

## 5.9.3. Upgrade Neo4j on Docker

The following is an example of a `docker run` command that launches a container and upgrades a Neo4j database stored in a Docker volume or a host folder.

```
docker run \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j/data:/data \
    --env NEO4J_dbms_allow__upgrade=true \
    neo4j:4.4.48 \
```

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

The environment variable that enables the upgrade.

The new version of the Neo4j Docker image to which you want to upgrade your database.



**Note:**

The upgrade to a later patch release of Neo4j 4.4 is straightforward — stop the container and then restart it using the later Neo4j docker image. For more details on upgrading, see Upgrade and Migration Guide → Upgrade to a newer PATCH release.

## 5.9.4. Monitor Neo4j

Neo4j logging output is written to files in the */logs* directory. This directory is mounted as a */logs* volume.



**Tip:**

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

Since a docker instance is run as `neo4j console`, you would not normally expect to see *neo4j.log* in the */logs* directory. However, you can still get it by running:

```
docker logs <containerID/name>
```

It is also possible to configure Neo4j to write the logs to a file by setting the configuration `NEO4J_dbms_logs_user_stdout__enabled=true` as an environment variable.

## 5.10. 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 complete 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` |
| `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` |
| `causal_clustering.catch_up_client_inactivity_timeout` | `NEO4J_causal__clustering_catch__up__client__inactivity__timeout` |
| `causal_clustering.catchup_batch_size` | `NEO4J_causal__clustering_catchup__batch__size` |
| `causal_clustering.cluster_allow_reads_on_followers` | `NEO4J_causal__clustering_cluster__allow__reads__on__followers` |
| `causal_clustering.cluster_binding_timeout` | `NEO4J_causal__clustering_cluster__binding__timeout` |
| `causal_clustering.cluster_topology_refresh` | `NEO4J_causal__clustering_cluster__topology__refresh` |
| `causal_clustering.command_applier_parallelism` | `NEO4J_causal__clustering_command__applier__parallelism` |
| `causal_clustering.connect-randomly-to-server-group` | `NEO4J_causal__clustering_connect-randomly-to-server-group` |
| `causal_clustering.discovery_advertised_address` | `NEO4J_causal__clustering_discovery__advertised__address` |
| `causal_clustering.discovery_listen_address` | `NEO4J_causal__clustering_discovery__listen__address` |
| `causal_clustering.discovery_type` | `NEO4J_causal__clustering_discovery__type` |

| Neo4j format | Docker format |
| --- | --- |
| causal_clustering.election_failure_detection_window | NEO4J_causal__clustering_election__failure__detection__window |
| causal_clustering.enable_pre_voting | NEO4J_causal__clustering_enable__pre__voting |
| causal_clustering.global_session_tracker_state_size | NEO4J_causal__clustering_global__session__tracker__state__size |
| causal_clustering.handshake_timeout | NEO4J_causal__clustering_handshake__timeout |
| causal_clustering.in_flight_cache.max_entries | NEO4J_causal__clustering_in__flight__cache_max__entries |
| causal_clustering.in_flight_cache.type | NEO4J_causal__clustering_in_flight_cache_type |
| causal_clustering.initial_discovery_members | NEO4J_causal__clustering_initial__discovery__members |
| causal_clustering.join_catch_up_max_lag | NEO4J_causal__clustering_join__catch__up__max__lag |
| causal_clustering.join_catch_up_timeout | NEO4J_causal__clustering_join__catch__up__timeout |
| causal_clustering.kubernetes.address | NEO4J_causal__clustering_kubernetes_address |
| causal_clustering.kubernetes.ca_crt | NEO4J_causal__clustering_kubernetes_ca__crt |
| causal_clustering.kubernetes.label_selector | NEO4J_causal__clustering_kubernetes_label__selector |
| causal_clustering.kubernetes.namespace | NEO4J_causal__clustering_kubernetes_namespace |
| causal_clustering.kubernetes.service_port_name | NEO4J_causal__clustering_kubernetes_service_port_name |
| causal_clustering.kubernetes.token | NEO4J_causal__clustering_kubernetes_token |
| causal_clustering.last_applied_state_size | NEO4J_causal__clustering_last__applied__state__size |
| causal_clustering.leader_election_timeout | NEO4J_causal__clustering_leader__election__timeout |
| causal_clustering.leader_failure_detection_window | NEO4J_causal__clustering_leader__failure__detection__window |
| causal_clustering.leadership_balancing | NEO4J_causal__clustering_leadership__balancing |
| causal_clustering.load_balancing.plugin | NEO4J_causal__clustering_load__balancing_plugin |
| causal_clustering.load_balancing.shuffle | NEO4J_causal__clustering_load__balancing_shuffle |
| causal_clustering.log_shipping_max_lag | NEO4J_causal__clustering_log__shipping__max__lag |
| causal_clustering.log_shipping_retry_timeout | NEO4J_causal__clustering_log__shipping__retry__timeout |
| causal_clustering.middleware.logging.level | NEO4J_causal__clustering_middleware_logging_level |
| causal_clustering.minimum_core_cluster_size_at_formation | NEO4J_causal__clustering_minimum__core__cluster__size__at__formation |
| causal_clustering.minimum_core_cluster_size_at_runtime | NEO4J_causal__clustering_minimum__core__cluster__size__at__runtime |
| causal_clustering.multi_dc_license | NEO4J_causal__clustering_multi__dc__license |

| Neo4j format | Docker format |
| --- | --- |
| causal_clustering.protocol_implementations.catchup | NEO4J_causal__clustering_protocol__implementations_catchup |
| causal_clustering.protocol_implementations.compression | NEO4J_causal__clustering_protocol__implementations_compression |
| causal_clustering.protocol_implementations.raft | NEO4J_causal__clustering_protocol__implementations_raft |
| causal_clustering.pull_interval | NEO4J_causal__clustering_pull__interval |
| causal_clustering.raft_advertised_address | NEO4J_causal__clustering_raft__advertised__address |
| causal_clustering.raft_handler_parallelism | NEO4J_causal__clustering_raft__handler__parallelism |
| causal_clustering.raft_in_queue_max_bytes | NEO4J_causal__clustering_raft__in__queue__max__bytes |
| causal_clustering.raft_listen_address | NEO4J_causal__clustering_raft__listen__address |
| causal_clustering.raft_log_implementation | NEO4J_causal__clustering_raft__log__implementation |
| causal_clustering.raft_log_prune_strategy | NEO4J_causal__clustering_raft__log__prune__strategy |
| causal_clustering.raft_log_pruning_frequency | NEO4J_causal__clustering_raft__log__pruning__frequency |
| causal_clustering.raft_log_reader_pool_size | NEO4J_causal__clustering_raft__log__reader__pool__size |
| causal_clustering.raft_log_rotation_size | NEO4J_causal__clustering_raft__log__rotation__size |
| causal_clustering.raft_membership_state_size | NEO4J_causal__clustering_raft__membership__state__size |
| causal_clustering.raft_term_state_size | NEO4J_causal__clustering_raft__term__state__size |
| causal_clustering.raft_vote_state_size | NEO4J_causal__clustering_raft__vote__state__size |
| causal_clustering.refuse_to_be_leader | NEO4J_causal__clustering_refuse__to__be__leader |
| causal_clustering.replicated_lease_state_size | NEO4J_causal__clustering_replicated__lease__state__size |
| causal_clustering.replication_leader_await_timeout | NEO4J_causal__clustering_replication__leader__await__timeout |
| causal_clustering.replication_retry_timeout_base | NEO4J_causal__clustering_replication__retry__timeout__base |
| causal_clustering.replication_retry_timeout_limit | NEO4J_causal__clustering_replication__retry__timeout__limit |
| causal_clustering.server_groups | NEO4J_causal__clustering_server__groups |
| causal_clustering.state_machine_apply_max_batch_size | NEO4J_causal__clustering_state__machine__apply__max__batch__size |
| causal_clustering.state_machine_flush_window_size | NEO4J_causal__clustering_state__machine__flush__window__size |
| causal_clustering.status_throughput_window | NEO4J_causal__clustering_status__throughput__window |
| causal_clustering.store_copy_chunk_size | NEO4J_causal__clustering_store__copy__chunk__size |

| Neo4j format | Docker format |
| --- | --- |
| causal_clustering.store_copy_max_retry_time_per_request | NEO4J_causal__clustering_store__copy__max__retry__time__per__request |
| causal_clustering.transaction_advertised_address | NEO4J_causal__clustering_transaction__advertised__address |
| causal_clustering.transaction_listen_address | NEO4J_causal__clustering_transaction__listen__address |
| causal_clustering.unknown_address_logging_throttle | NEO4J_causal__clustering_unknown__address__logging__throttle |
| causal_clustering.upstream_selection_strategy | NEO4J_causal__clustering_upstream__selection__strategy |
| causal_clustering.user_defined_upstream_strategy | NEO4J_causal__clustering_user__defined__upstream__strategy |
| cypher.default_language_version | NEO4J_cypher_default__language__version |
| cypher.forbid_exhaustive_shortestpath | NEO4J_cypher_forbid__exhaustive__shortestpath |
| cypher.forbid_shortestpath_common_nodes | NEO4J_cypher_forbid__shortestpath__common__nodes |
| cypher.hints_error | NEO4J_cypher_hints__error |
| cypher.lenient_create_relationship | NEO4J_cypher_lenient__create__relationship |
| cypher.min_replan_interval | NEO4J_cypher_min__replan__interval |
| cypher.planner | NEO4J_cypher_planner |
| cypher.statistics_divergence_threshold | NEO4J_cypher_statistics__divergence__threshold |
| db.temporal.timezone | NEO4J_db_temporal_timezone |
| dbms.allow_single_automatic_upgrade | NEO4J_dbms_allow__single__automatic__upgrade |
| dbms.allow_upgrade | NEO4J_dbms_allow__upgrade |
| dbms.backup.enabled | NEO4J_dbms_backup_enabled |
| dbms.backup.listen_address | NEO4J_dbms_backup_listen__address |
| dbms.checkpoint | NEO4J_dbms_checkpoint |
| dbms.checkpoint.interval.time | NEO4J_dbms_checkpoint_interval_time |
| dbms.checkpoint.interval.tx | NEO4J_dbms_checkpoint_interval_tx |
| dbms.checkpoint.iops.limit | NEO4J_dbms_checkpoint_iops_limit |
| dbms.config.strict_validation | NEO4J_dbms_config_strict__validation |
| dbms.connector.bolt.advertised_address | NEO4J_dbms_connector_bolt_advertised__address |
| dbms.connector.bolt.enabled | NEO4J_dbms_connector_bolt_enabled |
| dbms.connector.bolt.listen_address | NEO4J_dbms_connector_bolt_listen__address |
| dbms.connector.bolt.thread_pool_keep_alive | NEO4J_dbms_connector_bolt_thread__pool__keep__alive |
| dbms.connector.bolt.thread_pool_max_size | NEO4J_dbms_connector_bolt_thread__pool__max__size |

| Neo4j format | Docker format |
| --- | --- |
| dbms.connector.bolt.thread_pool_min_size | NEO4J_dbms_connector_bolt_thread__pool__min__size |
| dbms.connector.bolt.tls_level | NEO4J_dbms_connector_bolt_tls__level |
| dbms.connector.bolt.unsupported_thread_pool_shutdown_wait_time | NEO4J_dbms_connector_bolt_unsupported__thread__pool__shutdown__wait__time |
| dbms.connector.http.advertised_address | NEO4J_dbms_connector_http_advertised__address |
| dbms.connector.http.enabled | NEO4J_dbms_connector_http_enabled |
| dbms.connector.http.listen_address | NEO4J_dbms_connector_http_listen__address |
| dbms.connector.https.advertised_address` | NEO4J_dbms_connector_https_advertised__address |
| dbms.connector.https.enabled | NEO4J_dbms_connector_https_enabled |
| dbms.connector.https.listen_address | NEO4J_dbms_connector_https_listen__address |
| dbms.db.timezone | NEO4J_dbms_db_timezone |
| dbms.default_advertised_address | NEO4J_dbms_default__advertised__address |
| dbms.default_database | NEO4J_dbms_default__database |
| dbms.default_listen_address | NEO4J_dbms_default__listen__address |
| dbms.directories.data | NEO4J_dbms_directories_data |
| dbms.directories.dumps.root | NEO4J_dbms_directories_dumps_root |
| dbms.directories.import | NEO4J_dbms_directories_import |
| dbms.directories.lib | NEO4J_dbms_directories_lib |
| dbms.directories.logs | NEO4J_dbms_directories_logs |
| dbms.directories.metrics | NEO4J_dbms_directories_metrics |
| dbms.directories.neo4j_home | NEO4J_dbms_directories_neo4j__home |
| dbms.directories.plugins | NEO4J_dbms_directories_plugins |
| dbms.directories.run | NEO4J_dbms_directories_run |
| dbms.directories.transaction.logs.root | NEO4J_dbms_directories_transaction_logs_root |
| dbms.dynamic.setting.whitelist | NEO4J_dbms_dynamic_setting_whitelist |
| dbms.filewatcher.enabled | NEO4J_dbms_filewatcher_enabled |
| dbms.import.csv.buffer_size | NEO4J_dbms_import_csv_buffer__size |
| dbms.import.csv.legacy_quote_escaping | NEO4J_dbms_import_csv_legacy__quote__escaping |
| dbms.index.default_schema_provider | NEO4J_dbms_index_default__schema__provider |
| dbms.index.fulltext.default_analyzer | NEO4J_dbms_index_fulltext_default__analyzer |
| dbms.index.fulltext.eventually_consistent | NEO4J_dbms_index_fulltext_eventually__consistent |

| Neo4j format | Docker format |
| --- | --- |
| dbms.index.fulltext.eventually_consistent_index_update_queue_max_length | NEO4J_dbms_index_fulltext_eventually__consistent__index__update__queue__max__length |
| dbms.index_sampling.background_enabled | NEO4J_dbms_index_sampling_background__enabled |
| dbms.index_sampling.sample_size_limit | NEO4J_dbms_index_sampling_sample__size__limit |
| dbms.index_sampling.update_percentage | NEO4J_dbms_index_sampling_update__percentage |
| dbms.index_searcher_cache_size | NEO4J_dbms_index_searcher__cache__size |
| dbms.jvm.additional | NEO4J_dbms_jvm_additional |
| dbms.lock.acquisition.timeout | NEO4J_dbms_lock_acquisition_timeout |
| dbms.logs.debug.level | NEO4J_dbms_logs_debug_level |
| dbms.logs.debug.path | NEO4J_dbms_logs_debug_path |
| dbms.logs.debug.rotation.delay | NEO4J_dbms_logs_debug_rotation_delay |
| dbms.logs.debug.rotation.keep_number | NEO4J_dbms_logs_debug_rotation_keep__number |
| dbms.logs.debug.rotation.size | NEO4J_dbms_logs_debug_rotation_size |
| dbms.logs.gc.enabled | NEO4J_dbms_logs_gc_enabled |
| dbms.logs.gc.options | NEO4J_dbms_logs_gc_options |
| dbms.logs.gc.rotation.keep_number | NEO4J_dbms_logs_gc_rotation_keep__number |
| dbms.logs.gc.rotation.size | NEO4J_dbms_logs_gc_rotation_size |
| dbms.logs.http.enabled | NEO4J_dbms_logs_http_enabled |
| dbms.logs.http.path | NEO4J_dbms_logs_http_path |
| dbms.logs.http.rotation.keep_number | NEO4J_dbms_logs_http_rotation_keep__number |
| dbms.logs.http.rotation.size | NEO4J_dbms_logs_http_rotation_size |
| dbms.logs.query.allocation_logging_enabled | NEO4J_dbms_logs_query_allocation__logging__enabled |
| dbms.logs.query.early_raw_logging_enabled | NEO4J_dbms_logs_query_early__raw__logging__enabled |
| dbms.logs.query.enabled | NEO4J_dbms_logs_query_enabled |
| dbms.logs.query.page_logging_enabled | NEO4J_dbms_logs_query_page__logging__enabled |
| dbms.logs.query.parameter_full_entities | NEO4J_dbms_logs_query_parameter__full__entities |
| dbms.logs.query.parameter_logging_enabled | NEO4J_dbms_logs_query_parameter__logging__enabled |
| dbms.logs.query.path | NEO4J_dbms_logs_query_path |
| dbms.logs.query.rotation.keep_number | NEO4J_dbms_logs_query_rotation_keep__number |
| dbms.logs.query.rotation.size | NEO4J_dbms_logs_query_rotation_size |
| dbms.logs.query.runtime_logging_enabled | NEO4J_dbms_logs_query_runtime__logging__enabled |
| dbms.logs.query.threshold | NEO4J_dbms_logs_query_threshold |

| Neo4j format | Docker format |
| --- | --- |
| dbms.logs.query.time_logging_enabled | NEO4J_dbms_logs_query_time__logging__enabled |
| dbms.logs.security.level | NEO4J_dbms_logs_security_level |
| dbms.logs.security.path | NEO4J_dbms_logs_security_path |
| dbms.logs.security.rotation.delay | NEO4J_dbms_logs_security_rotation_delay |
| dbms.logs.security.rotation.keep_number | NEO4J_dbms_logs_security_rotation_keep__number |
| dbms.logs.security.rotation.size | NEO4J_dbms_logs_security_rotation_size |
| dbms.logs.user.path | NEO4J_dbms_logs_user_path |
| dbms.logs.user.rotation.delay | NEO4J_dbms_logs_user_rotation_delay |
| dbms.logs.user.rotation.keep_number | NEO4J_dbms_logs_user_rotation_keep__number |
| dbms.logs.user.rotation.size | NEO4J_dbms_logs_user_rotation_size |
| dbms.logs.user.stdout_enabled | NEO4J_dbms_logs_user_stdout__enabled |
| dbms.max_databases | NEO4J_dbms_max__databases |
| dbms.memory.heap.initial_size | NEO4J_dbms_memory_heap_initial__size |
| dbms.memory.heap.max_size | NEO4J_dbms_memory_heap_max__size |
| dbms.memory.off_heap.block_cache_size | NEO4J_dbms_memory_off__heap_block__cache__size |
| dbms.memory.off_heap.max_cacheable_block_size | NEO4J_dbms_memory_off__heap_max__cacheable__block__size |
| dbms.memory.off_heap.max_size | NEO4J_dbms_memory_off__heap_max__size |
| dbms.memory.pagecache.direction | NEO4J_dbms_memory_pagecache_direction |
| dbms.memory.pagecache.scan.prefetchers | NEO4J_dbms_memory_pagecache_scan_prefetchers |
| dbms.memory.pagecache.size | NEO4J_dbms_memory_pagecache_size |
| dbms.memory.pagecache.swapper | NEO4J_dbms_memory_pagecache_swapper |
| dbms.memory.pagecache.warmup.enable | NEO4J_dbms_memory_pagecache_warmup_enable |
| dbms.memory.pagecache.warmup.preload | NEO4J_dbms_memory_pagecache_warmup_preload |
| dbms.memory.pagecache.warmup.preload.whitelist | NEO4J_dbms_memory_pagecache_warmup_preload_whitelist |
| dbms.memory.pagecache.warmup.profile.interval | NEO4J_dbms_memory_pagecache_warmup_profile_interval |
| dbms.memory.tracking.enable | NEO4J_dbms_memory_tracking_enable |
| dbms.memory.transaction.datababase_max_size | NEO4J_dbms_memory_transaction_datababase__max__size |
| dbms.memory.transaction.global_max_size | NEO4J_dbms_memory_transaction_global__max__size |
| dbms.memory.transaction.max_size | NEO4J_dbms_memory_transaction_max__size |
| dbms.mode | NEO4J_dbms_mode |
| dbms.netty.ssl.provider | NEO4J_dbms_netty_ssl_provider |

| Neo4j format | Docker format |
| --- | --- |
| `dbms.query_cache_size` | `NEO4J_dbms_query__cache__size` |
| `dbms.read_only` | `NEO4J_dbms_read__only` |
| `dbms.reconciler.max_backoff` | `NEO4J_dbms_reconciler_max__backoff` |
| `dbms.reconciler.max_parallelism` | `NEO4J_dbms_reconciler_max__parallelism` |
| `dbms.reconciler.may_retry` | `NEO4J_dbms_reconciler_may__retry` |
| `dbms.reconciler.min_backoff` | `NEO4J_dbms_reconciler_min__backoff` |
| `dbms.record_format` | `NEO4J_dbms_record_format` |
| `dbms.recovery.fail_on_missing_files` | `NEO4J_dbms_recovery_fail__on__missing__files` |
| `dbms.relationship_grouping_threshold` | `NEO4J_dbms_relationship__grouping__threshold` |
| `dbms.rest.transaction.idle_timeout` | `NEO4J_dbms_rest_transaction_idle__timeout` |
| `dbms.routing.advertised_address` | `NEO4J_dbms_routing_advertised__address` |
| `dbms.routing.driver.api` | `NEO4J_dbms_routing_driver_api` |
| `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` |
| `dbms.routing.listen_address` | `NEO4J_dbms_routing_listen__address` |
| `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` |

| Neo4j format | Docker format |
| --- | --- |
| dbms.security.causal_clustering_status_auth_enabled | NEO4J_dbms_security_causal__clustering__status__auth__enabled |
| dbms.security.http_access_control_allow_origin | NEO4J_dbms_security_http__access__control__allow_origin |
| dbms.security.http_auth_whitelist | NEO4J_dbms_security_http__auth__whitelist |
| dbms.security.http_strict_transport_security | NEO4J_dbms_security_http__strict__transport__security |
| 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.use_samaccountname | NEO4J_dbms_security_ldap_authentication_use__samaccountname |
| dbms.security.ldap.authentication.user_dn_template | NEO4J_dbms_security_ldap_authentication_user__dn__template |
| dbms.security.ldap.authorization.group_membership_attributes | NEO4J_dbms_security_ldap_authorization_group__membership__attributes |
| dbms.security.ldap.authorization.group_to_role_mapping | NEO4J_dbms_security_ldap_authorization_group__to__role__mapping |
| 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.procedures.default_allowed | NEO4J_dbms_security_procedures_default__allowed |
| dbms.security.procedures.roles | NEO4J_dbms_security_procedures_roles |
| dbms.security.procedures.unrestricted | NEO4J_dbms_security_procedures_unrestricted |
| dbms.security.procedures.whitelist | NEO4J_dbms_security_procedures_whitelist |

| Neo4j format | Docker format |
| --- | --- |
| `dbms.shutdown_transaction_end_timeout` | `NEO4J_dbms_shutdown__transaction__end__timeout` |
| `dbms.threads.worker_count` | `NEO4J_dbms_threads_worker__count` |
| `dbms.track_query_allocation` | `NEO4J_dbms_track__query__allocation` |
| `dbms.track_query_cpu_time` | `NEO4J_dbms_track__query__cpu__time` |
| `dbms.transaction.bookmark_ready_timeout` | `NEO4J_dbms_transaction_bookmark__ready__timeout` |
| `dbms.transaction.concurrent.maximum` | `NEO4J_dbms_transaction_concurrent_maximum` |
| `dbms.transaction.monitor.check.interval` | `NEO4J_dbms_transaction_monitor_check_interval` |
| `dbms.transaction.sampling.percentage` | `NEO4J_dbms_transaction_sampling_percentage` |
| `dbms.transaction.timeout` | `NEO4J_dbms_transaction_timeout` |
| `dbms.transaction.tracing.level` | `NEO4J_dbms_transaction_tracing_level` |
| `dbms.tx_log.preallocate` | `NEO4J_dbms_tx__log_preallocate` |
| `dbms.tx_log.rotation.retention_policy` | `NEO4J_dbms_tx__log_rotation_retention__policy` |
| `dbms.tx_log.rotation.size` | `NEO4J_dbms_tx__log_rotation_size` |
| `dbms.tx_state.memory_allocation` | `NEO4J_dbms_tx__state_memory__allocation` |
| `dbms.unmanaged_extension_classes` | `NEO4J_dbms_unmanaged__extension__classes` |
| `dbms.upgrade_max_processors` | `NEO4J_dbms_upgrade__max__processors` |
| `dbms.windows_service_name` | `NEO4J_dbms_windows__service__name` |
| `fabric.database.name` | `NEO4J_fabric_database_name` |
| `fabric.driver.api` | `NEO4J_fabric_driver_api` |
| `fabric.driver.connection.connect_timeout` | `NEO4J_fabric_driver_connection_connect__timeout` |
| `fabric.driver.connection.max_lifetime` | `NEO4J_fabric_driver_connection_max__lifetime` |
| `fabric.driver.connection.pool.acquisition_timeout` | `NEO4J_fabric_driver_connection_pool_acquisition__timeout` |
| `fabric.driver.connection.pool.idle_test` | `NEO4J_fabric_driver_connection_pool_idle__test` |
| `fabric.driver.connection.pool.max_size` | `NEO4J_fabric_driver_connection_pool_max__size` |
| `fabric.driver.logging.level` | `NEO4J_fabric_driver_logging_level` |
| `fabric.routing.servers` | `NEO4J_fabric_routing_servers` |
| `fabric.routing.ttl` | `NEO4J_fabric_routing_ttl` |
| `fabric.stream.buffer.low_watermark` | `NEO4J_fabric_stream_buffer_low__watermark` |
| `fabric.stream.buffer.size` | `NEO4J_fabric_stream_buffer_size` |
| `fabric.stream.concurrency` | `NEO4J_fabric_stream_concurrency` |
| `fabric.graph.<id>.uri` | `NEO4J_fabric_graph_<id>_uri` |

| Neo4j format | Docker format |
| --- | --- |
| `fabric.graph.<id>.database` | `NEO4J_fabric_graph_<id>_database` |
| `fabric.graph.<id>.name` | `NEO4J_fabric_graph_<id>_name` |
| `metrics.bolt.messages.enabled` | `NEO4J_metrics_bolt_messages_enabled` |
| `metrics.csv.enabled` | `NEO4J_metrics_csv_enabled` |
| `metrics.csv.interval` | `NEO4J_metrics_csv_interval` |
| `metrics.csv.rotation.keep_number` | `NEO4J_metrics_csv_rotation_keep__number` |
| `metrics.csv.rotation.size` | `NEO4J_metrics_csv_rotation_size` |
| `metrics.cypher.replanning.enabled` | `NEO4J_metrics_cypher_replanning_enabled` |
| `metrics.enabled` | `NEO4J_metrics_enabled` |
| `metrics.graphite.enabled` | `NEO4J_metrics_graphite_enabled` |
| `metrics.graphite.interval` | `NEO4J_metrics_graphite_interval` |
| `metrics.graphite.server` | `NEO4J_metrics_graphite_server` |
| `metrics.jmx.enabled` | `NEO4J_metrics_jmx_enabled` |
| `metrics.jvm.buffers.enabled` | `NEO4J_metrics_jvm_buffers_enabled` |
| `metrics.jvm.file.descriptors.enabled` | `NEO4J_metrics_jvm_file_descriptors_enabled` |
| `metrics.jvm.gc.enabled` | `NEO4J_metrics_jvm_gc_enabled` |
| `metrics.jvm.heap.enabled` | `NEO4J_metrics_jvm_heap_enabled` |
| `metrics.jvm.memory.enabled` | `NEO4J_metrics_jvm_memory_enabled` |
| `metrics.jvm.pause_time.enabled` | `NEO4J_metrics_jvm_pause__time_enabled` |
| `metrics.jvm.threads.enabled` | `NEO4J_metrics_jvm_threads_enabled` |
| `metrics.neo4j.causal_clustering.enabled` | `NEO4J_metrics_neo4j_causal__clustering_enabled` |
| `metrics.neo4j.checkpointing.enabled` | `NEO4J_metrics_neo4j_checkpointing_enabled` |
| `metrics.neo4j.counts.enabled` | `NEO4J_metrics_neo4j_counts_enabled` |
| `metrics.neo4j.data.counts.enabled` | `NEO4J_metrics_neo4j_data_counts_enabled` |
| `metrics.neo4j.database_operation_count.enabled` | `NEO4J_metrics_neo4j_database__operation__count_enabled` |
| `metrics.neo4j.logs.enabled` | `NEO4J_metrics_neo4j_logs_enabled` |
| `metrics.neo4j.pagecache.enabled` | `NEO4J_metrics_neo4j_pagecache_enabled` |
| `metrics.neo4j.pools.enabled` | `NEO4J_metrics_neo4j_pools_enabled` |
| `metrics.neo4j.server.enabled` | `NEO4J_metrics_neo4j_server_enabled` |
| `metrics.neo4j.size.enabled` | `NEO4J_metrics_neo4j_size_enabled` |
| `metrics.neo4j.tx.enabled` | `NEO4J_metrics_neo4j_tx_enabled` |
| `metrics.prefix` | `NEO4J_metrics_prefix` |

| Neo4j format | Docker format |
| --- | --- |
| `metrics.prometheus.enabled` | `NEO4J_metrics_prometheus_enabled` |
| `metrics.prometheus.endpoint` | `NEO4J_metrics_prometheus_endpoint` |

[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.

# Chapter 6. Kubernetes



**Note:**

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.
- Configure a Neo4j deployment — Configure a Neo4j deployment using a customized *values.yaml* file.
- Persistent volumes — Use persistent volumes with the Neo4j Helm charts and what types Neo4j supports.
- Access a Neo4j DBMS — Access a Neo4j DBMS running on Kubernetes.
- Access a Neo4j cluster — Access a Neo4j cluster running on Kubernetes.
- Accessing Neo4j using Kubernetes Ingress — Access Neo4j using Kubernetes Ingress via Reverse proxy Helm chart.
- Import 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
  - Dump and load databases (offline)
  - Back up and restore a single database (online)
  - Upgrade Neo4j Community to Enterprise edition
  - 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
- Troubleshooting — Diagnose and troubleshoot a Neo4j deployment running on Kubernetes.

## 6.1. Introduction

Neo4j supports both a standalone and a cluster deployment of Neo4j on Kubernetes using 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*.

### 6.1.1. The Neo4j Helm chart repository

The Neo4j Helm chart repository contains helm charts for Neo4j standalone server (*neo4j/neo4j-standalone*), for both core (*neo4j/neo4j-core*) and read replica (neo4j/*neo4j-read-replica*) components of cluster installations, 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 Neo4j Helm charts is licensed under **Apache License 2.0.**

### 6.1.2. Using the Neo4j Helm chart repository

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

The users then run `helm install` selecting the chart to install and passing in the *values.yaml* file to customize the behavior. The Helm charts create 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 charts and the Kubernetes and Cloud resources they instantiate when installed, see Neo4j Helm charts for standalone server deployment and Neo4j Helm charts for cluster deployments.

## 6.2. Configure the Neo4j Helm chart repository

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

### 6.2.1. Prerequisites

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

### 6.2.2. 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
```

### 6.2.3. Check for the available Neo4j Helm charts

```
helm search repo neo4j/ --versions | grep 4.4.48
```

The output should be similar to the following:

```
neo4j/neo4j-admin                 4.4.48  4.4.48  Neo4j is the world's leading graph database
neo4j/neo4j-cluster-core          4.4.48  4.4.48  Neo4j is the world's leading graph database
neo4j/neo4j-headless-service      4.4.48  -       Neo4j is the world's leading graph database
neo4j/neo4j-cluster-loadbalancer  4.4.48  -       Neo4j is the world's leading graph database
neo4j/neo4j-cluster-read-replica  4.4.48  4.4.48  Neo4j is the world's leading graph database
neo4j/neo4j-docker-desktop-pv     4.4.48  -       Sets up persistent disks suitable for simple de...
neo4j/neo4j-gcloud-pv             4.4.48  -       Sets up persistent disks suitable for simple de...
neo4j/neo4j-reverse-proxy         4.4.48  4.4.48  Sets up an http server and a reverse proxy for ...
neo4j/neo4j-standalone            4.4.48  4.4.48  Neo4j is the world's leading graph database
```

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-gcloud-pv* can be used as an alternative way of creating persistent volumes in those environments.

## 6.3. Quickstart: Deploy a standalone instance

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

- Neo4j Helm charts for standalone server deployments — A schematic representation of how to use the Neo4j Helm charts 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-standalone` 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.

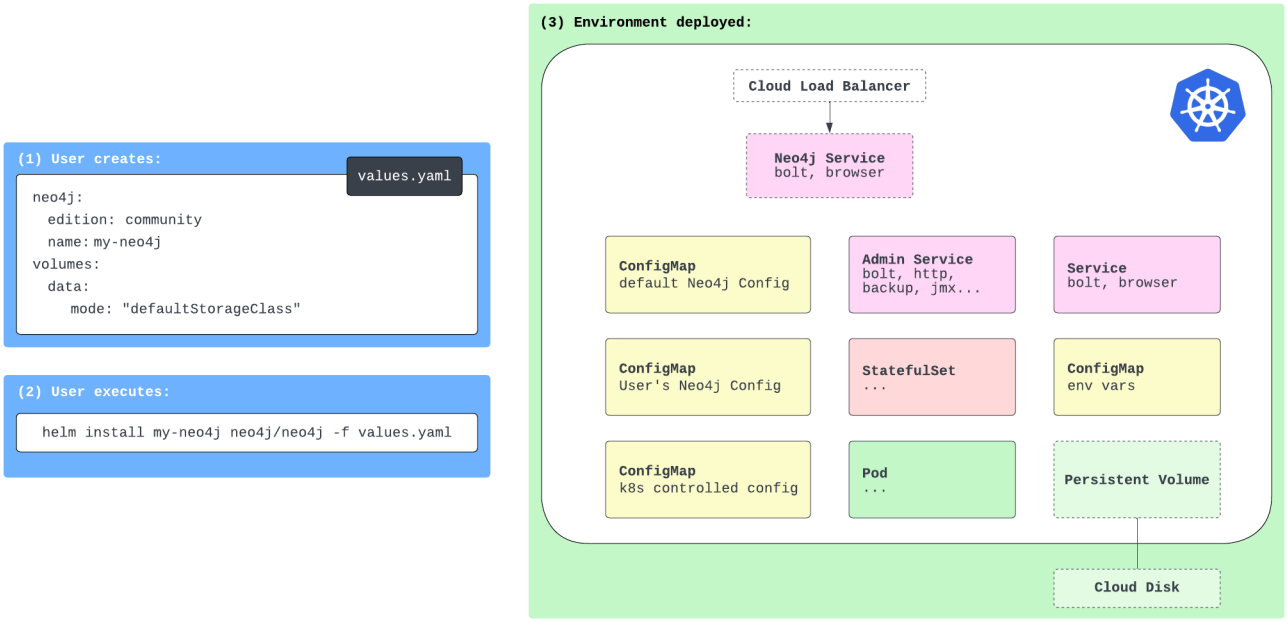
### 6.3.1. Neo4j Helm charts for standalone server deployments

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-standalone -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*




## 6.3.2. 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. The Neo4j Community Edition (default for the standalone chart) does not require a license. For more information, see link:https://neo4j.com/licensing/ or write to licensing@neo4j.com.
- Install the Kubernetes client command-line tool `kubectl` (link: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) (link: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" --release-channel "stable"
```

> **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) (link:https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html). Make sure you complete the AWS configuration step (link:https://docs.aws.amazon.com/cli/latest/userguide/getting-started-quickstart.html).
2. Install the `eksctl` command-line interface (CLI) (link: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) (link:https://docs.microsoft.com/en-us/cli/azure/install-azure-cli).
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.

## 6.3.3. 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.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: 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
```

For more information, see Configure resource allocation.

`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"`.

`volumes.data`

The `volumes.data` parameter maps the `data` volume mount of your Neo4j to the persistent volume 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:
  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
```

```yaml
neo4j:
  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:
  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:
  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
```

## 6.3.4. 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 derive from this *release name*.
Release name must consist of lower case alphanumeric characters, `-` or `.`, and must start and end with an alphanumeric character. This guide assumes the release name is `my-neo4j-release`.

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

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

*Example output*

```
NAME: my-neo4j-release
LAST DEPLOYED: Wed Jul 28 13:16:39 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Thank you for installing neo4j-standalone.

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

To view the progress of the rollout try:

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


The neo4j user's password has been set to "bO7YDTVOgs7CS1".

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

  $ kubectl run --rm -it --image "neo4j:4.4.48" cypher-shell \
     -- cypher-shell -a "neo4j://my-neo4j-release.default.svc.cluster.local:7687" -u neo4j -p
"bO7YDTVOgs7CS1"

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.

## 6.3.5. 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
```

```
2021-07-28 12:45:50.267+0000 INFO  Command expansion is explicitly enabled for configuration
2021-07-28 12:45:50.280+0000 INFO  Starting...
2021-07-28 12:45:55.680+0000 INFO  ======== Neo4j 4.4.48 ========
2021-07-28 12:46:00.006+0000 INFO  Bolt enabled on [0:0:0:0:0:0:0:0%0]:7687.
2021-07-28 12:46:02.476+0000 INFO  Remote interface available at _\http://localhost:7474_/
2021-07-28 12:46:02.478+0000 INFO  Started.
```

4. Check that the services look OK:

```
kubectl get services
```

```
NAME                     TYPE           CLUSTER-IP      EXTERNAL-IP    PORT(S)
AGE
kubernetes               ClusterIP      10.112.0.1      <none>         443/TCP
28h
my-neo4j-release         ClusterIP      10.112.10.159   <none>         7687/TCP,7474/TCP
41m
my-neo4j-release-admin   ClusterIP      10.112.4.73     <none>
6362/TCP,7687/TCP,7474/TCP          41m
my-neo4j-release-neo4j   LoadBalancer   10.112.6.75     34.140.48.23
7474:31420/TCP,7687:31650/TCP   41m
```

5. In a web browser, open the Neo4j Browser at *http://<EXTERNAL-IP>:7474/browser*.
6. 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    my-neo4j-release -pv   10Gi       RWO            manual
```

```
8m36s
```

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 4.4.48 ========
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
2d8h
my-neo4j-release-admin   ClusterIP      10.99.11.122     <none>
6362/TCP,7687/TCP,7474/TCP            2d8h
my-neo4j-release-neo4j   LoadBalancer   10.110.138.165   localhost
7474:31237/TCP,7687:32169/TCP   2d3h
```

6. Use port forwarding to get access to the browser:

```
kubectl port-forward svc/my-neo4j-release tcp-bolt tcp-http tcp-https
```

7. In a web browser, open the Neo4j Browser at *http://localhost:7474*.
8. 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.

## 6.3.6. Uninstall Neo4j and clean up the created resources

### Uninstall 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>
```

1. Check the name of the `PersistentVolumeClaim` (`pvc`):

```
kubectl get pvc
```

```
NAME                        STATUS   VOLUME                CAPACITY   ACCESS MODES
STORAGECLASS   AGE
data-my-neo4j-release-0   Bound    my-neo4j-release-pv   1Ti        RWO            manual
43h
```



**Note:**

If you re-create Neo4j with the same settings, it will pick up the PVC again, and all the data is still on it. If you use manually provisioned volumes and delete the `PersistentVolumeClaim` and the `PersistentVolume` in Kubernetes, the `hostPath` directory with the Neo4j data will still exist.

When you use dynamically provisioned volumes and delete the `PersistentVolume`, the underlying data may or may not be removed, depending on the Docker Desktop version and configuration.

2. To fully remove all the data and resources, delete the `PersistentVolumeClaim` in Kubernetes. The dynamically provisioned volumes are automatically removed when the `PersistentVolumeClaim` is deleted.

## 6.4. 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 core members and one read replica.

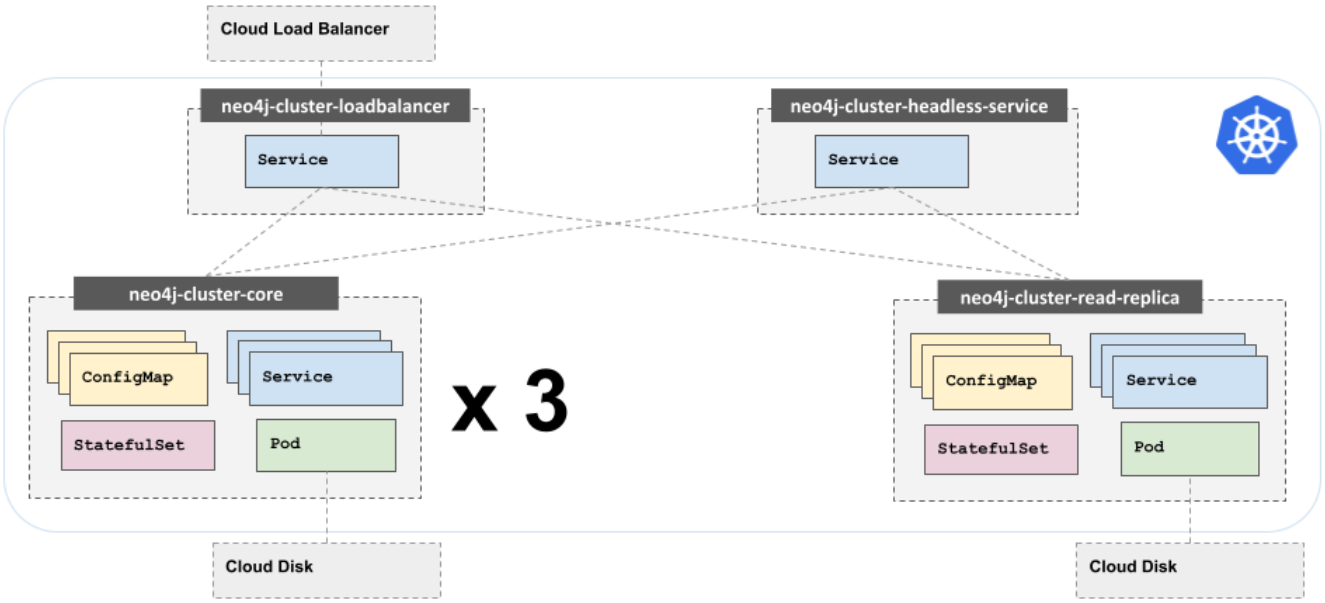
- Neo4j Helm charts for cluster deployments — A schematic representation of how to use the Neo4j Helm charts for deploying a cluster.
- Prerequisites — Set up your environment for deploying a Neo4j cluster on Kubernetes.
- Create Helm deployment values files — Create a Helm deployment *values.yaml* file for each Neo4j cluster member.
- Install Neo4j core members — Install each of your Neo4j core members using its deployment YAML file and the `neo4j/neo4j-cluster-core` Helm chart.
- Install Neo4j read replicas — Install your Neo4j read replica using its deployment YAML file and the `neo4j/neo4j-cluster-read-replica` Helm chart.
- Verify cluster formation — Verify that the Neo4j core members and read replica have formed a cluster.
- Access the Neo4j cluster from inside Kubernetes — Access the Neo4j cluster from inside Kubernetes using a specific core 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.

## 6.4.1. Neo4j Helm charts for cluster deployments

Neo4j Helm charts for cluster deployments support more configuration options than for a standalone server. To offer the flexibility required to cluster users, the Helm charts for clusters take a modular approach: the typical cluster deployment requires several Helm charts installation. For example, a cluster with three core nodes requires installing the *neo4j-cluster-core* 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 cores and one read replica. 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. Each component is installed using its own Helm chart.

## 6.4.2. 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 for Neo4j Enterprise Edition. For more information, see https://neo4j.com/licensing/ or write to licensing@neo4j.com.
- 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.
```

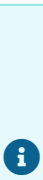
1. Install the `aws` command-line interface (CLI) (https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html). Make sure you complete the AWS configuration step

(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://docs.microsoft.com/en-us/cli/azure/install-azure-cli).
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
```

## 6.4.3. 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`

All Neo4j cluster members (cores or read-replicas) are linked together by the value of the parameter `neo4j.name`. When installed via the Neo4j Helm charts, they will join the cluster identified by `neo4j.name`.
`neo4j.name` is different from the Helm *release name*, which is used to reference a specific cluster member elsewhere in Kubernetes.
All other names and labels of K8s objects created by the helm charts derive from both `neo4j.name` and *release name*. If no name is specified, the cluster gets the default name `neo4j-cluster`.

`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-cluster-core/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 Configure resource allocation.

`neo4j.password`

The password for the `neo4j` user. The same password must be set on 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 cluster Helm charts install Neo4j Enterprise Edition since clusters are not available in Community Edition. This means that you do not have to explicitly set the configuration parameter `edition:` to `"enterprise"`, as it would be the case if you want to install a standalone server of Neo4j Enterprise Edition. However, you must acknowledge license compliance by setting `neo4j.acceptLicenseAgreement` to `"yes"`. For more information on how to obtain a valid license for Neo4j Enterprise Edition, see https://neo4j.com/licensing/ or write to licensing@neo4j.com.

`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 cluster member

Select the tab as per your Kubernetes environment and using the provided example, create a YAML file for each of your cluster members (both cores and the read replica) with all the configuration settings.

This guide assumes that the YAML files are named *core-1.values.yaml*, *core-2.values.yaml*, *core-3.values.yaml*, and *rr-1.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.

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

```
neo4j:
  name: "my-cluster"
  resources:
    cpu: "0.5"
    memory: "2Gi"
  password: "my-password"
  acceptLicenseAgreement: "yes"

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

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

## 6.4.4. Install Neo4j cluster cores

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 lower case alphanumeric characters, `-` or `.`, and must start and end with an alphanumeric character.



Note:

The following example installations use `core-1`, `core-2`, `core-3` as release names for the core members.

1. Install each core member individually using the deployment *core-<num>.values.yaml* file created in Create Helm deployment values files and the *neo4j/neo4j-cluster-core* Helm chart.

   a. Install `core-1`.

   ```
   helm install core-1 neo4j/neo4j-cluster-core -f core-1.values.yaml
   ```

   *Example output*

   ```
   NAME: core-1
   LAST DEPLOYED: Fri Nov  5 14:32:10 2021
   NAMESPACE: default
   STATUS: deployed
   REVISION: 1
   TEST SUITE: None
   NOTES:
   Thank you for installing neo4j-cluster-core.

   Your release "core-1" has been installed .

   The neo4j user's password has been set to "my-password".

   This release creates a single Neo4j Core instance. It  will not become ready until it is able to
   form a working Neo4j cluster by joining other Neo4j Core instances. To create a working cluster
   requires at least 3 Core instances.

   Once you have a working Neo4j cluster you must install at least one Neo4j Service before you can
   connect applications to Neo4j. Available Neo4j services are:
     neo4j-headless-service - for connecting applications running inside Kubernetes to Neo4j
     neo4j-loadbalancer - for connecting applications running outside Kubernetes to Neo4j

   Graphs are everywhere!
   ```

   b. Install `core-2`.

   ```
   helm install core-2 neo4j/neo4j-cluster-core -f core-2.values.yaml
   ```

   *Example output*

   ```
   NAME: core-2
   LAST DEPLOYED: Fri Nov  5 14:36:14 2021
   NAMESPACE: default
   STATUS: deployed
   REVISION: 1
   TEST SUITE: None
   NOTES:
   Thank you for installing neo4j-cluster-core.

   Your release "core-2" has been installed .

   The neo4j user's password has been set to "my-password".

   This release creates a single Neo4j Core instance. It  will not become ready until it is able to
   form a working Neo4j cluster by joining other Neo4j Core instances. To create a working cluster
   requires at least 3 Core instances.

   Once you have a working Neo4j cluster you must install at least one Neo4j Service before you can
   connect applications to Neo4j. Available Neo4j services are:
     neo4j-headless-service - for connecting applications running inside Kubernetes to Neo4j
   ```

```
  neo4j-loadbalancer - for connecting applications running outside Kubernetes to Neo4j

Graphs are everywhere!
```

c. Install `core-3`.

```
helm install core-3 neo4j/neo4j-cluster-core -f core-3.values.yaml
```

*Example output*

```
NAME: core-3
LAST DEPLOYED: Fri Nov  5 14:36:32 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Thank you for installing neo4j-cluster-core.

Your release "core-3" has been installed .

The neo4j user's password has been set to "my-password".

This release creates a single Neo4j Core instance. It  will not become ready until it is able to
form a working Neo4j cluster by joining other Neo4j Core instances. To create a working cluster
requires at least 3 Core instances.

Once you have a working Neo4j cluster you must install at least one Neo4j Service before you can
connect applications to Neo4j. Available Neo4j services are:
  neo4j-headless-service - for connecting applications running inside Kubernetes to Neo4j
  neo4j-loadbalancer - for connecting applications running outside Kubernetes to Neo4j

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 core members have formed a cluster. See the next section Verify the Neo4j cluster formation.

## 6.4.5. Verify the Neo4j cluster formation

You check the pods and services to verify that the core members 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
core-1-0   1/1     Running   0          4m51s
core-2-0   1/1     Running   0          4m25s
core-3-0   1/1     Running   0          4m7s
```

2. You can also check the logs of the pods, for example:

```
kubectl exec core-2-0 -- tail /logs/neo4j.log
```

```
2021-11-05 16:07:13.798+0000 INFO  Connected to core-2-
internals.default.svc.cluster.local/10.48.1.4:7000 [raft version:4.0]
2021-11-05 16:07:21.147+0000 INFO  Database 'neo4j' is waiting for a total of 3 core members...
2021-11-05 16:07:31.245+0000 INFO  This instance bootstrapped the 'neo4j' database.
2021-11-05 16:07:31.657+0000 INFO  Sending metrics to CSV file at /metrics
2021-11-05 16:07:31.948+0000 INFO  Bolt enabled on 0.0.0.0:7687.
2021-11-05 16:07:31.954+0000 INFO  Bolt (Routing) enabled on 0.0.0.0:7688.
2021-11-05 16:07:33.852+0000 INFO  Connected to core-3-
internals.default.svc.cluster.local/10.48.2.5:7000 [raft version:4.0]
2021-11-05 16:07:33.858+0000 INFO  Connected to core-2-
internals.default.svc.cluster.local/10.48.1.4:7000 [raft version:4.0]
2021-11-05 16:07:37.956+0000 INFO  Remote interface available at _\http://localhost:7474_/
2021-11-05 16:07:37.956+0000 INFO  Started.
```

3. Check that the services look good:

```
kubectl get services
```

```
NAME            TYPE          CLUSTER-IP      EXTERNAL-IP    PORT(S)
AGE
core-1             ClusterIP      10.112.0.8     <none>         7687/TCP,7474/TCP
22h
core-1-admin       ClusterIP      10.112.6.162   <none>         6362/TCP,7687/TCP,7474/TCP
22h
core-1-internals   ClusterIP      None           <none>
6362/TCP,7687/TCP,7474/TCP,7688/TCP,5000/TCP,7000/TCP,6000/TCP   22h
core-2             ClusterIP      10.112.8.77    <none>         7687/TCP,7474/TCP
22h
core-2-admin       ClusterIP      10.112.9.168   <none>         6362/TCP,7687/TCP,7474/TCP
22h
core-2-internals   ClusterIP      None           <none>
6362/TCP,7687/TCP,7474/TCP,7688/TCP,5000/TCP,7000/TCP,6000/TCP   22h
core-3             ClusterIP      10.112.8.68    <none>         7687/TCP,7474/TCP
22h
core-3-admin       ClusterIP      10.112.15.3    <none>         6362/TCP,7687/TCP,7474/TCP
22h
core-3-internals   ClusterIP      None           <none>
6362/TCP,7687/TCP,7474/TCP,7688/TCP,5000/TCP,7000/TCP,6000/TCP   22h
kubernetes         ClusterIP      10.112.0.1     <none>         443/TCP
134d
```

For more information about the Neo4j services, see Access a Neo4j cluster.

## 6.4.6. Install the Neo4j read replicas

Before you can install the read replica, you need to verify that the core members have formed a cluster. For more information, see Verify the Neo4j cluster formation.



Note:

Read Replicas can only be installed after a cluster of at least three cores is formed. Otherwise, the helm install command throws the following error:

```
Error: template: neo4j-cluster-read-replica/templates/neo4j-auth.yaml:5:8: executing
"neo4j-cluster-read-replica/templates/neo4j-auth.yaml" at <include
"neo4j.checkIfClusterIsPresent" .>: error calling include: template: neo4j-cluster-read-
replica/templates/_helpers.tpl:84:11: executing "neo4j.checkIfClusterIsPresent" at <fail
"Cannot install Read Replica until a cluster of 3 or more cores is formed">: error calling
fail: Cannot install Read Replica until a cluster of 3 or more cores is formed
```

1. Install the Neo4j read replica using `rr-1` as a release name, the deployment *rr-1.values.yaml* file created in Create value.yaml files, and the *neo4j/neo4j-cluster-read-replica* Helm chart:

   ```
   helm install rr-1 neo4j/neo4j-cluster-read-replica -f rr-1.values.yaml
   ```

   *Example output*

   ```
   NAME: rr-1
   LAST DEPLOYED: Fri Nov  5 16:15:13 2021
   NAMESPACE: default
   STATUS: deployed
   REVISION: 1
   TEST SUITE: None
   NOTES:
   Thank you for installing neo4j-cluster-read-replica.

   Your release "rr-1" has been installed .

   The neo4j user's password has been set to "my-password".

   This release creates a single Neo4j Read Replica instance. It  will not become ready until it is able
   to join a working Neo4j cluster.

   Graphs are everywhere!
   ```

2. Verify that the read replica has joined the cluster. See the next section Verify the read replica has joined the cluster.

## 6.4.7. Verify the read replica has joined the cluster

You check the pods and services to verify that the read replica has joined the cluster, which might take a few minutes. In the end, you should have a cluster with three cores and one read replica:

1. Check that the pods are `READY`:

   ```
   kubectl get pods
   ```

   ```
   NAME       READY   STATUS    RESTARTS   AGE
   core-1-0   1/1     Running   0          10m
   core-2-0   1/1     Running   0          10m
   core-3-0   1/1     Running   0          10m
   rr-1-0     1/1     Running   0          5m15s
   ```

2. Check that the services look good:

```
kubectl get services
```

```
NAME               TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)
AGE
core-1             ClusterIP      10.112.0.8     <none>         7687/TCP,7474/TCP
22h
core-1-admin       ClusterIP      10.112.6.162   <none>         6362/TCP,7687/TCP,7474/TCP
22h
core-1-internals   ClusterIP      None           <none>
6362/TCP,7687/TCP,7474/TCP,7688/TCP,5000/TCP,7000/TCP,6000/TCP   22h
core-2             ClusterIP      10.112.8.77    <none>         7687/TCP,7474/TCP
22h
core-2-admin       ClusterIP      10.112.9.168   <none>         6362/TCP,7687/TCP,7474/TCP
22h
core-2-internals   ClusterIP      None           <none>
6362/TCP,7687/TCP,7474/TCP,7688/TCP,5000/TCP,7000/TCP,6000/TCP   22h
core-3             ClusterIP      10.112.8.68    <none>         7687/TCP,7474/TCP
22h
core-3-admin       ClusterIP      10.112.15.3    <none>         6362/TCP,7687/TCP,7474/TCP
22h
core-3-internals   ClusterIP      None           <none>
6362/TCP,7687/TCP,7474/TCP,7688/TCP,5000/TCP,7000/TCP,6000/TCP   22h
kubernetes         ClusterIP      10.112.0.1     <none>         443/TCP
134d
rr-1               ClusterIP      10.112.14.65   <none>         7687/TCP,7474/TCP
22h
rr-1-admin         ClusterIP      10.112.9.86    <none>         6362/TCP,7687/TCP,7474/TCP
22h
rr-1-internals     ClusterIP      None           <none>
6362/TCP,7687/TCP,7474/TCP,7688/TCP,6000/TCP                   22h
```

For more information about the Neo4j services, see Access a Neo4j cluster.

## 6.4.8. 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 core members.

1. Run `cypher-shell` in a pod to access, for example, `core-3`:

```
kubectl run --rm -it --image "neo4j:4.4.48-enterprise" cypher-shell \
     -- cypher-shell -a "neo4j://core-3.default.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 4.4 at neo4j://core-3.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.
```

2. Run the Cypher command `SHOW DATABASES` to verify that all cluster members are online:

```
SHOW DATABASES;
```

```
+-------------------------------------------------------------------------------------------------
```

```
+-----------------------------------------------------+
| name     | aliases | access       | address                                | role          |
requestedStatus | currentStatus | error | default | home  |
+-----------------------------------------------------------------------------------------------------------------+
-----------------------------------------------------+
| "neo4j"  | []      | "read-write" | "core-1.default.svc.cluster.local:7687" | "follower"     |
"online"        | "online"      | ""    | TRUE    | TRUE  |
| "neo4j"  | []      | "read-write" | "core-3.default.svc.cluster.local:7687" | "follower"     |
"online"        | "online"      | ""    | TRUE    | TRUE  |
| "neo4j"  | []      | "read-write" | "core-2.default.svc.cluster.local:7687" | "leader"       |
"online"        | "online"      | ""    | TRUE    | TRUE  |
| "neo4j"  | []      | "read-write" | "rr-1.default.svc.cluster.local:7687"   | "read_replica" |
"online"        | "online"      | ""    | TRUE    | TRUE  |
| "system" | []      | "read-write" | "core-1.default.svc.cluster.local:7687" | "leader"       |
"online"        | "online"      | ""    | FALSE   | FALSE |
| "system" | []      | "read-write" | "core-3.default.svc.cluster.local:7687" | "follower"     |
"online"        | "online"      | ""    | FALSE   | FALSE |
| "system" | []      | "read-write" | "core-2.default.svc.cluster.local:7687" | "follower"     |
"online"        | "online"      | ""    | FALSE   | FALSE |
| "system" | []      | "read-write" | "rr-1.default.svc.cluster.local:7687"   | "read_replica" |
"online"        | "online"      | ""    | FALSE   | FALSE |
+-----------------------------------------------------------------------------------------------------------------+
-----------------------------------------------------+

8 rows
ready to start consuming query after 27 ms, results consumed after another 243 ms
```

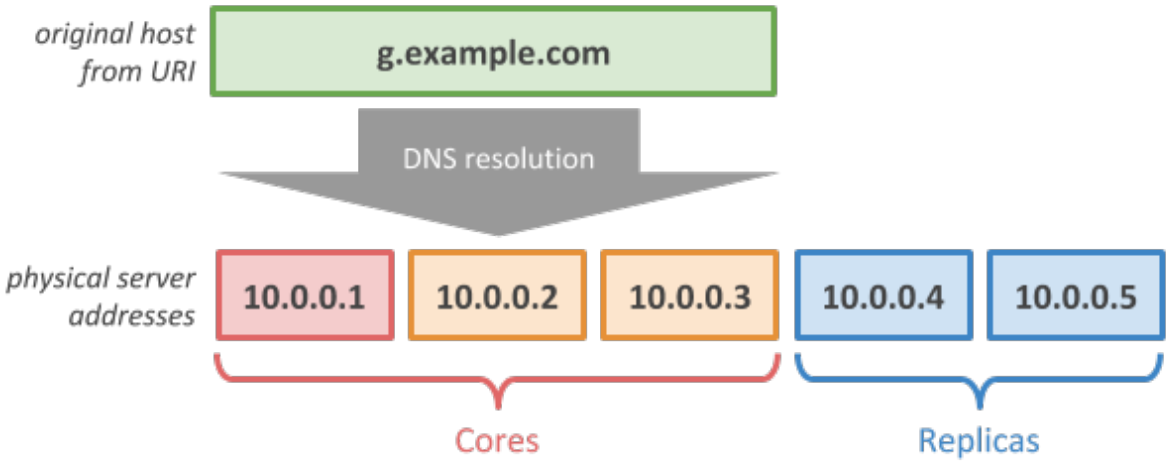
3. Exit `cypher-shell`. Exiting `cypher-shell` automatically deletes the pod created to run it.

```
: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 core 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 cores. You can use the created DNS entry to bootstrap drivers connecting to the cluster as shown in the image:



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-cluster-headless-service`.

```
helm install headless neo4j/neo4j-cluster-headless-service --set neo4j.name=my-cluster
```

```
NAME: headless
LAST DEPLOYED: Fri Nov  5 16:58:54 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Thank you for installing neo4j-cluster-headless-service.

Your release "headless" has been installed in namespace "default".

Once rollout is complete you can connect to your Neo4j cluster using "neo4j://headless-
neo4j.default.svc.cluster.local:7687". Try:

  $ kubectl run --rm -it --image "neo4j:4.4.48-enterprise" cypher-shell \
     -- cypher-shell -a "neo4j://headless-neo4j.default.svc.cluster.local:7687"

Graphs are everywhere!
```

1. Check that the `headless` service is available:

```
kubectl get services | grep head
```

```
NAME              TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)
AGE
headless-neo4j    ClusterIP      None            <none>          7474/TCP,7687/TCP
3m22s
```

2. Use `kubectl describe service` to see the service details:

```
kubectl describe service headless-neo4j
```

```
Name:              headless-neo4j
Namespace:         default
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: headless
                   meta.helm.sh/release-namespace: default
Selector:          app=my-
cluster,helm.neo4j.com/dbms.mode=CORE,helm.neo4j.com/neo4j.loadbalancer=include,helm.neo4j.com/neo4j.n
```

```
ame=my-cluster
Type:              ClusterIP
IP Families:       <none>
IP:                None
IPs:               None
Port:              http  7474/TCP
TargetPort:        7474/TCP
Endpoints:         10.108.1.19:7474,10.108.2.14:7474,10.108.4.37:7474
Port:              tcp-bolt  7687/TCP
TargetPort:        7687/TCP
Endpoints:         10.108.1.19:7687,10.108.2.14:7687,10.108.4.37:7687
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 cores 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 nodes via the headless service:

```
kubectl run --rm -it --image "neo4j:4.4.48-enterprise" cypher-shell \
     -- cypher-shell -a "neo4j://headless-neo4j.default.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 4.4 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 members are online.

```
SHOW DATABASES;
```

```
+-------------------------------------------------------------------------------------------------------
-----------------------------------------------------+
| name     | aliases | access       | address                                  | role           |
requestedStatus | currentStatus | error | default | home  |
+-------------------------------------------------------------------------------------------------------
-----------------------------------------------------+
| "neo4j"  | []      | "read-write" | "core-1.default.svc.cluster.local:7687" | "follower"     |
"online"        | "online"      | ""    | TRUE    | TRUE  |
| "neo4j"  | []      | "read-write" | "core-3.default.svc.cluster.local:7687" | "follower"     |
"online"        | "online"      | ""    | TRUE    | TRUE  |
| "neo4j"  | []      | "read-write" | "core-2.default.svc.cluster.local:7687" | "leader"       |
"online"        | "online"      | ""    | TRUE    | TRUE  |
| "neo4j"  | []      | "read-write" | "rr-1.default.svc.cluster.local:7687"   | "read_replica" |
"online"        | "online"      | ""    | TRUE    | TRUE  |
| "system" | []      | "read-write" | "core-1.default.svc.cluster.local:7687" | "leader"       |
"online"        | "online"      | ""    | FALSE   | FALSE |
| "system" | []      | "read-write" | "core-3.default.svc.cluster.local:7687" | "follower"     |
"online"        | "online"      | ""    | FALSE   | FALSE |
| "system" | []      | "read-write" | "core-2.default.svc.cluster.local:7687" | "follower"     |
"online"        | "online"      | ""    | FALSE   | FALSE |
| "system" | []      | "read-write" | "rr-1.default.svc.cluster.local:7687"   | "read_replica" |
"online"        | "online"      | ""    | FALSE   | FALSE |
+-------------------------------------------------------------------------------------------------------
-----------------------------------------------------+

8 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
```

## 6.4.9. 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

To access a Neo4j cluster from outside Kubernetes, you need to install the `neo4j-cluster-loadbalancer` service.

1. Install the load balancer service using the release name `lb`, the *neo4j/neo4j-cluster-loadbalancer* 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-loadbalancer* Helm chart, use `helm show values neo4j/neo4j-cluster-loadbalancer`.

```
helm install lb neo4j/neo4j-cluster-loadbalancer --set neo4j.name=my-cluster
```

```
NAME: lb
LAST DEPLOYED: Mon Nov 8 16:11:11 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Thank you for installing neo4j-cluster-loadbalancer.

Your release “lb” has been installed in namespace “default”.

To view the status of your Load Balancer service you can use
 $ kubectl get service lb-neo4j

Once your Load Balancer has an External-IP assigned you can connect to your Neo4j cluster using
“neo4j://<EXTERNAL-IP>:7687”. Try:

 $ cypher-shell -a “neo4j://<EXTERNAL-IP>:7687"

Graphs are everywhere!
```

2. Check that the `lb` service is available:

```
kubectl get services  | grep lb
```

```
NAME              TYPE          CLUSTER-IP      EXTERNAL-IP     PORT(S)
AGE
```

```
lb-neo4j          LoadBalancer   10.51.248.52    34.65.252.122   7474:31712/TCP,7687:31649/TCP
3m
```

3. Use `kubectl describe service` to see the service details:

```
kubectl describe service lb-neo4j
```

```
Name:                     lb-neo4j
Namespace:                default
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: lb
                          meta.helm.sh/release-namespace: default
Selector:                 app=my-
cluster,helm.neo4j.com/neo4j.loadbalancer=include,helm.neo4j.com/neo4j.name=my-cluster
Type:                     LoadBalancer
IP Families:              <none>
IP:                       10.112.6.150
IPs:                      10.112.6.150
LoadBalancer Ingress:     35.205.81.95
Port:                     http  7474/TCP
TargetPort:               7474/TCP
NodePort:                 http  31627/TCP
Endpoints:                10.108.0.3:7474,10.108.1.19:7474,10.108.2.14:7474 + 1 more...
Port:                     tcp-bolt  7687/TCP
TargetPort:               7687/TCP
NodePort:                 tcp-bolt  31988/TCP
Endpoints:                10.108.0.3:7687,10.108.1.19:7687,10.108.2.14:7687 + 1 more...
Session Affinity:         None
External Traffic Policy:  Local
HealthCheck NodePort:     31343
Events:                   <none>
```

The load-balancer service can send requests to all members of the cluster. You can see that it has four endpoints: three cores and one read-replica.

4. Run `cypher-shell` from the local machine and connect to the `LoadBalancer Ingress` address, in the example `34.65.252.122`:

```
./cypher-shell -a neo4j://34.65.252.122 -u neo4j -p my-password
```

```
If you don't see a command prompt, try pressing enter.
Connected to Neo4j using Bolt protocol version 4.4 at neo4j://34.65.252.122:7687 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.
```

5. Run the Cypher command `SHOW DATABASES` to verify that all cluster members are online:

```
SHOW DATABASES;
```

```
+-------------------------------------------------------------------------------------------------
-----------------------------------------------------+
| name     | aliases | access       | address                                | role           |
requestedStatus | currentStatus | error | default | home  |
+-------------------------------------------------------------------------------------------------
-----------------------------------------------------+
| "neo4j"  | []      | "read-write" | "core-1.default.svc.cluster.local:7687" | "follower"     |
```

```
"online"        | "online"      | ""    | TRUE    | TRUE  |
| "neo4j"  | []      | "read-write" | "core-3.default.svc.cluster.local:7687" | "follower"     |
"online"        | "online"      | ""    | TRUE    | TRUE  |
| "neo4j"  | []      | "read-write" | "core-2.default.svc.cluster.local:7687" | "leader"       |
"online"        | "online"      | ""    | TRUE    | TRUE  |
| "neo4j"  | []      | "read-write" | "rr-1.default.svc.cluster.local:7687"   | "read_replica" |
"online"        | "online"      | ""    | TRUE    | TRUE  |
| "system" | []      | "read-write" | "core-1.default.svc.cluster.local:7687" | "leader"       |
"online"        | "online"      | ""    | FALSE   | FALSE |
| "system" | []      | "read-write" | "core-3.default.svc.cluster.local:7687" | "follower"     |
"online"        | "online"      | ""    | FALSE   | FALSE |
| "system" | []      | "read-write" | "core-2.default.svc.cluster.local:7687" | "follower"     |
"online"        | "online"      | ""    | FALSE   | FALSE |
| "system" | []      | "read-write" | "rr-1.default.svc.cluster.local:7687"   | "read_replica" |
"online"        | "online"      | ""    | FALSE   | FALSE |
+-------------------------------------------------------------------------------------------------------------
-----------------------------------------------------+

8 rows
ready to start consuming query after 110 ms, results consumed after another 109 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://34.65.252.122:7474/browser*.
2. Once connected, verify that all databases are up and running using `SHOW DATABASES` in the Browser Editor:

```
SHOW DATABASES
```

```
+-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------+
| "name"   | "aliases" | "access"     | "address"                                | "role"         |
"requestedStatus" | "currentStatus" | "error" | "default" | "home" |
+-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------+
| "neo4j"  | []        | "read-write" | "core-1.default.svc.cluster.local:7687" | "follower"     |
"online"          | "online"        | ""       | true      | true   |
| "neo4j"  | []        | "read-write" | "core-3.default.svc.cluster.local:7687" | "follower"     |
"online"          | "online"        | ""       | true      | true   |
| "neo4j"  | []        | "read-write" | "core-2.default.svc.cluster.local:7687" | "leader"       |
"online"          | "online"        | ""       | true      | true   |
| "neo4j"  | []        | "read-write" | "rr-1.default.svc.cluster.local:7687"   | "read_replica" |
"online"          | "online"        | ""       | true      | true   |
| "system" | []        | "read-write" | "core-1.default.svc.cluster.local:7687" | "leader"       |
"online"          | "online"        | ""       | false     | false  |
| "system" | []        | "read-write" | "core-3.default.svc.cluster.local:7687" | "follower"     |
"online"          | "online"        | ""       | false     | false  |
| "system" | []        | "read-write" | "core-2.default.svc.cluster.local:7687" | "follower"     |
"online"          | "online"        | ""       | false     | false  |
| "system" | []        | "read-write" | "rr-1.default.svc.cluster.local:7687"   | "read_replica" |
"online"          | "online"        | ""       | false     | false  |
+-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------+
```

## 6.4.10. Uninstall Neo4j cluster and clean up resources

### Uninstall all Neo4j Helm deployments

1. Uninstall each of the cluster members (both the cores and read replica) and the services using their Helm release names:

```
helm uninstall core-1 core-2 core-3 rr-1 lb headless
```

```
release "core-1" uninstalled
release "core-2" uninstalled
release "core-3" uninstalled
release "rr-1" uninstalled
release "lb" 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>
```

## 6.5. Configure a Neo4j Helm deployment

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.

### 6.5.1. 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, such as *neo4j/neo4j-standalone*, *neo4j/neo4j-cluster-core*, *neo4j/neo4j-cluster-read-replica*, *neo4j/neo4j-cluster-headless-service*, and *neo4j/neo4j-cluster-loadbalancer*. For example:

```
helm show values neo4j/neo4j-standalone
```

```
# 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
# 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
  password: ""

  # Existing secret to use for initial database password
  passwordFromSecret: ""

  # Neo4j Edition to use (community|enterprise)
  edition: "community"
  # 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.
  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: { }
    #Set it to true when you do not want to use the subPathExpr
    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: { }
    #Set it to true when you do not want to use the subPathExpr
    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: { }
    #Set it to true when you do not want to use the subPathExpr
    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: { }
    #Set it to true when you do not want to use the subPathExpr
    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: { }
    #Set it to true when you do not want to use the subPathExpr
    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
      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 7474)
        # port: 443
      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
      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

  # 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 "headless" service for admin/ops and Neo4j cluster-internal communications
  # This service is available even if the deployment is not "ready"
  internals:
    enabled: false
    # Annotations for the internals service
    annotations: { }
    # n.b. there is no ports object for this service. Ports are autogenerated based on the neo4j
configuration


# Neo4j Configuration (yaml format)
config:
  dbms.config.strict_validation: "false"

  # 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.
  #dbms.memory.pagecache.size: "74m"

  #The number of Cypher query execution plans that are cached.
  #dbms.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.
  #dbms.memory.heap.initial_size: "317m"
  #dbms.memory.heap.max_size: "317m"

apoc_config: {}
#  apoc.trigger.enabled: "true"
#  apoc.jdbc.apoctest.url: "jdbc:foo:bar"
```

```
#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"

# containerSecurityContext 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

# 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:
  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:
  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:
  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: [ ]

# Kubernetes cluster domain suffix
clusterDomain: "cluster.local"

# Override image settings in Neo4j pod
image:
  imagePullPolicy: IfNotPresent
  # set a customImage if you want to use your own docker image
  # customImage: my-image:my-tag

  #imagePullSecrets list
  #  imagePullSecrets:
  #    - "demo"

  #imageCredentials list for which secret of type docker-registry will be created automatically using
the details provided
  # registry , username , password , email are compulsory field for an imageCredential , without any ,
helm chart will throw an error
  # imageCredential name should be part of the imagePullSecrets list or else the respective
imageCredential will be ignored and no secret creation will be done
#  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

  #Add tolerations to the Neo4j pod
  tolerations:
#    - key: "key1"
#      operator: "Equal"
#      value: "value1"
#      effect: "NoSchedule"
#    - key: "key2"
#      operator: "Equal"
#      effect: "NoSchedule"
#      value: "value2"

  #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"

  # 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.)
  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"
```

You can amend any of these settings. Passing that file during installation overrides the default Helm chart configuration of the Neo4j installation on Kubernetes and the configuration of the Neo4j database itself.

3. Create the *neo4j-values.yaml* file with your preferred configuration. For example:

```
# neo4j-values.yaml

neo4j:
  password: "my-password"
  resources:
    cpu: "2"
    memory: "5Gi"
```

```
volumes:
  data:
    mode: "defaultStorageClass"

# Neo4j configuration (yaml format)
config:
  dbms.default_database: "neo4j"
  dbms.config.strict_validation: "true"
```

4. Pass the *neo4j-values.yaml* file during installation.

```
helm install <release-name> neo4j/neo4j-standalone -f neo4j-values.yaml
```



**Tip:**

To see the values that have been set for a given release, use `helm get values <release-name>`.

*Some examples of possible K8s configurations*

- Configure (or disable completely) the Kubernetes LoadBalancer that exposes Neo4j outside the Kubernetes cluster by modifying the `externalService` 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.

*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.

## 6.5.2. Set Neo4j configuration

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:
  metrics.enabled: "true"
  metrics.namespaces.enabled: "false"
  metrics.csv.interval: "10s"
  metrics.csv.rotation.keep_number: "2"
  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 `dbms.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.

## 6.5.3. Set an initial password

You can set an initial password for accessing Neo4j in the *values.yaml* file. If no initial password is set, the Neo4j Helm chart will automatically generate one. 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.

## 6.5.4. Configure SSL

The Neo4j SSL Framework can be used with Neo4j Helm charts. You can specify SSL policy objects for `bolt`, `https`, `cluster`, `backup`, and `fabric`. 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 `dbms.connector.http.enabled: "false"` to the `config` object to disable `http`.

For more information on configuring SSL policies, see SSL Framework 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
   ```

## 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: "2023-01-03T16:04:11Z"
  managedFields:
  - apiVersion: v1
    fieldsType: FieldsV1
    fieldsV1:
      f:data:
        .: {}
        f:tls.crt: {}
        f:tls.key: {}
      f:type: {}
    manager: kubectl
    operation: Update
    time: "2023-01-03T16:04:11Z"
  name: neo4j-tls
  namespace: default
  resourceVersion: "77665"
  uid: 7feb1270-476e-4efd-afbe-57dc797768d9
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: default
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      23m
```

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 charts and the *ssl-values.yaml* file.

1. Install the core-1:

```
helm install core-1 neo4j/neo4j-cluster-core --set neo4j.acceptLicenseAgreement=yes --set
neo4j.password=my-password --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 `core-2` and `core-3`.
3. Verify that the Neo4j cluster is running:

```
kubectl get pods
```

*Example output*

```
NAME                      READY   STATUS    RESTARTS   AGE
core-1-0                  1/1     Running   0          2m
core-2-0                  1/1     Running   0          2m
core-3-0                  1/1     Running   0          2m
```

4. Connect to one of the cores and verify that the */certificates/cluster* directory contains the certificates:

```
kubectl exec -it core-1-0 -- bash
```

```
neo4j@core-1-0:~$ cd certificates/
neo4j@core-1-0:~/certificates$ ls -lst
```

*Example output*

```
total 12
4 drwxr-xr-x 2 root root 4096 Jan  3 16:07 bolt
4 drwxr-xr-x 3 root root 4096 Jan  3 16:07 cluster
4 drwxr-xr-x 3 root root 4096 Jan  3 16:07 https
```

```
neo4j@core-1-0:~/certificates$ cd cluster/
neo4j@core-1-0:~/certificates/cluster$ ls -lst
```

*Example output*

```
total 8
4 -rw-r--r-- 1 root neo4j 1159 Jan  3 16:06 public.crt
0 drwxrwsrwt 3 root neo4j  100 Jan  3 16:06 trusted
4 -rw-r--r-- 1 root neo4j 1704 Jan  3 16:06 private.key
```

```
neo4j@core-1-0:~/certificates/cluster$ cd trusted/
neo4j@core-1-0:~/certificates/cluster/trusted$ ls -lst
```

*Example output*

```
total 4
4 -rw-r--r-- 1 root neo4j 1159 Jan  3 16:06 public.crt
```

5. Exit the pod:

```
exit
```

6. Install a load balancer for the Neo4j cluster:

```
helm install lb neo4j/neo4j-cluster-loadbalancer
```

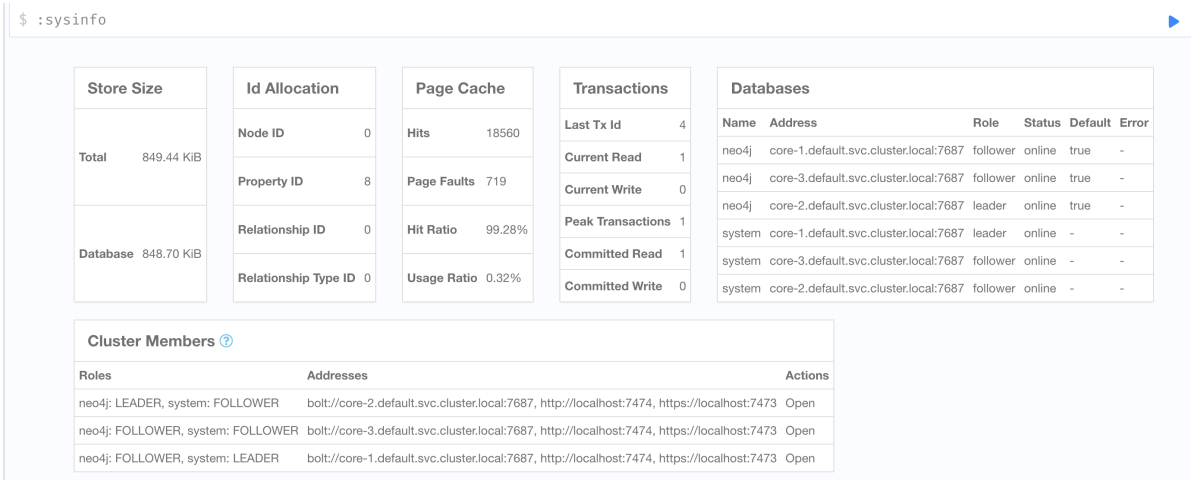
7. Verify that the load balancer is also running:

```
kubectl get services  | grep lb
```

*Example output*

```
lb-neo4j           LoadBalancer   10.0.177.22    20.62.191.227
7474:31164/TCP,7473:32555/TCP,7687:30541/TCP  36m
```

8. 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://20.62.191.227:7473/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 `CALL dbms.cluster.overview()`:




     d. Run `CALL dbms.listConfig('ssl') YIELD name, value` 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 core-1-0 -- bash
```

     b. Navigate to the *bin* directory and connect to core-1 using `cypher-shell`:

```
neo4j@core-1-0:~$ cd bin
neo4j@core-1-0:~/bin$ ./cypher-shell -u neo4j -p my-password -a neo4j+ssc://core-
1.default.svc.cluster.local:7687
```

*Example output*

```
Connected to Neo4j using Bolt protocol version 4.4 at neo4j+ssc://core-
1.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.
neo4j@neo4j>
```

c. Verify that the cluster is online by running `CALL dbms.cluster.overview()`:

```
neo4j@core-1-0:~/bin$ CALL dbms.cluster.overview();
```

*Example output*

```
+-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----------------------------------------------+
| id                                    | addresses
| databases                              | groups                                          |
+-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----------------------------------------------+
| "8edca247-a917-48d8-bfae-624509f3e515" | ["bolt://core-2.default.svc.cluster.local:7687",
"http://localhost:7474", "https://localhost:7473"] | {neo4j: "LEADER", system: "FOLLOWER"}   |
["$(bash -c 'echo pod-${POD_NAME}')", "cores"] |
| "e6ccd8ba-ad23-47b2-a417-8c0b925e1e7a" | ["bolt://core-3.default.svc.cluster.local:7687",
"http://localhost:7474", "https://localhost:7473"] | {neo4j: "FOLLOWER", system: "FOLLOWER"} |
["$(bash -c 'echo pod-${POD_NAME}')", "cores"] |
| "6c7ee59b-ad7e-4819-a29a-ab6c5838d7a2" | ["bolt://core-1.default.svc.cluster.local:7687",
"http://localhost:7474", "https://localhost:7473"] | {neo4j: "FOLLOWER", system: "LEADER"}   |
["$(bash -c 'echo pod-${POD_NAME}')", "cores"] |
+-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----------------------------------------------+

3 rows
ready to start consuming query after 11 ms, results consumed after another 1 ms
neo4j@neo4j>
```

d. Run `CALL dbms.listConfig('ssl') YIELD name, value;` to verify that the configuration is deployed as expected.

## 6.5.5. 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://login.microsoftonline.com/da501982-4ca7-
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"
  token_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: "e197354c-bd75-4524-abbc-
d44325904567=editor;fa31ce67-9e4d-4999-bf6d-25c55258d116=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 OpenId spec.

## 6.5.6. 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.

## 6.5.7. Configure resource allocation

### *CPU and memory*

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 `dbms.memory.heap.max_size` and `dbms.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:
  dbms.memory.heap.initial_size: "3G"
  dbms.memory.heap.max_size: "3G"
```

```
  dbms.memory.pagecache.size: "1G"
```

`dbms.memory.pagecache.size` and `dbms.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.

## 6.5.8. 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.

## 6.5.9. Configure a custom container image

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:4.4-enterprise"
```

## 6.5.10. Configure and install APOC core only

APOC core is shipped with Neo4j, but it is not installed in the Neo4j *plugins* directory. If APOC core is the *only* plugin that you want to add to Neo4j, it is not necessary to perform plugin installation as described in Install Plugins. 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 loading and unresticting the functions and procedures you need (for more details see APOC installation guide). For example:

```
config:
  dbms.directories.plugins: "/var/lib/neo4j/labs"
  dbms.security.procedures.unrestricted: "apoc.cypher.doIt"
  dbms.security.procedures.allowlist: "apoc.math.maxInteger,apoc.cypher.doIt"
```

2. Under `apoc_config`, configure the APOC settings 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-standalone -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()
```

5. Verify the APOC configs using the `apoc.config.list()` procedure:

```
CALL apoc.config.list() YIELD key, value WHERE key = "apoc.jdbc.neo4j.url" RETURN *;
```

## 6.5.11. Configure credentials for plugin's aliases using APOC-extended

From 4.4.23, 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"
```

## 6.5.12. Install Plugins

There are three recommended methods for adding Neo4j plugins to Neo4j Helm chart deployments. You can use:

- an automatic plugin download
- an init container
- 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 GSD 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 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:
    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.*,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 an init container

Init containers are specialized containers that run before the main containers (in this case, the Neo4j container) in a pod. You can use an init container to add plugins that have already been downloaded on an internal file server and are available to you via the internal network.

In the following example a YAML file, called *plugin_initContainer.yaml* file, configures an init container to download the plugins, in this case, `apoc`, from an internal file server to the */plugins* directory, which is backed by a persistent volume. Then, it deploys a Neo4j standalone server. When the Neo4j container starts, it automatically installs the plugins from the */plugins* directory.

1. Configure what operations the init container should run using the `initContainers` property in the *plugin_initContainer.yaml* file.

> **Note:**
>
> Some Neo4j plugins, such as Bloom and GDS Enterprise, require 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 write to licensing@neo4j.com.

```
neo4j:
  resources:
    cpu: "0.5"
    memory: "2G"

  password: "password"

  edition: "enterprise"
  acceptLicenseAgreement: "yes"


volumes:
  data:
    mode: defaultStorageClass
  plugins:
    mode: "share"
    share:
      name: "data"
```

```
# licenses:
  # mode: "share"
  # share:
  #   name: "data"
podSpec:
  initContainers:
    - name: get-plugins
      command: ["wget", "/path/to/the/downloaded/plugin-file/apoc-version-all.jar", "-O",
"/plugins/apoc.jar"]
  # - name: get-licenses
  #   command: ["wget", "/path/to/the/downloaded/plugin-file/plugin-file-version-all.jar", "-O",
"/licenses/plugin-file.license"]

config:
  dbms.directories.plugins: "/plugins"
  # dbms.directories.licenses: "/licenses"
  dbms.security.procedures.unrestricted: "apoc.*"
  dbms.config.strict_validation: "false"
  dbms.security.procedures.allowlist: "apoc.*"

apoc_config:
  apoc.trigger.enabled: "true"
  apoc.jdbc.neo4j.url: "jdbc:foo:bar"
  apoc.import.file.enabled: "true"
```

2. Deploy a Neo4j standalone server using the *plugin_initContainer.yaml* file and the *neo4j/standalone* Helm chart:

```
helm install neo4j neo4j/neo4j-standalone -f ~/path/to/plugin_initContainer.yaml
```

3. After the pod changes its status from `Init` to `Running`, verify that the mount has the *apoc.jar*:

```
k exec -it standalone-0 -- bash
```

*Example output*

```
neo4j@standalone-0:/$ cd /plugins
neo4j@standalone-0:/plugins$ ls -lst
total 21140
21140 -rw-r--r-- 1 neo4j neo4j 21645102 Nov 28 12:03 apoc.jar
```

Wait for the pod to change its state to `READY`.

4. Using the `<EXTERNAL-IP>` of the LoadBalancer service, open the Neo4j Browser at *http://<EXTERNAL-IP>:7474/browser*.
5. Use the password that you have configured in the *plugin_initContainer.yaml* file.
6. Run a query to verify that the Apoc plugin is installed and configured. For example, you can use the following query to get the value of your jdbc URL:

```
CALL apoc.config.list() YIELD key, value WHERE key = "apoc.jdbc.neo4j.url" RETURN *;
```

It should return `"jdbc:foo:bar"` as configured in the `apoc.config`.

## 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:**
>
> The Neo4j Bloom plugin (https://neo4j.com/deployment-center/#tools-tab) 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 write to licensing@neo4j.com.

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=4.4.48-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. For more details, see Configure a custom container 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 `dbms.config.strict_validation: "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 (https://neo4j.com/deployment-center/#tools-tab) 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 write to licensing@neo4j.com.

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 the restart:

kubectl exec <neo4j-pod-name> -- ls /plugins
```

## 6.6. Volume mounts and persistent volumes with the Neo4j Helm charts

Neo4j Helm chart uses volume mounts and persistent volumes to manage the storage of data and other Neo4j files.

### 6.6.1. 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 charts. For more information, see Add plugins using a plugins volume.

### 6.6.2. 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.

## 6.6.3. 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 version 4.4.21, 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` Recommended 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:
        data: "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.

## 6.6.4. 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 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: 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*

   ```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.

# 6.7. Access a Neo4j standalone server

A Neo4j DBMS is accessible via Kubernetes Services. Neo4j has a number of different interfaces for different application and operational purposes. For more details, see Neo4j ports.

## 6.7.1. Supported Kubernetes services

The Neo4j Helm chart publishes three K8s services:

- **Neo4j 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. 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.
- **External** — a LoadBalancer service for application `neo4j`/`bolt` and `http(s)` connections originating from outside the Kubernetes cluster.

*Table 13. K8s services per Neo4j interface*

| Neo4j Interface | Default Port | Neo4j Service | Admin Service | External 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 Interface | Default Port | Neo4j Service | Admin Service | External Service |
|---|---|---|---|---|
| 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 |

*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.

## 6.7.2. 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*.

### Access Neo4j using K8s label selector

Alternatively, the Neo4j service in Kubernetes can be located using Kubernetes service discovery by searching with the label selector: `helm.neo4j.com/service=neo4j,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=neo4j,helm.neo4j.com/instance=my-release
```

### Ad-hoc external access using `kubectl port-forward`

In most cases, it is possible to access the Neo4j service from a developer machine outside the Kubernetes cluster using `kubectl port-forward`. To access the Neo4j service for `http(s)` and `neo4j/bolt` from a developer machine, use the following command:

```
kubectl port-forward svc/<release-name> tcp-bolt tcp-http tcp-https
```

Neo4j is accessible via the Neo4j browser at *http://localhost:7474*.

## 6.7.3. Applications accessing Neo4j from outside Kubernetes

To access Neo4j from an application outside the Kubernetes cluster, use the IP address of the external service. The external IP(s) of the `LoadBalancer` can be found using `kubectl`:

- Using with the service name `<release-name>-external`:

  ```
  kubectl get service <release-name>-external -ocustom-columns=ip:.status.loadBalancer.ingress[].ip
  ```

- Using a label selector:

  ```
  kubectl get service -l helm.neo4j.com/service=external,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 `externalService.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 Configure 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 `externalService` object. The default values are:

```
externalService:
 annotations: { }
 loadBalancerIP: NULL

 ports:
   http:
     enabled: true
   https:
     enabled: true
   bolt:
     enabled: true
   backup:
     enabled: false
```

Disabling / enabling a port on the `externalService` 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.

## 6.7.4. 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.

*Table 14. Supported K8s resources customizations*

| Customization | *values.yaml* field | Type |
|---|---|---|
| Setting a pod securityContext for the Neo4j Pod | `securityContext` | `PodSecurityContext` |
| Adding annotations to Services | `neo4jService.annotations` | Annotations object for `ClusterIP` service. |
| | `adminService.annotations` | Annotations object for headless (DNS) service. |
| | `externalService.annotations` | Annotations object for `LoadBalancer` service. |

## 6.7.5. 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:4.4.48 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.

## 6.8. Access a Neo4j cluster

A Neo4j cluster is accessible via Kubernetes Services. Neo4j has a number of different interfaces for different application and operational purposes. For more details, see Neo4j ports.

### 6.8.1. Supported Kubernetes services

The Neo4j Helm chart publishes four K8s services:

- **Neo4j 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 causal clustering.
- **External** — a LoadBalancer service for application `neo4j`/`bolt` and `http(s)` connections originating from outside the Kubernetes cluster. This service is installed using the *neo4j/neo4j-cluster-loadbalancer* Neo4j Helm chart.

*Table 15. K8s services per Neo4j interface*

| Neo4j Interface | Default Port | Neo4j Service | Admin Service | Internal Service | External 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 | Neo4j Service | Admin Service | Internal Service | External 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 |
| Causal Cluster discovery management | `5000` | No | No | Yes | No |
| Causal Cluster transaction | `6000` | No | No | Yes | No |
| Causal Cluster RAFT | `7000` | No | No | Yes | No |
| Causal 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.

## 6.8.2. 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*.

To allow for an application running inside Kubernetes to access the 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>`.

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
```

### Ad-hoc external access using `kubectl port-forward`

In most cases, it is possible to access the Neo4j service from a developer machine outside the Kubernetes cluster using `kubectl port-forward`. To access the Neo4j service for `http(s)` and `neo4j/bolt` from a developer machine, use the following command:

```
kubectl port-forward svc/<release-name> tcp-bolt tcp-http tcp-https
```

Neo4j is accessible via the Neo4j browser at *http://localhost:7474*.

## 6.8.3. Applications accessing Neo4j from outside Kubernetes

To access a Neo4j cluster from outside Kubernetes, you need to install the `neo4j-cluster-loadbalancer` service using the *neo4j/neo4j-cluster-loadbalancer* Helm chart. For a detailed example, see Access the Neo4j cluster from outside Kubernetes.

## 6.8.4. 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.

*Table 16. Supported K8s resources customizations*

| 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.neo4j.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 | `annotations` | Annotations object for `LoadBalancer` service. |

## 6.8.5. 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:4.4.48 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.

# 6.9. 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 4.4.25. 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.

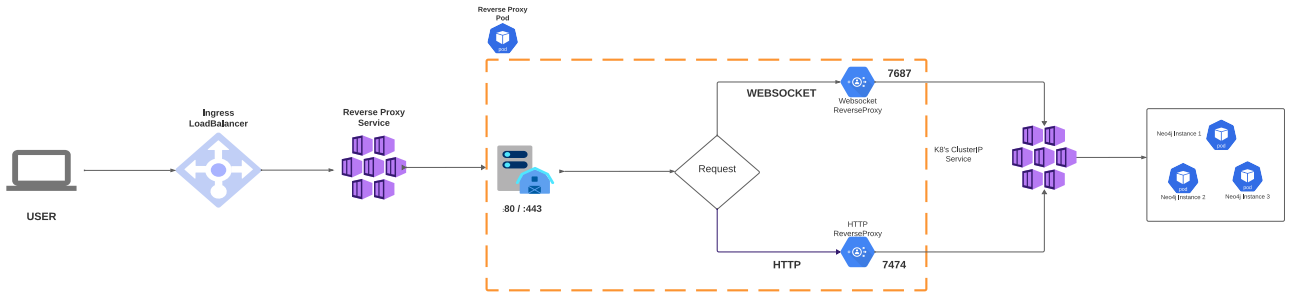



*Figure 1. Reverse proxy flow diagram*

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`.

## 6.9.1. 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:4.4.26"

  # 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"

  # 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"
    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
```

## 6.9.2. Configure the Kubernetes Ingress

Configure the *ingress.yaml* file that you will use to install the Reverse proxy Helm chart.

### Configure the *ingress.yaml* file to access Neo4j on port `:443`

The following example shows how to configure the *ingress.yaml* file to access Neo4j on port `:443`:

1. Create a Kubernetes secret containing the Ingress self-signed certificates and then create the *ingress.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.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:4.4.26
    serviceName: "standalone-admin"
    ingress:
        enabled: true
        tls:
            enabled: true
            config:
                - secretName: ingress-cert
                  hosts:
                    - localhost
```

### Configure the *ingress.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.yaml* file with the following content:

```
reverseProxy:
    image: neo4j/helm-charts-reverse-proxy:4.4.26
    serviceName: "standalone-admin"
    ingress:
        enabled: true
        tls:
            enabled: false
```

## 6.9.3. Install the Reverse proxy Helm chart

Install the Reverse proxy Helm chart using the *ingress.yaml* file that you have created:

```
helm install rp neo4j/neo4j-reverse-proxy -f /path/to/your/ingress.yaml
```

## 6.9.4. 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.

## 6.9.5. 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 scv -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=inress-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>
```

# 6.10. Import Data

## 6.10.1. Importing data into Neo4j on Kubernetes

The Neo4j Helm chart configures a volume mount at */import* as the Neo4j *import* directory, as described in 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 import` or the Cypher query `LOAD CSV`.

- The `neo4j-admin 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 a simple example, see Getting Started Guide → Import data.

> **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.

## 6.10.2. 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
```

## 6.10.3. 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.

## 6.10.4. Use `neo4j-admin import`

The simplest way to run `neo4j-admin import` is to use `kubectl exec` to run it in the Neo4j container. However, running `neo4j-admin 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 import`.

`neo4j-admin 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 import`.

## 6.10.5. 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.

# 6.11. Monitoring

## 6.11.1. 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
  ```

## 6.11.2. 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). We recommend running 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.

## 6.11.3. Metrics

If Neo4j is configured to listen for Graphite, JMX, or Prometheus connections for metrics, those services can be accessed as described in Access a Neo4j Helm release.

The Helm chart supports standard Neo4j metrics configuration settings, for example:

```
# To listen for Prometheus connections
# Neo4j configuration (yaml format)
config:
  metrics.prometheus.enabled: "true"
  metrics.prometheus.endpoint: "0.0.0.0:2004"
```

```
# To publish Graphite connections
# Neo4j configuration (yaml format)
config:
  metrics.graphite.enabled: "true"
  metrics.graphite.interval: "3s"
  metrics.graphite.server: "graphite.default.svc.cluster.local:2003"
```

```
# To write CSV metrics
# Neo4j configuration (yaml format)
config:
  metrics.csv.enabled: "true"
  metrics.csv.interval: "10s"
```

```
# To enable JMX
# Neo4j configuration (yaml format)
config:
  metrics.jmx.enabled: "true"
```

For more information and examples, see Neo4j metrics.

## 6.12. 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
- Dump and load databases (offline)
- Back up and restore a single database (online)
- Upgrade Neo4j Community to Enterprise edition
- 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

### 6.12.1. 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 store-info --all /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-standalone -f values.yaml
     ```

   - Alternatively, you can set `neo4j.offlineMaintenanceModeEnabled` to `true` as part of the `helm upgrade` command:

     ```
     helm upgrade <release-name> neo4j/neo4j-standalone --version={neo4j-version-exact} --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 store-info --all /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 check-consistency --database=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-standalone -f values.yaml
```

- Alternatively, you can run `helm upgrade` with the flag set to `false`:

```
helm upgrade <release-name> neo4j/neo4j-standalone --version=4.4.48 --set
neo4j.offlineMaintenanceModeEnabled=false
```

## 6.12.2. 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-standalone -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-standalone -f values.yaml
```

   Authentication is now enabled, and the Neo4j user password has been reset to the desired password.

## 6.12.3. Dump and load databases (offline)

You can use the `neo4j-admin dump` command to make a full backup (an archive) of an **offline** database(s) and `neo4j-admin 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 dump --expand-commands --database=system --to /backups/system.dump && neo4j-admin dump
--expand-commands --database=neo4j --to /backups/neo4j.dump
```

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 load` commands:

```
neo4j-admin load --expand-commands --database=system --from /backups/system.dump && neo4j-admin load
--expand-commands --database=neo4j --from /backups/neo4j.dump
```

> **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.

## 6.12.4. Back up 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.

## 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 version 4.4.22, the *neo4j/neo4j-admin* Helm chart also supports performing a backup of multiple databases. From 4.4.27, the *neo4j/neo4j-admin* Helm chart also supports workload identity integration for GCP, AWS, and Azure. From 4.4.28, 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. Only non-TLS/SSL endpoints are supported. 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: "4.4.22"
  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"
  checkConsistency: true
```

```
neo4j:
  image: "neo4j/helm-charts-backup"
  imageTag: "4.4.22"
  jobSchedule: "* * * * *"
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
  backoffLimit: 3

backup:
  bucketName: "my-bucket"
  databaseAdminServiceName:  "standalone-admin"
  database: "neo4j,system"
  cloudProvider: "aws"
  secretName: "awscreds"
  secretKeyName: "credentials"
  checkConsistency: true
```

```
neo4j:
  image: "neo4j/helm-charts-backup"
  imageTag: "4.4.22"
  jobSchedule: "* * * * *"
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
  backoffLimit: 3

backup:
  bucketName: "my-bucket"
  databaseAdminServiceName:  "standalone-admin"
  database: "neo4j,system"
  cloudProvider: "azure"
  secretName: "azurecreds"
  secretKeyName: "credentials"
  checkConsistency: 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:

```yaml
neo4j:
  image: "neo4j/helm-charts-backup"
  imageTag: "4.4.27"
  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: ""
  checkConsistency: true

serviceAccountName: "demo-service-account"
```

```yaml
neo4j:
  image: "neo4j/helm-charts-backup"
  imageTag: "4.4.27"
  jobSchedule: "* * * * *"
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
  backoffLimit: 3

backup:
  bucketName: "my-bucket"
  databaseAdminServiceName:  "standalone-admin"
  database: "neo4j,system"
  cloudProvider: "aws"
  secretName: ""
  secretKeyName: ""
  checkConsistency: true

serviceAccountName: "demo-service-account"
```

```yaml
neo4j:
  image: "neo4j/helm-charts-backup"
  imageTag: "4.4.27"
  jobSchedule: "* * * * *"
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
  backoffLimit: 3

backup:
  bucketName: "my-bucket"
  databaseAdminServiceName:  "standalone-admin"
  database: "neo4j,system"
  cloudProvider: "azure"
  azureStorageAccountName: "storageAccountName"
  consistencyCheck: 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 the *backup-values.yaml* file for using MinIO

*This feature is available from Neo4j 4.4.28*

MinIO is an AWS S3-compatible object storage API. You can specify the `minioEndpoint` parameter in the *backup-values.yaml* file to push your backups to your MinIO bucket. This endpoint must be a s3 API endpoint or else the backup Helm chart will fail. Only non-TLS/SSL endpoints are supported. For example:

```
neo4j:
  image: "neo4j/helm-charts-backup"
  imageTag: "4.4.28"
  jobSchedule: "* * * * *"
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
  backoffLimit: 3

backup:
  bucketName: "my-bucket"
  databaseAdminServiceName:  "standalone-admin"
  minioEndpoint: "http://demo.minio.svc.cluster.local:9000"
  database: "neo4j,system"
  cloudProvider: "aws"
  secretName: "awscreds"
  secretKeyName: "credentials"

consistencyCheck:
  enabled: true
```

## Backup parameters

To see what options are configurable on the Helm chart use `helm show values` and the Helm chart *neo4j/neo4j-admin*.
From version 4.4.22, the *neo4j/neo4j-admin* Helm chart also supports assigning your Neo4j pods to specific nodes using `nodeSelector` labels, and from Neo4j 4.4.23, 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
# You can enable this when executing helm commands with --dry-run command
```

```
disableLookups: false

neo4j:
  image: "neo4j/helm-charts-backup"
  imageTag: "4.4.27"
  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: ""

  #address details of the neo4j instance from which backup is to be done (serviceName or ip either one is
required)

  #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 minio endpoint ex: http://demo.minio.svc.cluster.local:9000
  # please ensure this endpoint is the s3 api endpoint or else the backup helm chart will fail
  # as of now it works only with non tls endpoints
  # to be used only when aws is used as cloudProvider
  minioEndpoint: ""

  #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
  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/4.4/backup-
restore/online-backup/
  pageCache: ""
  fallbackToFull: true
  includeMetadata: "all"
  parallelRecovery: false
  verbose: true
  heapSize: ""
  checkConsistency: true
  checkIndexes: true
  checkIndexStructure: true
  checkGraph: true
  prepareRestore: 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"

# default ephemeral storage of backup container
resources:
  requests:
    ephemeralStorage: "4Gi"
  limits:
    ephemeralStorage: "5Gi"

# 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 backup pod
tolerations: []
#  - key: "key1"
#    operator: "Equal"
#    value: "value1"
#    effect: "NoSchedule"
#  - key: "key2"
#    operator: "Equal"
#    value: "value2"
#    effect: "NoSchedule"
```

### Install the *neo4j-admin* Helm chart

1. Install *neo4j-admin* Helm chart using the *backup-values.yaml* file:

   ```
   helm install backup-name 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.

## 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, and 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 members. The command is automatically routed to the leader and 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 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.

1. Restore the `neo4j` database backup.



**Note:**

For Neo4j cluster deployments, restore the database backup on each cluster member.

2. Run the `neo4j-admin restore` command:

```
neo4j-admin restore --database=neo4j --from=/backups/neo4j --expand-commands
```

3. Connect to the `system` database using `cypher-shell`:

```
cypher-shell -u neo4j -p <password> -d system
```

4. Create `neo4j` database.



**Note:**

For Neo4j cluster deployments, you run the Cypher command `CREATE DATABASE name`

only on one of the cluster members.

```
CREATE DATABASE neo4j;
```

5. Open the browser at *http://<external-ip>:7474/browser/* and check that all data has been successfully restored.
6. 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).

## 6.12.5. Upgrade Neo4j Community to Enterprise edition

To upgrade from Neo4j Community to Enterprise edition, run:

```
helm upgrade <release-name> neo4j/neo4j-standalone --set neo4j.edition=enterprise --set
neo4j.acceptNeo4jLicenseAgreement=yes
```

To upgrade to the next patch 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:4.4.48
```

2. Run `helm upgrade` to apply the changes:

```
helm upgrade <release-name> neo4j/neo4j-standalone -f values.yaml
```

## 6.12.6. Migrate Neo4j from Labs Helm to Neo4j Helm charts

To migrate your Neo4j deployment from the Labs Helm charts to the Neo4j Helm charts, back up your standalone instance or cluster created with the Labs Helm charts and restore it in a standalone instance or

a cluster created using the Neo4j Helm charts.

Neo4j supports the following migration paths for a single instance and a cluster:

*Single instance*

- From the Labs Helm charts 3.5 or earlier to either Neo4j Helm charts 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 charts (4.3 or 4.4) to Neo4j Helm charts 4.3 or 4.4 using the steps described here.
- From the Labs Helm charts 4.3 to Neo4j Helm charts 4.3 — follow the steps described here.
- From the Labs Helm charts 4.3 to Neo4j Helm charts 4.4 — follow the steps described here.

*Cluster*

From the Labs Helm charts 4.3 or 4.4 to Neo4j Helm charts 4.4 — follow the steps described here.

### Back up a Neo4j deployment created with the Labs Helm charts

To back up your Neo4j deployment created with the Labs Helm charts, 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 charts

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 charts 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 charts, 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 back up following the steps described in Restore a single database.

## 6.12.7. 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 (`dbms.memory.heap.max_size` and `dbms.memory.pagecache.size` in particular). See Configure resource allocation 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:
  dbms.memory.heap.initial_size: "2G"
  dbms.memory.heap.max_size: "2G"
  dbms.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:
     dbms.memory.heap.initial_size: "2G"
     dbms.memory.heap.max_size: "2G"
     dbms.memory.pagecache.size: "1G"
   ```

2. Run `helm upgrade` with the modified deployment *values.yaml* file and the respective Helm chart (*neo4j/neo4j-standalone*, *neo4j/neo4j-cluster-core*, or *neo4j/neo4j-cluster-read-replica*) to apply the changes. For example:

   ```
   helm upgrade <release-name> neo4j/neo4j-standalone -f values.yaml
   ```

## Horizontal scaling

You can add a new core member or a read replica to the Neo4j cluster to scale out write or read workloads.

1. In the Kubernetes cluster, verify that you have a node that you can use for the new Neo4j cluster member.
2. Create a persistent disk for the new Neo4j cluster member to be used for its `data` volume mount. For

more information, see and Volume mounts and persistent volumes.

3. Create a Helm deployment YAML file for the new Neo4j cluster member with all the configuration settings and the disk you have created for it. For more information, see Create Helm deployment values files and Configure a Neo4j Helm deployment.
4. Install the new member using the command `helm install`, the deployment *values.yaml* file, and the respective Helm chart (*neo4j/neo4j-cluster-core* or *neo4j/neo4j-cluster-read-replica*). For example:

```
helm install rr-2 neo4j/neo4j-cluster-read-replica -f rr-2.values.yaml
```

## 6.12.8. Use custom images from private registries

Neo4j 4.4.4 introduces the support for using 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: "mypassword"
      email: "myusername@example.com"
      name: "mysecret"
```

## 6.12.9. Assign Neo4j pods to specific nodes

The Neo4j Helm charts `neo4j/neo4j` (from version 4.4.5) and `neo4j/neo4j-admin` (from version 4.4.22) 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 charts 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
```

# 6.13. Troubleshooting

## 6.13.1. 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 17. 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`.<br>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`. |

## 6.13.2. 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 which 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.

# Chapter 7. Configuration

The topics described are:

- The *neo4j.conf* file — An introduction to the primary configuration file in Neo4j.
- 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 Neo4j connectors — How to configure Neo4j connectors.
- Set initial password — How to set an initial password.
- Update dynamic settings — How to configure certain Neo4j parameters while Neo4j is running.
- Configuration settings — A complete reference of all configuration settings.
- Transaction logs — The transaction logs record all write operations in the database.

For a complete reference of Neo4j configuration settings, see Configuration settings.

## 7.1. The neo4j.conf file



**Tip:**

For a complete reference of Neo4j configuration settings, see Configuration settings.

### 7.1.1. Introduction

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 below. Many of the configuration settings are also used by the `neo4j` launcher scripts.

### 7.1.2. Syntax

- The equals sign (`=`) maps configuration setting keys to configuration values.
- Lines that start with the number sign (`#`) are handled as comments.
- Empty lines are ignored.
- Configuring a setting in *neo4j.conf* will overwrite any default values. In case a setting can define a list of values, and you wish to amend the default values with custom values, you will have to explicitly list the default values along with the new values.
- There is no order for configuration settings, and each setting in the *neo4j.conf* file must be uniquely

specified. If you have multiple configuration settings with the same key, but different values, this can lead to unpredictable behavior.

The only exception to this is `dbms.jvm.additional`. If you set more than one value for `dbms.jvm.additional`, then each setting value will add another custom JVM argument to the `java` launcher.

## 7.1.3. JVM-specific configuration settings

A Java virtual machine (JVM) is a virtual machine that enables a computer to run Java programs as well as programs written in other languages that are also compiled to 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 18. JVM-specific settings*

| Setting | Description |
| --- | --- |
| `dbms.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. |
| `dbms.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. |
| `dbms.jvm.additional` | Sets additional options for the JVM. The options are set as a string and can vary depending on JVM implementation. |



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.

## 7.1.4. List currently active settings

You can use the procedure `dbms.listConfig()` to list the currently active configuration settings and their values.

*Example 23. List currently active configuration settings*

```
CALL dbms.listConfig()
YIELD name, value
WHERE name STARTS WITH 'dbms.default'
RETURN name, value
ORDER BY name
LIMIT 3;
```

```
+-----------------------------------------------+
| name                            | value       |
+-----------------------------------------------+
| "dbms.default_advertised_address" | "localhost" |
| "dbms.default_database"           | "neo4j"     |
```

```
| "dbms.default_listen_address"     | "localhost" |
+-----------------------------------------------+
```



Tip:

For information about dynamic settings, see Update dynamic settings and Configuration settings reference.

## 7.2. 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.

### 7.2.1. 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.

## 7.2.2. Enabling

The Neo4j startup script and the `neo4j` service can expand and execute the external commands by using the argument `--expand-commands`.

```
bin/neo4j start --expand-commands
```

If the startup script does not receive the `--expand-commands` argument, commands in the configuration file are treated as invalid settings.

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* file must, at least, be readable by its owner or by the user-group to which the owner belongs.
- 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* file must, at least, be readable by the user that the Neo4j process runs as.

## 7.2.3. 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>...
```

## 7.2.4. 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.

# 7.3. File locations

## 7.3.1. Default file locations

The following table lists the default location of the Neo4j files, per type and distribution.

*Table 19. Default file locations*

| File type | Description | Linux / macOS / Docker | Windows | Debian / RPM | Neo4j Desktop [1] |
|---|---|---|---|---|---|
| Bin | The Neo4j running script and built-in tools, such as Cypher Shell and Neo4j Admin. | *<neo4j-home>/bin* | *<neo4j-home>\bin* | */usr/bin* | From the Open dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd bin`. |
| Certificates | The Neo4j TLS certificates. | *<neo4j-home>/certificates* | *<neo4j-home>\certificates* | */var/lib/neo4j/certificates* | From the Open dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd certificates`. |
| Configuration [2] | The Neo4j configuration settings, Log4j configuration settings, and the JMX access credentials. | *<neo4j-home>/conf/neo4j.conf* | *<neo4j-home>\conf\neo4j.conf* | */etc/neo4j/neo4j.conf* | From the Open dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd conf`. |

| File type | Description | Linux / macOS / Docker | Windows | Debian / RPM | Neo4j Desktop [1] |
|---|---|---|---|---|---|
| Data [3] | 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). | *<neo4j-home>/data* | *<neo4j-home>\data* | */var/lib/neo4j/data* | From the Open dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd data`. |
| Import | All CSV files that the command `LOAD CSV` uses as sources to import data in Neo4j. | *<neo4j-home>/import* | *<neo4j-home>\import* | */var/lib/neo4j/import* | From the Open dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd import`. |
| Labs [4] | Contains APOC Core. | *<neo4j-home>/labs* | *<neo4j-home>\labs* | */var/lib/neo4j/labs* | From the Open dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd labs`. |
| Lib | All Neo4j dependencies. | *<neo4j-home>/lib* | *<neo4j-home>\lib* | */usr/share/neo4j/lib* | From the Open dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd lib`. |
| Licenses | For storing license files from Neo4j. | *<neo4j-home>/licenses* | *<neo4j-home>\licenses* | */var/lib/neo4j/licenses* | From the Open dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd licences`. |

| File type | Description | Linux / macOS / Docker | Windows | Debian / RPM | Neo4j Desktop [1] |
|---|---|---|---|---|---|
| Logs | The Neo4j log files. | *<neo4j-home>/logs* [5] | *<neo4j-home>\logs* | */var/log/neo4j/* [6] | From the *Open* dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd logs`. |
| Metrics | The Neo4j built-in metrics for monitoring the Neo4j DBMS and each individual database. | *<neo4j-home>/metrics* | *<neo4j-home>\metrics* | */var/lib/neo4j/metrics* | From the *Open* dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd metrics`. |
| Plugins | Custom code that extends Neo4j, for example, user-defined procedures, functions, and security plugins. | *<neo4j-home>/plugins* | *<neo4j-home>\plugins* | */var/lib/neo4j/plugins* | From the *Open* dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd plugins`. |
| Products | The jar files of the Neo4j products: Graph Data Science Library and Bloom. The folder also contains a *README.txt* file with information on enabling them. | *<neo4j-home>/products* | *<neo4j-home>\products* | */var/lib/neo4j/products* | From the *Open* dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd products`. |
| Run | The processes IDs. | *<neo4j-home>/run* | *<neo4j-home>\run* | */var/lib/neo4j/run* | From the *Open* dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd run`. |

## 7.3.2. 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 20. 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.

```
#dbms.directories.data=data
#dbms.directories.plugins=plugins
#dbms.directories.logs=logs
#dbms.directories.lib=lib
#dbms.directories.run=run
#dbms.directories.licenses=licenses
#dbms.directories.metrics=metrics
#dbms.directories.transaction.logs.root=data/transactions
#dbms.directories.dumps.root=data/dumps
#dbms.directories.import=import
```

## 7.3.3. File permissions

The operating system user that Neo4j server runs as must have the following minimal permissions:

*Read only*

- *bin*
- *certificates*
- *conf*
- *import*
- *labs*
- *lib*
- *licenses*
- *plugins*
- *products*

*Read and write*

- *data*
- *logs*
- *metrics*
- *run*

*Execute*

- all files in *bin* and *tmp*



Note:

If *tmp* is set to `noexec`, it is recommended to set `dbms.jvm.additional=-Djava.io.tmpdir=/home/neo4j` in *conf/neo4j.conf*. Additionally, 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.

## 7.4. 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 taking your particular conditions into consideration.



Note:

When exposing network services, make sure they are always protected.

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 21. Listen address configuration settings overview*

| Name | Default port | Related configuration setting |
|---|---|---|
| Backup | `6362` | `dbms.backup.listen_address` |
| HTTP | `7474` | `dbms.connector.http.listen_address` |
| HTTPS | `7473` | `dbms.connector.https.listen_address` |
| Bolt | `7687` | `dbms.connector.bolt.listen_address` |
| Causal Cluster discovery management | `5000` | `causal_clustering.discovery_listen_address` |
| Causal Cluster transaction | `6000` | `causal_clustering.transaction_listen_address` |
| Causal Cluster RAFT | `7000` | `causal_clustering.raft_listen_address` |
| Causal Cluster routing connector | `7688` | `dbms.routing.listen_address` |
| Graphite monitoring | `2003` | `metrics.graphite.server` |
| Prometheus monitoring | `2004` | `metrics.prometheus.endpoint` |

| Name | Default port | Related configuration setting |
| --- | --- | --- |
| JMX monitoring | `3637` | `dbms.jvm.additional=-Dcom.sun.management.jmxremote.port=3637` |
| Remote debugging | `5005` | `dbms.jvm.additional=-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=*:5005` |



The configuration setting `dbms.default_listen_address` configures the default network interface to listen for incoming connections.

The advertised address configuration settings are used for routing purposes. An advertised address is composed by 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 22. Advertised address configuration settings overview*

| Name | Default port | Related configuration setting |
| --- | --- | --- |
| HTTP | `7474` | `dbms.connector.http.advertised_address` |
| HTTPS | `7473` | `dbms.connector.https.advertised_address` |
| Bolt | `7687` | `dbms.connector.bolt.advertised_address` |
| Causal Cluster discovery management | `5000` | `causal_clustering.discovery_advertised_address` |
| Causal Cluster transaction | `6000` | `causal_clustering.transaction_advertised_address` |
| Causal Cluster RAFT | `7000` | `causal_clustering.raft_advertised_address` |
| Causal Cluster routing connector | `7688` | `dbms.routing.advertised_address` |



The configuration setting `dbms.default_advertised_address` configures the default hostname/IP-address for advertised address.

### 7.4.1. Backup

Default port: `6362`

*Table 23. Backup*

| Related configuration setting | Default value | Description |
| --- | --- | --- |
| `dbms.backup.listen_address` | `127.0.0.1:6362` | Network interface and port for the backup server to listen on. |
| `dbms.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 Server configuration.

## 7.4.2. HTTP

Default port: `7474`

*Table 24. HTTP connector*

| Related configuration setting | Default value | Description |
| --- | --- | --- |
| `dbms.connector.http.listen_address` | `:7474` | Network interface and port for the HTTP connector to listen on. |
| `dbms.connector.http.advertised_address` | `:7474` | Advertised hostname/IP-address and port for the HTTP connector. |
| `dbms.connector.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 connectors.

## 7.4.3. HTTPS

Default port: `7473`

*Table 25. HTTPS connector*

| Related configuration setting | Default value | Description |
| --- | --- | --- |
| `dbms.connector.https.listen_address` | `:7473` | Network interface and port for the HTTPS connector to listen on. |
| `dbms.connector.https.advertised_address` | `:7473` | Advertised hostname/IP-address and port for the HTTPS connector. |
| `dbms.connector.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 connectors.

## 7.4.4. Bolt

Default port: `7687`

*Table 26. Bolt connector*

| Related configuration setting | Default value | Description |
|---|---|---|
| `dbms.connector.bolt.listen_address` | `:7687` | Network interface and port for the Bolt connector to listen on. |
| `dbms.connector.bolt.advertised_address` | `:7687` | Advertised hostname/IP-address and port for the Bolt connector. |
| `dbms.connector.bolt.enabled` | `true` | Enable the Bolt connector. |
| `dbms.connector.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 connectors.

## 7.4.5. Causal Cluster

By default, the operating mode of a Neo4j instance (`dbms.mode`) is set to `SINGLE`.

*Table 27. Cluster listen address*

| Name | Default port | Default value | Related configuration setting |
|---|---|---|---|
| Discovery management | `5000` | `:5000` | `causal_clustering.discovery_listen_address` |
| Transaction | `6000` | `:6000` | `causal_clustering.transaction_listen_address` |
| RAFT | `7000` | `:7000` | `causal_clustering.raft_listen_address` |
| Routing connector | `7688` | `:7688` | `dbms.routing.listen_address` |

*Table 28. Cluster advertised address*

| Name | Default port | Default value | Related configuration setting |
| --- | --- | --- | --- |
| Discovery management | `5000` | `:5000` | `causal_clustering.discovery_advertised_address` |
| Transaction | `6000` | `:6000` | `causal_clustering.transaction_advertised_address` |
| RAFT | `7000` | `:7000` | `causal_clustering.raft_advertised_address` |
| Routing connector | `7688` | `:7688` | `dbms.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 cluster
- Settings reference

## 7.4.6. Graphite monitoring

Default port: `2003`

*Table 29. Graphite*

| Related configuration setting | Default value | Description |
| --- | --- | --- |
| `metrics.graphite.server` | `:2003` | Hostname/IP-address and port of the Graphite 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 Graphite and the Graphite official documentation.

## 7.4.7. Prometheus monitoring

Default port: `2004`

*Table 30. Prometheus*

| Related configuration setting | Default value | Description |
| --- | --- | --- |
| `metrics.prometheus.endpoint` | `localhost:2004` | Network interface and port for the Prometheus endpoint to listen on. |

| Related configuration setting | Default value | Description |
| --- | --- | --- |
| `metrics.prometheus.enabled` | `false` | Enable exporting metrics with the Prometheus endpoint. |

For more information, see Prometheus.

### 7.4.8. JMX monitoring

Default port: `3637`

*Table 31. Java Management Extensions*

| Related configuration setting | Default value | Description |
| --- | --- | --- |
| `dbms.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.

### 7.4.9. Remote debugging

Default port: `5005`

*Table 32. Remote debugging*

| Related configuration setting | Default value | Description |
| --- | --- | --- |
| `dbms.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.

## 7.5. Configure connectors

### 7.5.1. Available connectors

The table below lists the available Neo4j connectors:

*Table 33. Neo4j connectors and port number*

| Connector name | Protocol | Default port number |
| --- | --- | --- |
| `dbms.connector.bolt` | Bolt | `7687` |
| `dbms.connector.http` | HTTP | `7474` |
| `dbms.connector.https` | HTTPS | `7473` |

When configuring the HTTPS or Bolt connector, see also SSL framework for details on how to work with SSL certificates.

## 7.5.2. Configuration options

The connectors are configured by settings on the format `dbms.connector.<connector-name>.<setting-suffix>>`. The available suffixes are described in the table below:

*Table 34. Configuration option suffixes for connectors*

| Option name | Default | Setting(s) | Description |
|---|---|---|---|
| `enabled` | `true` [7] | `dbms.connector.bolt.enabled`, `dbms.connector.http.enabled`, `dbms.connector.https.enabled` [8] | 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:<connector-default-port>` | `dbms.connector.bolt.listen_address`, `dbms.connector.https.listen_address`, `dbms.connector.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:<connector-default-port>` | `dbms.connector.bolt.advertised_address`, `dbms.connector.https.advertised_address`, `dbms.connector.http.advertised_address` | This setting specifies the address that clients should use for this connector. This is useful in a Causal Cluster as it allows each server to correctly advertise 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` | `dbms.connector.bolt.tls_level` | This setting is only applicable to the Bolt connector. It allows the 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.<br><br>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:

```
dbms.connector.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 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:

```
dbms.connector.bolt.advertised_address=<server-name>:9000
```

## 7.5.3. Options for Bolt thread pooling

See Bolt thread pool configuration to learn more about Bolt thread pooling and how to configure it on the connector level.

## 7.5.4. Defaults for addresses

It is possible to specify defaults for the configuration options with `listen_address` and `advertised_address` suffixes, as described below. Setting a default value will apply to all the connectors, unless specifically

configured for a certain connector.

`dbms.default_listen_address`

This configuration option defines a default IP address of the settings with the `listen_address` suffix for all connectors. If the IP address part of the `listen_address` is not specified, it is inherited from the shared setting `dbms.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:

```
dbms.connector.bolt.listen_address=0.0.0.0:7000
```

This is equivalent to specifying the IP address by using the `dbms.default_listen_address` setting, and then specifying the port number for the Bolt connector.

```
dbms.default_listen_address=0.0.0.0
dbms.connector.bolt.listen_address=:7000
```

`dbms.default_advertised_address`

This configuration option defines a default address of the settings with the `advertised_address` suffix for all connectors. If the address part of the `advertised_address` is not specified, it is inherited from the shared setting `dbms.default_advertised_address`.

*Example 27. Specify `advertised_address` for the Bolt connector*

Specify the address that clients should use for the Bolt connector:

```
dbms.connector.bolt.advertised_address=server1:9000
```

This is equivalent to specifying the address by using the `dbms.default_advertised_address` setting, and then specifying the port number for the Bolt connector.

```
dbms.default_advertised_address=server1
dbms.connector.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 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.

## 7.6. Set an initial password

Use the `set-initial-password` command of `neo4j-admin` to define the password for the native user `neo4j`. This must be performed before starting up the database for the first time.

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:**

It is recommended that passwords have at least 8 characters.

**Syntax:**

```
neo4j-admin set-initial-password <password> [--require-password-change]
```

*Example 28. Use the `set-initial-password` command of neo4j-admin*

Set the password for the native `neo4j` user to 'h6u4%kr' before starting the database for the first time.

```
$neo4j-home> bin/neo4j-admin set-initial-password h6u4%kr
```

*Example 29. Use the `set-initial-password` command of neo4j-admin with the optional `--require-password-change` flag*

Set the password for the native `neo4j` user to 'secretpassword' before starting the database for the first time. You will be prompted to change this password to one of your own choice at first login.

```
$neo4j-home> bin/neo4j-admin set-initial-password secretpassword --require-password-change
```

## 7.7. Configure plugins

Neo4j distributions come bundled with a range of pre-installed products, such as Graph Data Science library, Bloom, and Ops Manager, which can be used to extend the Neo4j capabilities. The JAR files for these products are located in the *product* and *labs* folders and can be installed as plugins.

If you want to use your own plugins, ensure that you add them to the designated *plugins* directory. This directory serves as the central location where Neo4j looks for and loads the plugins at startup.



**Note:**

Bloom and GDS Enterprise require a license activation key to become available to the user

within Neo4j. Reach out to your Neo4j account representative or request a representative to contact you.

The following plugins are supported:

*Table 35. Supported Neo4j plugins*

| Name | Key | License required | Further information |
|---|---|---|---|
| APOC | `apoc` | ✗ | APOC user guide |
| APOC Core | `apoc-core` | ✗ | APOC user guide |
| Bloom | `bloom` | ✓ | Neo4j Bloom |
| Streams | `streams` | ✗ | Neo4j Streaming Data Integrations User Guide |
| GraphQL | `graphql` | ✗ | Neo4j GraphQL Library |
| Graph Algorithms | `graph-algorithms` | ✗ | The Neo4j Graph Data Science Library Manual |
| Graph Data Science | `graph-data-science` | ✗ | The Neo4j Graph Data Science Library Manual |
| Neo Semantics | `n10s` | ✗ | Neo Semantics |

For more information on using plugins in a different Neo4j setup, see Neo4j Desktop → Install a plugin, Docker → Configure Neo4j plugins, Kubernetes → Plugins, and Java-Reference → Setting up a plugin project.

## 7.7.1. Install and configure plugins

Here are the steps to enable the plugins:

1. Move or copy the plugins (*.jar files*) from *<NEO4J_HOME>/products* and *<NEO4J_HOME>/labs* to the *<NEO4J_HOME>/plugins* directory. See the table in File locations for more information.
2. Add the following lines in *$NEO4J_HOME/conf/neo4j.conf*:

```
# to enable GDS:

* dbms.security.procedures.unrestricted=gds.*
* dbms.security.procedures.allowlist=gds.*
* gds.enterprise.license_file=/path/to/my/license/keyfile


#  to enable Bloom:

* dbms.security.procedures.unrestricted=bloom.*
* dbms.bloom.license_file=/path/to/my/license/keyfile


#  to enable both GDS and Bloom:

* dbms.security.procedures.unrestricted=gds.*,bloom.*
* dbms.security.procedures.allowlist=gds.*
* gds.enterprise.license_file=/path/to/my/license/keyfile
* dbms.bloom.license_file=/path/to/my/license/keyfile
```

3. Install the plugins.

   Refer to Bloom documentation, GDS documentation, and Neo4j Ops Manager documentation for more details on how to install them.

> **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.

# 7.8. Update dynamic settings

## 7.8.1. Introduction

Neo4j Enterprise Edition supports changing some configuration settings at runtime, without restarting the service. Changes to the configuration at runtime are not persisted. To avoid losing changes when restarting Neo4j, you must ensure that the *neo4j.conf* file is also updated.

> **Note:**
>
> Cluster
>
> In a clustered environment, `CALL dbms.setConfigValue` affects only the cluster member it is run against, and it is not propagated to other members. Therefore, you should connect using `bolt://` rather than `neo4j://` to ensure that the setting is changed on the member connected to, and not on the unintended leader. 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.

## 7.8.2. Discover dynamic settings

Use the procedure `dbms.listConfig()` to discover which configuration values can be dynamically updated, or see the list below.

*Example 30. Discover dynamic settings*

```
CALL dbms.listConfig()
YIELD name, dynamic
WHERE dynamic
RETURN name
ORDER BY name;
```

```
+-------------------------------------------------------------+
| name                                                        |
+-------------------------------------------------------------+
| "causal_clustering.cluster_allow_reads_on_leader"           |
| "causal_clustering.connect_randomly_to_server_group"        |
| "causal_clustering.server_groups"                           |
| "dbms.allow_single_automatic_upgrade"                       |
```

```
| "dbms.allow_upgrade"                                                |
| "dbms.backup.incremental.strategy"                                  |
| "dbms.checkpoint.iops.limit"                                        |
| "dbms.databases.default_to_read_only"                               |
| "dbms.databases.read_only"                                          |
| "dbms.databases.writable"                                           |
| "dbms.lock.acquisition.timeout"                                     |
| "dbms.logs.debug.level"                                             |
| "dbms.logs.query.allocation_logging_enabled"                        |
| "dbms.logs.query.early_raw_logging_enabled"                         |
| "dbms.logs.query.enabled"                                           |
| "dbms.logs.query.max_parameter_length"                              |
| "dbms.logs.query.obfuscate_literals"                                |
| "dbms.logs.query.obfuscate_errors"                                  |
| "dbms.logs.query.page_logging_enabled"                              |
| "dbms.logs.query.parameter_full_entities"                           |
| "dbms.logs.query.parameter_logging_enabled"                         |
| "dbms.logs.query.plan_description_enabled"                          |
| "dbms.logs.query.rotation.keep_number"                              |
| "dbms.logs.query.rotation.size"                                     |
| "dbms.logs.query.runtime_logging_enabled"                           |
| "dbms.logs.query.threshold"                                         |
| "dbms.logs.query.time_logging_enabled"                              |
| "dbms.logs.query.transaction.enabled"                               |
| "dbms.logs.query.transaction.threshold"                             |
| "dbms.logs.query.transaction_id.enabled"                            |
| "dbms.memory.pagecache.flush.buffer.enabled"                        |
| "dbms.memory.pagecache.flush.buffer.size_in_pages"                  |
| "dbms.memory.transaction.database_max_size"                         |
| "dbms.memory.transaction.global_max_size"                           |
| "dbms.memory.transaction.max_size"                                  |
| "dbms.routing.client_side.enforce_for_domains"                      |
| "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.user_search_base"                 |
| "dbms.security.ldap.authorization.user_search_filter"               |
| "dbms.track_query_allocation"                                       |
| "dbms.track_query_cpu_time"                                         |
| "dbms.transaction.bookmark_ready_timeout"                           |
| "dbms.transaction.concurrent.maximum"                               |
| "dbms.transaction.sampling.percentage"                              |
| "dbms.transaction.timeout"                                          |
| "dbms.transaction.tracing.level"                                    |
| "dbms.tx_log.preallocate"                                           |
| "dbms.tx_log.rotation.retention_policy"                             |
| "dbms.tx_log.rotation.size"                                         |
| "dbms.upgrade_max_processors"                                       |
| "fabric.routing.servers"                                            |
| "systemdb.secrets.key.name"                                         |
| "systemdb.secrets.keystore.password"                                |
| "systemdb.secrets.keystore.path"                                    |
+---------------------------------------------------------------------+
57 rows
```

## 7.8.3. 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 can not be changed at runtime. |
| Invalid setting value. |

The following example shows how to dynamically enable query logging.

*Example 31. Set a config value*

```
CALL dbms.setConfigValue('dbms.logs.query.enabled', 'info')
```

If an invalid value is passed, the procedure will show a message to that effect.

*Example 32. Try to set invalid config value*

```
CALL dbms.setConfigValue('dbms.logs.query.enabled', 'yes')
```

```
Failed to invoke procedure `dbms.setConfigValue`: Caused by:
org.neo4j.graphdb.config.InvalidSettingException: Bad value 'yes' for setting
'dbms.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 33. Reset a config value to default*

```
CALL dbms.setConfigValue('dbms.logs.query.enabled', '')
```

# 7.9. Transaction log

- The transaction log record all write operations in the database.
- The transaction log is the "source of truth" in scenarios where the database needs to be recovered.
- The transaction log can be used to provide for incremental backups, as well as for cluster operations.
- For any given configuration, at least the latest non-empty transaction log will be kept.

Each database keeps its own directory with *transaction logs*. The root directory where the transaction log folders are located is configured by `dbms.directories.transaction.logs.root`.



**Note:**

The transaction log has nothing to do with log monitoring.

## 7.9.1. Transaction logging

The transaction logs record all write operations in the database. This includes additions or modifications to data, as well as the addition or modification of any indexes or constraints.

- The transaction logs are the "source of truth" in scenarios where the database needs to be recovered.
- The transaction logs are used for providing incremental backups, as well as for cluster operations.
- For any given configuration, at least the latest non-empty transaction log will be kept.

An overview of configuration settings for transaction logging:

| The *transaction log* configuration | Default value | Description |
|---|---|---|
| `dbms.directories.transaction.logs.root` | `transactions` | Root location where Neo4j will store transaction logs for configured databases. |
| `dbms.tx_log.preallocate` | `true` | Specify if Neo4j should try to preallocate logical log file in advance. |
| `dbms.tx_log.rotation.retention_policy` | `7 days` | Make Neo4j keep the logical transaction logs for being able to backup the database. Can be used for specifying the threshold to prune logical logs after. |
| `dbms.tx_log.rotation.size` | `250M` | Specifies at which file size the logical log will auto-rotate. Minimum accepted value is `128K` (128 KiB). |

The retention and rotation policies for the Neo4j transaction logs, and how to configure them.

## 7.9.2. Log location

By default, transaction logs for a database are located at *<neo4j-home>/data/transactions/<database-name>*. Each database keeps its own directory with transaction logs.

The root directory where those folders are located is configured by `dbms.directories.transaction.logs.root`. For maximum performance, it is recommended to configure transaction logs to be stored on a dedicated device.

## 7.9.3. Log rotation

Log rotation is configured using the parameter `dbms.tx_log.rotation.size`. By default, log switches happen when log sizes surpass 250 MB.

## 7.9.4. Log retention



**Warning:**

Manually deleting transaction log files is not supported.

You can control the number of transaction logs that Neo4j keeps using the parameter `dbms.tx_log.rotation.retention_policy`. It is set to `7 days` by default, which means Neo4j keeps logical logs that contain any transaction committed within 7 days. The configuration is dynamic, so if you need to update it, you do not have to restart Neo4j for the change to take effect.

Other possible values are:

- `true` or `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.

- `false` or `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 will be kept for some time, and only after a checkpoint, the pruning strategy will remove them. For more details on how to speed up checkpointing, see Log pruning. To force a checkpoint, run the procedure `call db.checkpoint()`.

  > **Note:**
  >
  > This option is not recommended in production Enterprise Edition environments, as incremental backups rely on the presence of the transaction logs since the last backup.

- `<number><optional unit> <type>` where valid units are `k`, `M`, and `G`, and valid types are `files`, `size`, `txs`, `entries`, `hours`, and `days`.

  *Table 36. Types that can be used to control log retention*

| Type | Description | Example |
|---|---|---|
| files | The number of the most recent logical log files to keep. | "10 files" |
| size | Max disk size to allow log files to occupy. | "300M size" or "1G size". |
| txs | The number of transactions to keep. | "250k txs" or "5M txs". |
| hours | Keep logs that contain any transaction committed within N hours from the current time. | "10 hours" |
| days | Keep logs that contain any transaction committed within N days from the current time. | "50 days" |

*Example 34. Configure log retention policy*

This example shows some different ways to configure the log retention policy.

- Keep transaction logs indefinitely:

```
dbms.tx_log.rotation.retention_policy=true
```

or

```
dbms.tx_log.rotation.retention_policy=keep_all
```

- Keep only the most recent non-empty log:

```
dbms.tx_log.rotation.retention_policy=false
```

or

```
dbms.tx_log.rotation.retention_policy=keep_none
```

- Keep logical logs which contain any transaction committed within 30 days:

```
dbms.tx_log.rotation.retention_policy=30 days
```

- Keep logical logs which contain any of the most recent 500 000 transactions:

```
dbms.tx_log.rotation.retention_policy=500k txs
```

## 7.9.5. Log pruning

Transaction log pruning refers to the safe and automatic removal of old, unnecessary transaction log files. The transaction log can be pruned when one or more files fall outside of the configured retention policy.

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.

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 expected and observed number of log files will be closed on the next successful checkpoint. The interval between checkpoints can be configured using:

| Checkpoint configuration | Default value | Description |
| --- | --- | --- |
| `dbms.checkpoint.interval.time` | `15m` | Configures the time interval between check-points. |
| `dbms.checkpoint.interval.tx` | `100000` | Configures the transaction interval between check-points. |

If your goal is to have the least amount of transaction log data, it can also help to speed up the checkpoint process itself. The configuration parameter `dbms.checkpoint.iops.limit` controls the number of IOs per second the checkpoint process is allowed to use. Setting the value of this parameter to `-1` allows unlimited IOPS, which can speed up checkpointing.



Note:

Disabling the IOPS limit can cause transaction processing to slow down a bit. For more information, see Checkpoint IOPS limit.

# 7.10. Configuration settings

This page provides a complete reference to the Neo4j configuration settings, which can be set in *neo4j.conf*. Refer to The *neo4j.conf* file for details on how to use configuration settings.

Some of the settings are labeled ***Dynamic***, which means that they can be changed at runtime, without restarting the service. For more information on how to update dynamic configuration settings, see Update dynamic settings.

## 7.10.1. Configuration setting group for the load balancing framework

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:

```
causal_clustering.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:

```
causal_clustering.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.

See Appendix → Advanced Causal Clustering → Multi-data center load balancing for more details.

## 7.10.2. Checkpoint settings

### `dbms.checkpoint`

*Table 37. dbms.checkpoint*

| | |
|---|---|
| Description | Configures the general policy for when check-points should occur. The default policy is the 'periodic' check-point policy, as specified by the 'dbms.checkpoint.interval.tx' and 'dbms.checkpoint.interval.time' settings. The Neo4j Enterprise Edition provides two alternative policies: The first is the 'continuous' check-point policy, which will ignore those settings and run the check-point process all the time. The second is the 'volumetric' check-point policy, which makes a best-effort at check-pointing often enough so that the database doesn't get too far behind on deleting old transaction logs in accordance with the 'dbms.tx_log.rotation.retention_policy' setting. |
| Valid values | dbms.checkpoint, one of [PERIODIC, CONTINUOUS, VOLUME, VOLUMETRIC] |
| Default value | `PERIODIC` |

### `dbms.checkpoint.interval.time`

*Table 38. dbms.checkpoint.interval.time*

| | |
|---|---|
| Description | Configures the time interval between check-points. The database will not check-point more often than this (unless check pointing is triggered by a different event), but might check-point less often than this interval, if performing a check-point takes longer time than the configured interval. A check-point is a point in the transaction logs, which recovery would start from. Longer check-point intervals typically mean that recovery will take longer to complete in case of a crash. On the other hand, a longer check-point interval can also reduce the I/O load that the database places on the system, as each check-point implies a flushing and forcing of all the store files. |
| Valid values | dbms.checkpoint.interval.time, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `15m` |

### `dbms.checkpoint.interval.tx`

*Table 39. dbms.checkpoint.interval.tx*

| | |
|---|---|
| Description | Configures the transaction interval between check-points. The database will not check-point more often than this (unless check pointing is triggered by a different event), but might check-point less often than this interval, if performing a check-point takes longer time than the configured interval. A check-point is a point in the transaction logs, which recovery would start from. Longer check-point intervals typically mean that recovery will take longer to complete in case of a crash. On the other hand, a longer check-point interval can also reduce the I/O load that the database places on the system, as each check-point implies a flushing and forcing of all the store files. The default is '100000' for a check-point every 100000 transactions. |
| Valid values | dbms.checkpoint.interval.tx, an integer which is minimum `1` |
| Default value | `100000` |

### `dbms.checkpoint.interval.volume`

*Table 40. dbms.checkpoint.interval.volume*

| | |
|---|---|
| Description | Configures the volume of transacton logs between check-points. The database will not check-point more often than this (unless check pointing is triggered by a different event), but might check-point less often than this interval, if performing a check-point takes longer time than the configured interval. A check-point is a point in the transaction logs, which recovery would start from. Longer check-point intervals typically mean that recovery will take longer to complete in case of a crash. On the other hand, a longer check-point interval can also reduce the I/O load that the database places on the system, as each check-point implies a flushing and forcing of all the store files. |
| Valid values | dbms.checkpoint.interval.volume, 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`, `PiB`, `PB`, `EiB`, `EB`) which is minimum `1.00KiB` |
| Default value | `250.00MiB` |

### `dbms.checkpoint.iops.limit`

Dynamic

*Table 41. dbms.checkpoint.iops.limit*

| | |
|---|---|
| Description | Limit the number of IOs the background checkpoint process will consume per second. This setting is advisory, is ignored in Neo4j Community Edition, and is followed to best effort in Enterprise Edition. An IO is in this case a 8 KiB (mostly sequential) write. Limiting the write IO in this way will leave 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 will let the checkpointer flush data as fast as the hardware will go. Removing the setting, or commenting it out, will set the default value of 600. |
| Valid values | dbms.checkpoint.iops.limit, an integer |
| Default value | `600` |

## 7.10.3. Cluster settings

### `causal_clustering.catch_up_client_inactivity_timeout`

*Table 42. causal_clustering.catch_up_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 time out duration. |
| Valid values | causal_clustering.catch_up_client_inactivity_timeout, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `10m` |

### `causal_clustering.catchup_batch_size`

*Table 43. causal_clustering.catchup_batch_size*

| | |
|---|---|
| Description | The maximum batch size when catching up (in unit of entries) |
| Valid values | causal_clustering.catchup_batch_size, an integer |
| Default value | `64` |

## causal_clustering.cluster_allow_reads_on_followers

*Table 44. causal_clustering.cluster_allow_reads_on_followers*

| | |
|---|---|
| Description | Configure if the `dbms.routing.getRoutingTable()` procedure should include followers as read endpoints or return only read replicas. Note: if there are no read replicas in the cluster, followers are returned as read end points regardless the value of this setting. Defaults to true so that followers are available for read-only queries in a typical heterogeneous setup. |
| Valid values | causal_clustering.cluster_allow_reads_on_followers, a boolean |
| Default value | `true` |

## causal_clustering.cluster_allow_reads_on_leader

Dynamic

*Table 45. causal_clustering.cluster_allow_reads_on_leader*

| | |
|---|---|
| Description | Configure if the `dbms.routing.getRoutingTable()` procedure should include the leader as read endpoint or return only read replicas/followers. Note: leader is returned as read endpoint if no other member is present all. |
| Valid values | causal_clustering.cluster_allow_reads_on_leader, a boolean |
| Default value | `false` |

## causal_clustering.cluster_binding_timeout

*Table 46. causal_clustering.cluster_binding_timeout*

| | |
|---|---|
| Description | The time allowed for a database on a Neo4j server to either join a cluster or form a new cluster with the other Neo4j Core Servers provided by `causal_clustering.initial_discovery_members`. |
| Valid values | causal_clustering.cluster_binding_timeout, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `10m` |

## causal_clustering.cluster_topology_refresh

*Table 47. causal_clustering.cluster_topology_refresh*

| | |
|---|---|
| Description | Time between scanning the cluster to refresh current server's view of topology. |
| Valid values | causal_clustering.cluster_topology_refresh, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) which is minimum `1s` |

| | |
|---|---|
| Default value | `5s` |

## `causal_clustering.command_applier_parallelism`

*Table 48. causal_clustering.command_applier_parallelism*

| | |
|---|---|
| Description | Limits amount of global threads for applying commands. |
| Valid values | causal_clustering.command_applier_parallelism, an integer which is minimum `1` |
| Default value | `8` |

## `causal_clustering.connect_randomly_to_server_group`

Dynamic

*Table 49. causal_clustering.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 if the list of strategies (`causal_clustering.upstream_selection_strategy`) includes the value `connect-randomly-to-server-group`. |
| Valid values | causal_clustering.connect_randomly_to_server_group, a ',' separated list with elements of type 'a string identifying a Server Group'. |
| Default value | |

## `causal_clustering.delete_store_before_store_copy`

*Table 50. causal_clustering.delete_store_before_store_copy*

| | |
|---|---|
| Description | Deletes the old store (on cores and replicas) before performing a store copy (instead of deleting it after). |
| Valid values | causal_clustering.delete_store_before_store_copy, a boolean |
| Default value | `true` |

## `causal_clustering.discovery_advertised_address`

*Table 51. causal_clustering.discovery_advertised_address*

| | |
|---|---|
| Description | Advertised cluster member discovery management communication. |
| Valid values | causal_clustering.discovery_advertised_address, a socket address in the format 'hostname:port', 'hostname' or ':port'. If missing port or hostname it is acquired from dbms.default_advertised_address |
| Default value | `:5000` |

## `causal_clustering.discovery_listen_address`

*Table 52. causal_clustering.discovery_listen_address*

| | |
|---|---|
| Description | Host and port to bind the cluster member discovery management communication. |
| Valid values | causal_clustering.discovery_listen_address, a socket address in the format 'hostname:port', 'hostname' or ':port'. If missing port or hostname it is acquired from dbms.default_listen_address |
| Default value | `:5000` |

## causal_clustering.discovery_type

*Table 53. causal_clustering.discovery_type*

| | |
|---|---|
| Description | Configure the discovery type used for cluster name resolution. |
| Valid values | causal_clustering.discovery_type, one of [DNS, LIST, SRV, K8S] which depends on dbms.mode. If dbms.mode one of `[CORE, READ_REPLICA]` then it may require different settings depending on the discovery type: `DNS requires [causal_clustering.initial_discovery_members], LIST requires [causal_clustering.initial_discovery_members], SRV requires [causal_clustering.initial_discovery_members], K8S requires [causal_clustering.kubernetes.label_selector, causal_clustering.kubernetes.service_port_name]` otherwise it depends on dbms.mode. If dbms.mode one of `[SINGLE]` then it depends on dbms.clustering.enable. If dbms.clustering.enable is `true` then it may require different settings depending on the discovery type: `DNS requires [causal_clustering.initial_discovery_members], LIST requires [causal_clustering.initial_discovery_members], SRV requires [causal_clustering.initial_discovery_members], K8S requires [causal_clustering.kubernetes.label_selector, causal_clustering.kubernetes.service_port_name]` otherwise it is unconstrained. otherwise it is unconstrained.. |
| Default value | `LIST` |

## causal_clustering.election_failure_detection_window

*Table 54. causal_clustering.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 | causal_clustering.election_failure_detection_window, a duration-range <min-max> (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `3s-6s` |

## causal_clustering.enable_pre_voting

*Table 55. causal_clustering.enable_pre_voting*

| | |
|---|---|
| Description | Enable pre-voting extension to the Raft protocol (this is breaking and must match between the core cluster members) |
| Valid values | causal_clustering.enable_pre_voting, a boolean |
| Default value | `true` |

## causal_clustering.global_session_tracker_state_size

*Table 56. causal_clustering.global_session_tracker_state_size*

| | |
|---|---|
| Description | The maximum file size before the global session tracker state file is rotated (in unit of entries) |
| Valid values | causal_clustering.global_session_tracker_state_size, an integer |
| Default value | `1000` |

## causal_clustering.handshake_timeout

*Table 57. causal_clustering.handshake_timeout*

| | |
|---|---|
| Description | Time out for protocol negotiation handshake. This configuration is applicable to: `Raft` (communication between `CORE` instances only), `Catchup` (communication between any instances: `CORE` → `CORE`, `RR` → `CORE`, `RR` → `RR`, `CORE` → `RR`, including `RR` → `SINGLE` in a replica-only cluster). `Backup` (communication between any instance and a backup client that lives in the `neo4j-admin` command, such as `BackupClient` → `SINGLE`, `BackupClient` → `CORE`, `BackupClient` → `RR`). |
| Valid values | causal_clustering.handshake_timeout, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `20s` |

## causal_clustering.in_flight_cache.max_bytes

*Table 58. causal_clustering.in_flight_cache.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 cache. If the bytes limit is reached, cache size will be limited even if max_entries is not exceeded. |
| Valid values | causal_clustering.in_flight_cache.max_bytes, 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`, `PiB`, `PB`, `EiB`, `EB`) |
| Default value | `2.00GiB` |

## causal_clustering.in_flight_cache.max_entries

*Table 59. causal_clustering.in_flight_cache.max_entries*

| | |
|---|---|
| Description | The maximum number of entries in the in-flight cache. Increasing size will require more memory but might improve performance in high load situations. |
| Valid values | causal_clustering.in_flight_cache.max_entries, an integer |
| Default value | `1024` |

## causal_clustering.in_flight_cache.type

*Table 60. causal_clustering.in_flight_cache.type*

| | |
|---|---|
| Description | Type of in-flight cache. CONSECUTIVE should be used for production instances, NONE will disable cache which might be useful in specific support cases. UNBOUNDED is for internal use only. |
| Valid values | causal_clustering.in_flight_cache.type, one of [NONE, CONSECUTIVE, UNBOUNDED] |

| | |
|---|---|
| Default value | `CONSECUTIVE` |

## `causal_clustering.initial_discovery_members`

*Table 61. causal_clustering.initial_discovery_members*

| | |
|---|---|
| Description | A comma-separated list of other members of the cluster to join. |
| Valid values | causal_clustering.initial_discovery_members, a ',' separated list with elements of type 'a socket address in the format 'hostname:port', 'hostname' or ':port''. |

## `causal_clustering.join_catch_up_max_lag`

| | |
|---|---|
| Description | Maximum amount of lag accepted for a new follower to join the Raft group. |
| Valid values | causal_clustering.join_catch_up_max_lag, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `10s` |

## `causal_clustering.join_catch_up_timeout`

*Table 62. causal_clustering.join_catch_up_timeout*

| | |
|---|---|
| Description | Time out for a new member to catch up. |
| Valid values | causal_clustering.join_catch_up_timeout, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `10m` |

## `causal_clustering.kubernetes.address`

*Table 63. causal_clustering.kubernetes.address*

| | |
|---|---|
| Description | Address for Kubernetes API. |
| Valid values | causal_clustering.kubernetes.address, a socket address in the format 'hostname:port', 'hostname' or ':port' |
| Default value | `kubernetes.default.svc:443` |

## `causal_clustering.kubernetes.ca_crt`

*Table 64. causal_clustering.kubernetes.ca_crt*

| | |
|---|---|
| Description | File location of CA certificate for Kubernetes API. |
| Valid values | causal_clustering.kubernetes.ca_crt, a path |
| Default value | `/var/run/secrets/kubernetes.io/serviceaccount/ca.crt` |

## `causal_clustering.kubernetes.cluster_domain`

*Table 65. causal_clustering.kubernetes.cluster_domain*

| | |
|---|---|
| Description | Kubernetes cluster domain. |
| Valid values | causal_clustering.kubernetes.cluster_domain, a string |
| Default value | `cluster.local` |

## causal_clustering.kubernetes.label_selector

*Table 66. causal_clustering.kubernetes.label_selector*

| | |
|---|---|
| Description | LabelSelector for Kubernetes API. |
| Valid values | causal_clustering.kubernetes.label_selector, a string |

## causal_clustering.kubernetes.namespace

*Table 67. causal_clustering.kubernetes.namespace*

| | |
|---|---|
| Description | File location of namespace for Kubernetes API. |
| Valid values | causal_clustering.kubernetes.namespace, a path |
| Default value | `/var/run/secrets/kubernetes.io/serviceaccount/namespace` |

## causal_clustering.kubernetes.service_port_name

*Table 68. causal_clustering.kubernetes.service_port_name*

| | |
|---|---|
| Description | Service port name for discovery for Kubernetes API. |
| Valid values | causal_clustering.kubernetes.service_port_name, a string |

## causal_clustering.kubernetes.token

*Table 69. causal_clustering.kubernetes.token*

| | |
|---|---|
| Description | File location of token for Kubernetes API. |
| Valid values | causal_clustering.kubernetes.token, a path |
| Default value | `/var/run/secrets/kubernetes.io/serviceaccount/token` |

## causal_clustering.last_applied_state_size

| | |
|---|---|
| Description | The maximum file size before the storage file is rotated (in unit of entries) |
| Valid values | causal_clustering.last_applied_state_size, an integer |
| Default value | `1000` |

## causal_clustering.leader_election_timeout

Deprecated

*Table 70. causal_clustering.leader_election_timeout*

| | |
|---|---|
| Description | This setting is moved and enhanced into causal_clustering.leader_failure_detection_window and causal_clustering.election_failure_detection_window. |
| Valid values | causal_clustering.leader_election_timeout, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `7s` |

## `causal_clustering.leader_failure_detection_window`

*Table 71. causal_clustering.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 | causal_clustering.leader_failure_detection_window, a duration-range <min-max> (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `20s-23s` |

## `causal_clustering.leadership_balancing`

*Table 72. causal_clustering.leadership_balancing*

| | |
|---|---|
| Description | Which strategy to use when transferring database leaderships around a cluster. This can be one of `equal_balancing` or `no_balancing`. `equal_balancing` automatically ensures that each Core server holds the leader role for an equal number of databases. `no_balancing` prevents any automatic balancing of the leader role.Note that if a `leadership_priority_group` is specified for a given database, the value of this setting will be ignored for that database. |
| Valid values | causal_clustering.leadership_balancing, one of [NO_BALANCING, EQUAL_BALANCING] |
| Default value | `EQUAL_BALANCING` |

## `causal_clustering.leadership_priority_group`

*Table 73. causal_clustering.leadership_priority_group*

| | |
|---|---|
| Description | The name of a server_group whose members should be prioritized as leaders. This does not guarantee that members of this group will be leader at all times, but the cluster will attempt to transfer leadership to such a member when possible. If a database is specified using `causal_clustering.leadership_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 which have no priority group explicitly set. Using this setting will disable leadership balancing. |
| Valid values | causal_clustering.leadership_priority_group, a string identifying a Server Group |
| Default value | |

## `causal_clustering.load_balancing.plugin`

*Table 74. causal_clustering.load_balancing.plugin*

| | |
|---|---|
| **Description** | The load balancing plugin to use. |
| **Valid values** | causal_clustering.load_balancing.plugin, a string which depends on dbms.mode. If dbms.mode one of `[CORE]` then it specified load balancer plugin exist. otherwise it is unconstrained. |
| **Default value** | `server_policies` |

## `causal_clustering.load_balancing.shuffle`

*Table 75. causal_clustering.load_balancing.shuffle*

| | |
|---|---|
| **Description** | Enables shuffling of the returned load balancing result. |
| **Valid values** | causal_clustering.load_balancing.shuffle, a boolean |
| **Default value** | `true` |

## `causal_clustering.log_shipping_max_lag`

*Table 76. causal_clustering.log_shipping_max_lag*

| | |
|---|---|
| **Description** | The maximum lag allowed before log shipping pauses (in unit of entries) |
| **Valid values** | causal_clustering.log_shipping_max_lag, an integer |
| **Default value** | `256` |

## `causal_clustering.log_shipping_retry_timeout`

*Table 77. causal_clustering.log_shipping_retry_timeout*

| | |
|---|---|
| **Description** | Retry time for log shipping to followers after a stall. |
| **Valid values** | causal_clustering.log_shipping_retry_timeout, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| **Default value** | `5s` |

## `causal_clustering.max_raft_channels`

*Table 78. causal_clustering.max_raft_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** | causal_clustering.max_raft_channels, an integer |
| **Default value** | `8` |

## `causal_clustering.middleware.logging.level`

*Table 79. causal_clustering.middleware.logging.level*

| | |
|---|---|
| Description | The level of middleware logging. |
| Valid values | causal_clustering.middleware.logging.level, one of [DEBUG, INFO, WARN, ERROR, NONE] |
| Default value | `WARN` |

## causal_clustering.minimum_core_cluster_size_at_formation

*Table 80. causal_clustering.minimum_core_cluster_size_at_formation*

| | |
|---|---|
| Description | Minimum number of Core machines initially required to form a cluster. The cluster will form when at least this many Core members have discovered each other. |
| Valid values | causal_clustering.minimum_core_cluster_size_at_formation, an integer which is minimum `2` |
| Default value | `3` |

## causal_clustering.minimum_core_cluster_size_at_runtime

*Table 81. causal_clustering.minimum_core_cluster_size_at_runtime*

| | |
|---|---|
| Description | The minimum size of the dynamically adjusted voting set (which only core members may be a part of). Adjustments to the voting set happen automatically as the availability of core members changes, due to explicit operations such as starting or stopping a member, or unintended issues such as network partitions. Note that this dynamic scaling of the voting set is generally desirable as under some circumstances it can increase the number of instance failures which may be tolerated. A majority of the voting set must be available before voting in or out members. |
| Valid values | causal_clustering.minimum_core_cluster_size_at_runtime, an integer which is minimum `2` and depends on dbms.mode. If dbms.mode one of `[CORE]` then it must be set less than or equal to value of 'causal_clustering.minimum_core_cluster_size_at_formation' otherwise it is unconstrained. |
| Default value | `3` |

## causal_clustering.multi_dc_license

*Table 82. causal_clustering.multi_dc_license*

| | |
|---|---|
| Description | Enable multi-data center features. Requires appropriate licensing. |
| Valid values | causal_clustering.multi_dc_license, a boolean |
| Default value | `false` |

## causal_clustering.protocol_implementations.catchup

*Table 83. causal_clustering.protocol_implementations.catchup*

| | |
|---|---|
| Description | Catchup protocol implementation versions that this instance will allow in negotiation as a comma-separated list. Order is not relevant: the greatest value will be preferred. An empty list will allow all supported versions. Example value: "1.1, 1.2, 2.1, 2.2" |
| Valid values | causal_clustering.protocol_implementations.catchup, a ',' separated list with elements of type 'an application protocol version'. |

| | |
|---|---|
| Default value | |

## causal_clustering.protocol_implementations.compression

*Table 84. causal_clustering.protocol_implementations.compression*

| | |
|---|---|
| Description | Network compression algorithms that this instance will allow in negotiation as a comma-separated list. Listed in descending order of preference for incoming connections. 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 | causal_clustering.protocol_implementations.compression, a ',' separated list with elements of type 'a string'. |
| Default value | |

## causal_clustering.protocol_implementations.raft

*Table 85. causal_clustering.protocol_implementations.raft*

| | |
|---|---|
| Description | Raft protocol implementation versions that this instance will allow in negotiation as a comma-separated list. Order is not relevant: the greatest value will be preferred. An empty list will allow all supported versions. Example value: "1.0, 1.3, 2.0, 2.1" |
| Valid values | causal_clustering.protocol_implementations.raft, a ',' separated list with elements of type 'an application protocol version'. |
| Default value | |

## causal_clustering.pull_interval

*Table 86. causal_clustering.pull_interval*

| | |
|---|---|
| Description | Interval of pulling updates from cores. |
| Valid values | causal_clustering.pull_interval, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `1s` |

## causal_clustering.raft_advertised_address

*Table 87. causal_clustering.raft_advertised_address*

| | |
|---|---|
| Description | Advertised hostname/IP address and port for the RAFT server. |
| Valid values | causal_clustering.raft_advertised_address, a socket address in the format 'hostname:port', 'hostname' or ':port'. If missing port or hostname it is acquired from dbms.default_advertised_address |
| Default value | `:7000` |

## causal_clustering.raft_handler_parallelism

*Table 88. causal_clustering.raft_handler_parallelism*

| | |
|---|---|
| Description | Limits amount of global threads shared by raft groups for handling bathing of messages and timeout events. |
| Valid values | causal_clustering.raft_handler_parallelism, an integer which is minimum `1` |
| Default value | `8` |

## causal_clustering.raft_in_queue_max_batch_bytes

*Table 89. causal_clustering.raft_in_queue_max_batch_bytes*

| | |
|---|---|
| Description | Largest batch processed by RAFT in bytes. |
| Valid values | causal_clustering.raft_in_queue_max_batch_bytes, 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`, `PiB`, `PB`, `EiB`, `EB`) |
| Default value | `8.00MiB` |

## causal_clustering.raft_in_queue_max_bytes

*Table 90. causal_clustering.raft_in_queue_max_bytes*

| | |
|---|---|
| Description | Maximum number of bytes in the RAFT in-queue. |
| Valid values | causal_clustering.raft_in_queue_max_bytes, 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`, `PiB`, `PB`, `EiB`, `EB`) |
| Default value | `2.00GiB` |

## causal_clustering.raft_listen_address

*Table 91. causal_clustering.raft_listen_address*

| | |
|---|---|
| Description | Network interface and port for the RAFT server to listen on. |
| Valid values | causal_clustering.raft_listen_address, a socket address in the format 'hostname:port', 'hostname' or ':port'. If missing port or hostname it is acquired from dbms.default_listen_address |
| Default value | `:7000` |

## causal_clustering.raft_log_entry_prefetch_buffer.max_entries

*Table 92. causal_clustering.raft_log_entry_prefetch_buffer.max_entries*

| | |
|---|---|
| Description | The maximum number of entries in the raft log entry prefetch buffer. |
| Valid values | causal_clustering.raft_log_entry_prefetch_buffer.max_entries, an integer |
| Default value | `1024` |

## causal_clustering.raft_log_implementation

*Table 93. causal_clustering.raft_log_implementation*

| | |
|---|---|
| **Description** | RAFT log implementation. |
| **Valid values** | causal_clustering.raft_log_implementation, a string |
| **Default value** | `SEGMENTED` |

## `causal_clustering.raft_log_prune_strategy`

*Table 94. causal_clustering.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** | causal_clustering.raft_log_prune_strategy, a string |
| **Default value** | `1g size` |

## `causal_clustering.raft_log_pruning_frequency`

*Table 95. causal_clustering.raft_log_pruning_frequency*

| | |
|---|---|
| **Description** | RAFT log pruning frequency. |
| **Valid values** | causal_clustering.raft_log_pruning_frequency, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| **Default value** | `10m` |

## `causal_clustering.raft_log_reader_pool_size`

*Table 96. causal_clustering.raft_log_reader_pool_size*

| | |
|---|---|
| **Description** | RAFT log reader pool size. |
| **Valid values** | causal_clustering.raft_log_reader_pool_size, an integer |
| **Default value** | `8` |

## `causal_clustering.raft_log_rotation_size`

*Table 97. causal_clustering.raft_log_rotation_size*

| | |
|---|---|
| **Description** | RAFT log rotation size. |
| **Valid values** | causal_clustering.raft_log_rotation_size, 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`, `PiB`, `PB`, `EiB`, `EB`) which is minimum `1.00KiB` |
| **Default value** | `250.00MiB` |

## `causal_clustering.raft_membership_state_size`

*Table 98. causal_clustering.raft_membership_state_size*

| | |
|---|---|
| Description | The maximum file size before the membership state file is rotated (in unit of entries) |
| Valid values | causal_clustering.raft_membership_state_size, an integer |
| Default value | `1000` |

## `causal_clustering.raft_term_state_size`

*Table 99. causal_clustering.raft_term_state_size*

| | |
|---|---|
| Description | The maximum file size before the term state file is rotated (in unit of entries) |
| Valid values | causal_clustering.raft_term_state_size, an integer |
| Default value | `1000` |

## `causal_clustering.raft_vote_state_size`

*Table 100. causal_clustering.raft_vote_state_size*

| | |
|---|---|
| Description | The maximum file size before the vote state file is rotated (in unit of entries) |
| Valid values | causal_clustering.raft_vote_state_size, an integer |
| Default value | `1000` |

## `causal_clustering.refuse_to_be_leader`

Deprecated

*Table 101. causal_clustering.refuse_to_be_leader*

| | |
|---|---|
| Description | Deprecated, use dbms.databases.default_to_read_only |
| Valid values | causal_clustering.refuse_to_be_leader, a boolean |
| Default value | `false` |
| Replaced by | dbms.databases.default_to_read_only |

## `causal_clustering.replicated_lease_state_size`

*Table 102. causal_clustering.replicated_lease_state_size*

| | |
|---|---|
| Description | The maximum file size before the replicated lease state file is rotated (in unit of entries) |
| Valid values | causal_clustering.replicated_lease_state_size, an integer |
| Default value | `1000` |

## `causal_clustering.replication_leader_await_timeout`

*Table 103. causal_clustering.replication_leader_await_timeout*

| | |
|---|---|
| Description | The duration for which the replicator will await a new leader. |
| Valid values | causal_clustering.replication_leader_await_timeout, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `10s` |

## `causal_clustering.replication_retry_timeout_base`

*Table 104. causal_clustering.replication_retry_timeout_base*

| | |
|---|---|
| Description | The initial timeout until replication is retried. The timeout will increase exponentially. |
| Valid values | causal_clustering.replication_retry_timeout_base, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `10s` |

## `causal_clustering.replication_retry_timeout_limit`

*Table 105. causal_clustering.replication_retry_timeout_limit*

| | |
|---|---|
| Description | The upper limit for the exponentially incremented retry timeout. |
| Valid values | causal_clustering.replication_retry_timeout_limit, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `1m` |

## `causal_clustering.server_groups`

Dynamic

*Table 106. causal_clustering.server_groups*

| | |
|---|---|
| Description | A list of group names for the server used when configuring load balancing and replication policies. |
| Valid values | causal_clustering.server_groups, a ',' separated list with elements of type 'a string identifying a Server Group'. |
| Default value | |

## `causal_clustering.state_machine_apply_max_batch_size`

*Table 107. causal_clustering.state_machine_apply_max_batch_size*

| | |
|---|---|
| Description | The maximum number of operations to be batched during applications of operations in the state machines. |
| Valid values | causal_clustering.state_machine_apply_max_batch_size, an integer |
| Default value | `16` |

## causal_clustering.state_machine_flush_window_size

*Table 108. causal_clustering.state_machine_flush_window_size*

| | |
|---|---|
| **Description** | The number of operations to be processed before the state machines flush to disk. |
| **Valid values** | causal_clustering.state_machine_flush_window_size, an integer |
| **Default value** | `4096` |

## causal_clustering.status_throughput_window

*Table 109. causal_clustering.status_throughput_window*

| | |
|---|---|
| **Description** | Sampling window for throughput estimate reported in the status endpoint. |
| **Valid values** | causal_clustering.status_throughput_window, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) which is in the range `1s` to `5m` |
| **Default value** | `5s` |

## causal_clustering.store_copy_chunk_size

*Table 110. causal_clustering.store_copy_chunk_size*

| | |
|---|---|
| **Description** | Store copy chunk size. |
| **Valid values** | causal_clustering.store_copy_chunk_size, an integer which is in the range `4096` to `1048576` |
| **Default value** | `32768` |

## causal_clustering.store_copy_max_retry_time_per_request

*Table 111. causal_clustering.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** | causal_clustering.store_copy_max_retry_time_per_request, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| **Default value** | `20m` |

## causal_clustering.store_copy_parallelism

*Table 112. causal_clustering.store_copy_parallelism*

| | |
|---|---|
| **Description** | Limits amount of global threads for store copy. |
| **Valid values** | causal_clustering.store_copy_parallelism, an integer which is minimum `1` |
| **Default value** | `8` |

## causal_clustering.transaction_advertised_address

*Table 113. causal_clustering.transaction_advertised_address*

| | |
|---|---|
| Description | Advertised hostname/IP address and port for the transaction shipping server. |
| Valid values | causal_clustering.transaction_advertised_address, a socket address in the format 'hostname:port', 'hostname' or ':port'. If missing port or hostname it is acquired from dbms.default_advertised_address. |
| Default value | `:6000` |

## causal_clustering.transaction_listen_address

*Table 114. causal_clustering.transaction_listen_address*

| | |
|---|---|
| Description | Network interface and port for the transaction shipping server to listen on. Please 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 | causal_clustering.transaction_listen_address, a socket address in the format 'hostname:port', 'hostname' or ':port'. If missing port or hostname it is acquired from dbms.default_listen_address |
| Default value | `:6000` |

## causal_clustering.unknown_address_logging_throttle

*Table 115. causal_clustering.unknown_address_logging_throttle*

| | |
|---|---|
| Description | Throttle limit for logging unknown cluster member address. |
| Valid values | causal_clustering.unknown_address_logging_throttle, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `10s` |

## causal_clustering.upstream_selection_strategy

*Table 116. causal_clustering.upstream_selection_strategy*

| | |
|---|---|
| Description | An ordered list in descending preference of the strategy which read replicas use to choose the upstream server from which to pull transactional updates. |
| Valid values | causal_clustering.upstream_selection_strategy, a ',' separated list with elements of type 'a string'. |
| Default value | `default` |

## causal_clustering.user_defined_upstream_strategy

*Table 117. causal_clustering.user_defined_upstream_strategy*

| | |
|---|---|
| Description | Configuration of a user-defined upstream selection strategy. The user-defined strategy is used if the list of strategies (`causal_clustering.upstream_selection_strategy`) includes the value `user_defined`. |
| Valid values | causal_clustering.user_defined_upstream_strategy, a string |
| Default value | |

## 7.10.4. Fabric settings

### `fabric.database.name`

*Table 118. fabric.database.name*

| | |
|---|---|
| **Description** | Name of the Fabric database. Only one Fabric database is currently supported per Neo4j instance. |
| **Valid values** | fabric.database.name, 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' |

### `fabric.driver.api`

*Table 119. fabric.driver.api*

| | |
|---|---|
| **Description** | Determines which driver API will be used. ASYNC must be used when the remote instance is 3.5. |
| **Valid values** | fabric.driver.api, one of [RX, ASYNC] |
| **Default value** | `RX` |

### `fabric.driver.connection.connect_timeout`

*Table 120. fabric.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** | fabric.driver.connection.connect_timeout, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| **Default value** | `5s` |

### `fabric.driver.connection.max_lifetime`

*Table 121. fabric.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** | fabric.driver.connection.max_lifetime, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| **Default value** | `1h` |

### `fabric.driver.connection.pool.acquisition_timeout`

*Table 122. fabric.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 maximum connection pool size has been reached. Error is raised when connection can't be acquired within configured time. Negative values are allowed and result in unlimited acquisition timeout. Value of 0 is allowed and results in no timeout and immediate failure when connection is unavailable. |
| Valid values | fabric.driver.connection.pool.acquisition_timeout, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `1m` |

## `fabric.driver.connection.pool.idle_test`

*Table 123. fabric.driver.connection.pool.idle_test*

| | |
|---|---|
| Description | Pooled connections that have been idle in the pool for longer than this timeout will be tested before they are used again, to ensure they are still alive. If this option is set too low, an additional network call will be incurred when acquiring a connection, which causes a performance hit. If this is set high, no longer live connections might be used which might lead to errors. Hence, this parameter tunes a balance between the likelihood of experiencing connection problems and performance Normally, this parameter should not need tuning. Value 0 means connections will always be tested for validity. |
| Valid values | fabric.driver.connection.pool.idle_test, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `No connection liveliness check is done by default.` |

## `fabric.driver.connection.pool.max_size`

*Table 124. fabric.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. |
| Valid values | fabric.driver.connection.pool.max_size, an integer |
| Default value | `Unlimited` |

## `fabric.driver.logging.level`

*Table 125. fabric.driver.logging.level*

| | |
|---|---|
| Description | Sets level for driver internal logging. |
| Valid values | fabric.driver.logging.level, one of [DEBUG, INFO, WARN, ERROR, NONE] |
| Default value | `Value of dbms.logs.debug.level` |

## `fabric.graph.<graph ID>.database`

*Table 126. fabric.graph.<graph ID>.database*

| | |
|---|---|
| Description | Name of the database associated to the Fabric graph. |
| Valid values | fabric.graph.<graph ID>.database, a string |
| Default value | `The default database on the target DBMS. Typically 'Neo4j'` |

## `fabric.graph.<graph ID>.driver.api`

*Table 127. fabric.graph.<graph ID>.driver.api*

| | |
|---|---|
| Description | Determines which driver API will be used. ASYNC must be used when the remote instance is 3.5 This setting can be used as a graph-specific override of the global setting 'fabric.driver.api' |
| Valid values | fabric.graph.<graph ID>.driver.api, one of [RX, ASYNC] |

## `fabric.graph.<graph ID>.driver.connection.connect_timeout`

*Table 128. fabric.graph.<graph ID>.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. This setting can be used as a graph-specific override of the global setting 'fabric.driver.connection.connect_timeout' |
| Valid values | fabric.graph.<graph ID>.driver.connection.connect_timeout, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |

## `fabric.graph.<graph ID>.driver.connection.max_lifetime`

*Table 129. fabric.graph.<graph ID>.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. This setting can be used as a graph-specific override of the global setting 'fabric.driver.connection.max_lifetime' |
| Valid values | fabric.graph.<graph ID>.driver.connection.max_lifetime, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |

## `fabric.graph.<graph ID>.driver.connection.pool.acquisition_timeout`

*Table 130. fabric.graph.<graph ID>.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 maximum connection pool size has been reached. Error is raised when connection can't be acquired within configured time. Negative values are allowed and result in unlimited acquisition timeout. Value of 0 is allowed and results in no timeout and immediate failure when connection is unavailable. This setting can be used as a graph-specific override of the global setting 'fabric.driver.connection.pool.acquisition_timeout' |
| Valid values | fabric.graph.<graph ID>.driver.connection.pool.acquisition_timeout, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |

## fabric.graph.<graph ID>.driver.connection.pool.idle_test

*Table 131. fabric.graph.<graph ID>.driver.connection.pool.idle_test*

| | |
|---|---|
| Description | Pooled connections that have been idle in the pool for longer than this timeout will be tested before they are used again, to ensure they are still alive. If this option is set too low, an additional network call will be incurred when acquiring a connection, which causes a performance hit. If this is set high, no longer live connections might be used which might lead to errors. Hence, this parameter tunes a balance between the likelihood of experiencing connection problems and performance Normally, this parameter should not need tuning. Value 0 means connections will always be tested for validity. This setting can be used as a graph-specific override of the global setting 'fabric.driver.connection.pool.idle_test' |
| Valid values | fabric.graph.<graph ID>.driver.connection.pool.idle_test, a duration (Valid units are: `ns`, `µ s`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |

## fabric.graph.<graph ID>.driver.connection.pool.max_size

*Table 132. fabric.graph.<graph ID>.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. This setting can be used as a graph-specific override of the global setting 'fabric.driver.connection.pool.max_size' |
| Valid values | fabric.graph.<graph ID>.driver.connection.pool.max_size, an integer |

## fabric.graph.<graph ID>.driver.logging.leaked_sessions

*Table 133. fabric.graph.<graph ID>.driver.logging.leaked_sessions*

| | |
|---|---|
| Description | Enables logging of leaked driver session. |
| Valid values | fabric.graph.<graph ID>.driver.logging.leaked_sessions, a boolean |

## fabric.graph.<graph ID>.driver.logging.level

*Table 134. fabric.graph.<graph ID>.driver.logging.level*

| | |
|---|---|
| Description | Sets level for driver internal logging. This setting can be used as a graph-specific override of the global setting 'fabric.driver.logging.level' |
| Valid values | fabric.graph.<graph ID>.driver.logging.level, one of [DEBUG, INFO, WARN, ERROR, NONE] |

## fabric.graph.<graph ID>.driver.ssl_enabled

*Table 135. fabric.graph.<graph ID>.driver.ssl_enabled*

| | |
|---|---|
| Description | SSL for Fabric drivers is configured using 'fabric' SSL policy.This setting can be used to instruct the driver not to use SSL even though 'fabric' SSL policy is configured.The driver will use SSL if 'fabric' SSL policy is configured and this setting is set to 'true' |
| Valid values | fabric.graph.<graph ID>.driver.ssl_enabled, a boolean |
| Default value | `true` |

## fabric.graph.<graph ID>.name

*Table 136. fabric.graph.<graph ID>.name*

| | |
|---|---|
| Description | Name assigned to the Fabric graph. The name can be used in Fabric queries. |
| Valid values | fabric.graph.<graph ID>.name, A valid graph name. Containing only alphabetic characters, numbers, dots and dashes, with a length between 3 and 63 characters. It should be starting with an alphabetic character. The name 'graph' is reserved. |

*Table 137. fabric.graph.<graph ID>.uri*

| | |
|---|---|
| Description | URI of the Neo4j DBMS hosting the database associated to the Fabric graph. Example: neo4j://somewhere:7687 A comma separated list of URIs is acceptable. This is useful when the Fabric graph is hosted on a cluster and more that one bootstrap address needs to be provided in order to avoid a single point of failure. The provided addresses will be considered as an initial source of a routing table. Example: neo4j://core-1:1111,neo4j://core-2:2222. |
| Valid values | fabric.graph.<graph ID>.uri, a ',' separated list with elements of type 'a URI'. |

## fabric.routing.servers

Dynamic

*Table 138. fabric.routing.servers*

| | |
|---|---|
| Description | A comma-separated list of Fabric instances that form a routing group. A driver will route transactions to available routing group members. A Fabric instance is represented by its Bolt connector address. |
| Valid values | fabric.routing.servers, a ',' separated list with elements of type 'a socket address in the format 'hostname:port', 'hostname' or ':port''. |

## fabric.routing.ttl

*Table 139. fabric.routing.ttl*

| | |
|---|---|
| Description | The time to live (TTL) of a routing table for fabric routing group. |
| Valid values | fabric.routing.ttl, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `1m` |

## fabric.stream.buffer.low_watermark

*Table 140. fabric.stream.buffer.low_watermark*

| | |
|---|---|
| Description | Number of records in prefetching buffer that will trigger prefetching again. This is strongly related to fabric.stream.buffer.size |
| Valid values | fabric.stream.buffer.low_watermark, an integer which is minimum `0` |
| Default value | `300` |

## fabric.stream.buffer.size

*Table 141. fabric.stream.buffer.size*

| | |
|---|---|
| **Description** | Maximal size of a buffer used for pre-fetching result records of remote queries. To compensate for latency to remote databases, the Fabric execution engine pre-fetches records needed for local executions. This limit is enforced per fabric query. If a fabric query uses multiple remote stream at the same time, this setting represents the maximal number of pre-fetched records counted together for all such remote streams. |
| **Valid values** | fabric.stream.buffer.size, an integer which is minimum `1` |
| **Default value** | `1000` |

### `fabric.stream.concurrency`

*Table 142. fabric.stream.concurrency*

| | |
|---|---|
| **Description** | Maximal concurrency within Fabric queries. Limits the number of iterations of each subquery that are executed concurrently. Higher concurrency may consume more memory and network resources simultaneously, while lower concurrency may force sequential execution, requiring more time. |
| **Valid values** | fabric.stream.concurrency, an integer which is minimum `1` |
| **Default value** | `The number of remote graphs` |

## 7.10.5. Connection settings

### `dbms.default_advertised_address`

*Table 143. dbms.default_advertised_address*

| | |
|---|---|
| **Description** | Default hostname or IP address the server uses to advertise itself. |
| **Valid values** | dbms.default_advertised_address, a socket address in the format 'hostname:port', 'hostname' or ':port' which has no specified port |
| **Default value** | `localhost` |

### `dbms.default_listen_address`

*Table 144. dbms.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** | dbms.default_listen_address, a socket address in the format 'hostname:port', 'hostname' or ':port' which has no specified port |
| **Default value** | `localhost` |

### `dbms.http_enabled_modules`

*Table 145. dbms.http_enabled_modules*

| | |
|---|---|
| Description | Defines the set of modules loaded into the Neo4j web server. Options include TRANSACTIONAL_ENDPOINTS, BROWSER, UNMANAGED_EXTENSIONS and ENTERPRISE_MANAGEMENT_ENDPOINTS (if applicable). |
| Valid values | dbms.http_enabled_modules, a ',' separated set with elements of type 'one of [TRANSACTIONAL_ENDPOINTS, UNMANAGED_EXTENSIONS, BROWSER, ENTERPRISE_MANAGEMENT_ENDPOINTS]'. |
| Default value | `TRANSACTIONAL_ENDPOINTS,UNMANAGED_EXTENSIONS,BROWSER,ENTERPRISE_MANAGEMENT_ENDPOINTS` |

## dbms.routing.advertised_address

*Table 146. dbms.routing.advertised_address*

| | |
|---|---|
| Description | The advertised address for the intra-cluster routing connector. |
| Valid values | dbms.routing.advertised_address, a socket address in the format 'hostname:port', 'hostname' or ':port'. If missing port or hostname it is acquired from dbms.default_advertised_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 | dbms.routing.client_side.enforce_for_domains, a ',' separated set with elements of type '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 will rely on server-side routing (which must be enabled for proper operation, i.e. `dbms.routing.enabled=true`). Can be overridden by `dbms.routing.client_side.enforce_for_domains`. |
| Valid values | dbms.routing.default_router, one of [SERVER, CLIENT] |
| Default value | `CLIENT` |

## dbms.routing.driver.api

*Table 149. dbms.routing.driver.api*

| | |
|---|---|
| Description | Determines which driver API will be used. `ASYNC` must be used when the remote instance is 3.5, but is only retained for backwards-compatibility reasons. `RX` should be used in all other cases. |

| | |
|---|---|
| **Valid values** | dbms.routing.driver.api, one of [RX, ASYNC] |
| **Default value** | `RX` |

## `dbms.routing.driver.connection.connect_timeout`

*Table 150. 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** | dbms.routing.driver.connection.connect_timeout, 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 151. 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** | dbms.routing.driver.connection.max_lifetime, 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 152. 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 maximum connection pool size has been reached. Error is raised when connection can't be acquired within configured time. Negative values are allowed and result in unlimited acquisition timeout. Value of 0 is allowed and results in no timeout and immediate failure when connection is unavailable. |
| **Valid values** | dbms.routing.driver.connection.pool.acquisition_timeout, 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 153. 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 before they are used again, to ensure they are still alive. If this option is set too low, an additional network call will be incurred when acquiring a connection, which causes a performance hit. If this is set high, no longer live connections might be used which might lead to errors. Hence, this parameter tunes a balance between the likelihood of experiencing connection problems and performance Normally, this parameter should not need tuning. Value 0 means connections will always be tested for validity. |
| Valid values | dbms.routing.driver.connection.pool.idle_test, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `No connection liveliness check is done by default.` |

## dbms.routing.driver.connection.pool.max_size

*Table 154. 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. |
| Valid values | dbms.routing.driver.connection.pool.max_size, an integer |
| Default value | `Unlimited` |

## dbms.routing.driver.logging.level

*Table 155. dbms.routing.driver.logging.level*

| | |
|---|---|
| Description | Sets level for driver internal logging. |
| Valid values | dbms.routing.driver.logging.level, one of [DEBUG, INFO, WARN, ERROR, NONE] |
| Default value | `Value of dbms.logs.debug.level` |

## dbms.routing.enabled

*Table 156. 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 can't be satisfied by the first member (e.g. write requests received by a non-leader). |
| Valid values | dbms.routing.enabled, a boolean |
| Default value | `false` |

## dbms.routing.listen_address

*Table 157. dbms.routing.listen_address*

| | |
|---|---|
| Description | The address the routing connector should bind to. |
| Valid values | dbms.routing.listen_address, a socket address in the format 'hostname:port', 'hostname' or ':port'. If missing port or hostname it is acquired from dbms.default_listen_address |

| | |
|---|---|
| Default value | `:7688` |

## `dbms.routing_ttl`

*Table 158. dbms.routing_ttl*

| | |
|---|---|
| Description | How long callers should cache the response of the routing procedure `dbms.routing.getRoutingTable()` |
| Valid values | dbms.routing_ttl, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) which is minimum `1s` |
| Default value | `5m` |

## `dbms.connector.bolt.advertised_address`

*Table 159. dbms.connector.bolt.advertised_address*

| | |
|---|---|
| Description | Advertised address for this connector. |
| Valid values | dbms.connector.bolt.advertised_address, a socket address in the format 'hostname:port', 'hostname' or ':port'. If missing port or hostname it is acquired from dbms.default_advertised_address |
| Default value | `:7687` |

## `dbms.connector.bolt.connection_keep_alive`

*Table 160. dbms.connector.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 | dbms.connector.bolt.connection_keep_alive, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) which is minimum `1ms` |
| Default value | `1m` |

## `dbms.connector.bolt.connection_keep_alive_for_requests`

*Table 161. dbms.connector.bolt.connection_keep_alive_for_requests*

| | |
|---|---|
| Description | The type of messages to enable keep-alive messages for (ALL, STREAMING or OFF) |
| Valid values | dbms.connector.bolt.connection_keep_alive_for_requests, one of [ALL, STREAMING, OFF] |
| Default value | `STREAMING` |

## `dbms.connector.bolt.connection_keep_alive_probes`

*Table 162. dbms.connector.bolt.connection_keep_alive_probes*

| | |
|---|---|
| Description | The total amount of probes to be missed before a connection is considered stale.The minimum for this value is 1. |

| | |
|---|---|
| Valid values | dbms.connector.bolt.connection_keep_alive_probes, an integer which is minimum `1` |
| Default value | `2` |

## dbms.connector.bolt.connection_keep_alive_streaming_scheduling_interval

*Table 163. dbms.connector.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 | dbms.connector.bolt.connection_keep_alive_streaming_scheduling_interval, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) which is minimum `0s` |
| Default value | `1m` |

## dbms.connector.bolt.enabled

*Table 164. dbms.connector.bolt.enabled*

| | |
|---|---|
| Description | Enable the bolt connector. |
| Valid values | dbms.connector.bolt.enabled, a boolean |
| Default value | `true` |

## dbms.connector.bolt.listen_address

*Table 165. dbms.connector.bolt.listen_address*

| | |
|---|---|
| Description | Address the connector should bind to. |
| Valid values | dbms.connector.bolt.listen_address, a socket address in the format 'hostname:port', 'hostname' or ':port'. If missing port or hostname it is acquired from dbms.default_listen_address |
| Default value | `:7687` |

## dbms.connector.bolt.ocsp_stapling_enabled

*Table 166. dbms.connector.bolt.ocsp_stapling_enabled*

| | |
|---|---|
| Description | Enable server OCSP stapling for bolt and http connectors. |
| Valid values | dbms.connector.bolt.ocsp_stapling_enabled, a boolean |
| Default value | `false` |

## dbms.connector.bolt.thread_pool_keep_alive

*Table 167. dbms.connector.bolt.thread_pool_keep_alive*

| | |
|---|---|
| Description | The maximum time an idle thread in the thread pool bound to this connector will wait for new tasks. |

| | |
|---|---|
| Valid values | dbms.connector.bolt.thread_pool_keep_alive, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `5m` |

## dbms.connector.bolt.thread_pool_max_size

*Table 168. dbms.connector.bolt.thread_pool_max_size*

| | |
|---|---|
| Description | The maximum number of threads allowed in the thread pool bound to this connector. |
| Valid values | dbms.connector.bolt.thread_pool_max_size, an integer |
| Default value | `400` |

## dbms.connector.bolt.thread_pool_min_size

*Table 169. dbms.connector.bolt.thread_pool_min_size*

| | |
|---|---|
| Description | The number of threads to keep in the thread pool bound to this connector, even if they are idle. |
| Valid values | dbms.connector.bolt.thread_pool_min_size, an integer |
| Default value | `5` |

## dbms.connector.bolt.tls_level

*Table 170. dbms.connector.bolt.tls_level*

| | |
|---|---|
| Description | Encryption level to require this connector to use. |
| Valid values | dbms.connector.bolt.tls_level, one of [REQUIRED, OPTIONAL, DISABLED] |
| Default value | `DISABLED` |

## dbms.connector.bolt.unsupported_thread_pool_shutdown_wait_time

*Table 171. dbms.connector.bolt.unsupported_thread_pool_shutdown_wait_time*

| | |
|---|---|
| Description | The maximum time to wait for the thread pool to finish processing its pending jobs and shutdown. |
| Valid values | dbms.connector.bolt.unsupported_thread_pool_shutdown_wait_time, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `5s` |

## dbms.connector.http.advertised_address

*Table 172. dbms.connector.http.advertised_address*

| | |
|---|---|
| Description | Advertised address for this connector. |

| | |
|---|---|
| Valid values | dbms.connector.http.advertised_address, a socket address in the format 'hostname:port', 'hostname' or ':port'. If missing port or hostname it is acquired from dbms.default_advertised_address |
| Default value | `:7474` |

## dbms.connector.http.enabled

*Table 173. dbms.connector.http.enabled*

| | |
|---|---|
| Description | Enable the http connector. |
| Valid values | dbms.connector.http.enabled, a boolean |
| Default value | `true` |

## dbms.connector.http.listen_address

*Table 174. dbms.connector.http.listen_address*

| | |
|---|---|
| Description | Address the connector should bind to. |
| Valid values | dbms.connector.http.listen_address, a socket address in the format 'hostname:port', 'hostname' or ':port'. If missing port or hostname it is acquired from dbms.default_listen_address |
| Default value | `:7474` |

## dbms.connector.https.advertised_address

*Table 175. dbms.connector.https.advertised_address*

| | |
|---|---|
| Description | Advertised address for this connector. |
| Valid values | dbms.connector.https.advertised_address, a socket address in the format 'hostname:port', 'hostname' or ':port'. If missing port or hostname it is acquired from dbms.default_advertised_address |
| Default value | `:7473` |

## dbms.connector.https.enabled

*Table 176. dbms.connector.https.enabled*

| | |
|---|---|
| Description | Enable the https connector. |
| Valid values | dbms.connector.https.enabled, a boolean |
| Default value | `false` |

## dbms.connector.https.listen_address

*Table 177. dbms.connector.https.listen_address*

| | |
|---|---|
| Description | Address the connector should bind to. |

| | |
|---|---|
| Valid values | dbms.connector.https.listen_address, a socket address in the format 'hostname:port', 'hostname' or ':port'. If missing port or hostname it is acquired from dbms.default_listen_address |
| Default value | `:7473` |

## 7.10.6. Cypher settings

### `cypher.default_language_version`

*Table 178. cypher.default_language_version*

| | |
|---|---|
| Description | Set this to specify the default parser (language version). |
| Valid values | cypher.default_language_version, one of [default, 3.5, 4.3, 4.4] |
| Default value | `default` |

### `cypher.forbid_exhaustive_shortestpath`

*Table 179. 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 | cypher.forbid_exhaustive_shortestpath, a boolean |
| Default value | `false` |

### `cypher.forbid_shortestpath_common_nodes`

*Table 180. 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 | cypher.forbid_shortestpath_common_nodes, a boolean |
| Default value | `true` |

## `cypher.hints_error`

*Table 181. 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 | cypher.hints_error, a boolean |
| Default value | `false` |

## `cypher.lenient_create_relationship`

*Table 182. 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 | cypher.lenient_create_relationship, a boolean |
| Default value | `false` |

## `cypher.min_replan_interval`

*Table 183. 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 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 | cypher.min_replan_interval, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `10s` |

## `cypher.planner`

*Table 184. cypher.planner*

| | |
|---|---|
| Description | Set this to specify the default planner for the default language version. |
| Valid values | cypher.planner, one of [DEFAULT, COST] |
| Default value | `DEFAULT` |

## `cypher.statistics_divergence_threshold`

*Table 185. cypher.statistics_divergence_threshold*

| | |
|---|---|
| Description | The threshold for statistics above which a plan is considered stale.<br><br>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)`.<br><br>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.<br><br>This interval is defined by `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 | cypher.statistics_divergence_threshold, a double which is in the range `0.0` to `1.0` |
| Default value | `0.75` |

## 7.10.7. Database settings

### `dbms.filewatcher.enabled`

*Table 186. dbms.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 | dbms.filewatcher.enabled, a boolean |
| Default value | `true` |

### `dbms.record_format`

*Table 187. dbms.record_format*

| | |
|---|---|
| Description | Database record format. Valid values are blank(no value, default), `standard`, `aligned`, or `high_limit`. Specifying a value will force new databases to that format and existing databases to migrate if `dbms.allow_upgrade =true` is specified. 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` format is available for Enterprise Edition only. It is required if you have a graph that is larger than 34 billion nodes, 34 billion relationships, or 68 billion properties. A change of the record format is irreversible. Certain operations may suffer from a performance penalty of up to 10%, which is why this format is not switched on by default. However, if you want to change the configured record format value, you must also set `dbms.allow_upgrade =true`, because the setting implies a one-way store format migration. |
| Valid values | dbms.record_format, a string |
| Default value | `aligned` for new databases. Existing databases stay on their current format. |

### `dbms.relationship_grouping_threshold`

*Table 188. dbms.relationship_grouping_threshold*

| | |
|---|---|
| Description | Relationship count threshold for considering a node to be dense. |
| Valid values | dbms.relationship_grouping_threshold, an integer which is minimum `1` |
| Default value | `50` |

## `dbms.store.files.preallocate`

*Table 189. dbms.store.files.preallocate*

| | |
|---|---|
| Description | Specify if Neo4j should try to preallocate store files as they grow. |
| Valid values | dbms.store.files.preallocate, a boolean |
| Default value | `true` |

## `db.temporal.timezone`

*Table 190. 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 | db.temporal.timezone, a string describing a timezone, either described by offset (e.g. `+02:00`) or by name (e.g. `Europe/Stockholm`) |
| Default value | `Z` |

## `dbms.track_query_cpu_time`

Enterprise Edition Dynamic

*Table 191. dbms.track_query_cpu_time*

| | |
|---|---|
| Description | Enables or disables tracking of how much time a query spends actively executing on the CPU. Calling `dbms.listQueries` will display the time. This can also be logged in the query log by using `dbms.logs.query.time_logging_enabled`. . |
| Valid values | dbms.track_query_cpu_time, a boolean |
| Default value | `false` |

## `dbms.track_query_allocation`

Dynamic

*Table 192. dbms.track_query_allocation*

| | |
|---|---|
| Description | Enables or disables tracking of how many bytes are allocated by the execution of a query. If enabled, calling `dbms.listQueries` will display the allocated bytes. This can also be logged in the query log by using `dbms.logs.query.allocation_logging_enabled`. |
| Valid values | dbms.track_query_allocation, a boolean |
| Default value | `true` |

## 7.10.8. DBMS settings

### `dbms.backup.enabled`

*Table 193. dbms.backup.enabled*

| | |
|---|---|
| **Description** | Enable support for running online backups. |
| **Valid values** | dbms.backup.enabled, a boolean |
| **Default value** | `true` |

### `dbms.backup.incremental.strategy`

Dynamic

*Table 194. dbms.backup.incremental.strategy*

| | |
|---|---|
| **Description** | Strategy for incremental backup. START_TIME means that this server will send transactions until the time of when the backup started has been reached. UNBOUNDED will keep sending until all committed transactions have been sent, even if they where committed after the backup job started. |
| **Valid values** | dbms.backup.incremental.strategy, one of [UNBOUNDED, START_TIME] |
| **Default value** | `UNBOUNDED` |

### `dbms.backup.listen_address`

*Table 195. dbms.backup.listen_address*

| | |
|---|---|
| **Description** | Network interface and port for the backup server to listen on. |
| **Valid values** | dbms.backup.listen_address, a socket address in the format 'hostname:port', 'hostname' or ':port' |
| **Default value** | `127.0.0.1:6362` |

### `dbms.config.strict_validation`

*Table 196. dbms.config.strict_validation*

| | |
|---|---|
| **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). |
| **Valid values** | dbms.config.strict_validation, a boolean |
| **Default value** | `false` |

### `dbms.databases.default_to_read_only`

Dynamic

*Table 197. dbms.databases.default_to_read_only*

| | |
|---|---|
| Description | Whether or not any database on this instance are read_only by default. If false, individual databases may be marked as read_only using dbms.database.read_only. If true, individual databases may be marked as writable using dbms.databases.writable. |
| Valid values | dbms.databases.default_to_read_only, a boolean |
| Default value | `false` |

## dbms.databases.read_only

Dynamic

*Table 198. dbms.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 dbms.databases.default_to_read_only. |
| Valid values | dbms.databases.read_only, a ',' separated set with elements of type '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''. which Value 'system' can't be included in read only databases collection! |
| Default value | |

## dbms.databases.writable

Dynamic

*Table 199. dbms.databases.writable*

| | |
|---|---|
| Description | List of databases for which to allow write queries. Databases not included in this list will allow write queries anyway, unless dbms.databases.default_to_read_only is set to true. |
| Valid values | dbms.databases.writable, a ',' separated set with elements of type '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 | |

## dbms.dynamic.setting.allowlist

*Table 200. dbms.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 | dbms.dynamic.setting.allowlist, a ',' separated list with elements of type 'a string'. |
| Default value | * |

## dbms.dynamic.setting.whitelist

Deprecated in 4.2

*Table 201. dbms.dynamic.setting.whitelist*

| | |
|---|---|
| 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 | dbms.dynamic.setting.whitelist, a ',' separated list with elements of type 'a string'. |
| Replaced by | dbms.dynamic.setting.allowlist |
| Default value | * |

## dbms.jvm.additional

*Table 202. dbms.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 | dbms.jvm.additional, one or more jvm arguments |

## dbms.panic.shutdown_on_panic

Enterprise Edition

*Table 203. dbms.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. This feature is available in Neo4j Enterprise Edition. |
| Valid values | dbms.panic.shutdown_on_panic, a boolean |
| Default value | `false` except for Neo4j Enterprise Edition deployments running on Kubernetes where it is `true`. |

## dbms.threads.worker_count

*Table 204. dbms.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 amount 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, with the aim of getting more throughput from the database. Your OS might enforce a lower limit than the maximum value specified here. |
| Valid values | dbms.threads.worker_count, an integer which is in the range `1` to `44738` |
| Default value | `Number of available processors, or 500 for machines which have more than 500 processors.` |

## dbms.unmanaged_extension_classes

*Table 205. dbms.unmanaged_extension_classes*

| | |
|---|---|
| Description | Comma-separated list of <classname>=<mount point> for unmanaged extensions. |

| | |
|---|---|
| Valid values | dbms.unmanaged_extension_classes, a ',' separated list with elements of type '<classname>=<mount point> string'. |
| Default value | |

## dbms.upgrade_max_processors

Dynamic

*Table 206. dbms.upgrade_max_processors*

| | |
|---|---|
| Description | Max number of processors used when upgrading the store. Defaults to the number of processors available to 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 shouldn't be greater than the number of available processors. |
| Valid values | dbms.upgrade_max_processors, an integer which is minimum `0` |
| Default value | `0` |

## dbms.windows_service_name

*Table 207. dbms.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 individual installation. |
| Valid values | dbms.windows_service_name, a string |
| Default value | `neo4j` |

## dbms.default_database

*Table 208. dbms.default_database*

| | |
|---|---|
| Description | Name of the default database (aliases are not supported). |
| Valid values | dbms.default_database, 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` |

## dbms.db.timezone

*Table 209. dbms.db.timezone*

| | |
|---|---|
| Description | Database timezone. Among other things, this setting influences which timezone the logs and monitoring procedures use. |
| Valid values | dbms.db.timezone, one of [UTC, SYSTEM] |
| Default value | `UTC` |

## dbms.max_databases

*Table 210. dbms.max_databases*

| | |
|---|---|
| Description | The maximum number of databases. |
| Valid values | dbms.max_databases, a long which is minimum `2` |
| Default value | `100` |

## `dbms.mode`

*Table 211. dbms.mode*

| | |
|---|---|
| Description | Configure the operating mode of the database — 'SINGLE' for stand-alone operation, 'CORE' for operating as a core member of a Causal Cluster, or 'READ_REPLICA' for operating as a read replica member of a Causal Cluster. Only SINGLE mode is allowed in Community. |
| Valid values | dbms.mode, one of [SINGLE, CORE, READ_REPLICA] |
| Default value | `SINGLE` |

## `dbms.read_only`

Deprecated

*Table 212. dbms.read_only*

| | |
|---|---|
| Description | Only allow read operations from this Neo4j instance. This mode still requires write access to the directory for lock purposes. |
| Valid values | dbms.read_only, a boolean |
| Default value | `false` |
| Replaced by | dbms.databases.default_to_read_only, dbms.databases.read_only, dbms.databases.writable |

## `dbms.clustering.enable`

Deprecated

*Table 213. dbms.clustering.enable*

| | |
|---|---|
| Description | Enable discovery service and a catchup server to be started on an Enterprise Standalone Instance 'dbms.mode=SINGLE', and with that allow for Read Replicas to connect and pull transaction from it. When 'dbms.mode' is clustered (CORE, READ_REPLICA) this setting is not recognized. |
| Valid values | dbms.clustering.enable, a boolean |
| Default value | `false` |

## `dbms.allow_single_automatic_upgrade`

Dynamic

*Table 214. dbms.allow_single_automatic_upgrade*

| | |
|---|---|
| Description | Whether to allow a system graph upgrade to happen automatically in single instance mode (dbms.mode=SINGLE). Default is true. In clustering environments no automatic upgrade will happen (dbms.mode=CORE or dbms.mode=READ_REPLICA). If set to false, or when in a clustering environment, it is necessary to call the procedure `dbms.upgrade()` to complete the upgrade. |
| Valid values | dbms.allow_single_automatic_upgrade, a boolean |
| Default value | `true` |

## dbms.allow_upgrade

Dynamic

*Table 215. dbms.allow_upgrade*

| | |
|---|---|
| Description | Whether to allow a store upgrade in case the current version of the database starts against an older version of the store. |
| Valid values | dbms.allow_upgrade, a boolean |
| Default value | `false` |

## dbms.reconciler.max_backoff

*Table 216. dbms.reconciler.max_backoff*

| | |
|---|---|
| Description | Defines the maximum amount of time to wait before retrying after the dbms fails to reconcile a database to its desired state. |
| Valid values | dbms.reconciler.max_backoff, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) which is minimum `1m` |
| Default value | `1h` |

## dbms.reconciler.max_parallelism

*Table 217. dbms.reconciler.max_parallelism*

| | |
|---|---|
| Description | Defines the level of parallelism employed by the reconciler. By default the parallelism equals the number of available processors or 8 (whichever is smaller). If configured as 0, the parallelism of the reconciler will be unbounded. |
| Valid values | dbms.reconciler.max_parallelism, an integer which is minimum `0` |
| Default value | `8` |

## dbms.reconciler.may_retry

*Table 218. dbms.reconciler.may_retry*

| | |
|---|---|
| Description | Defines whether the dbms may retry reconciling a database to its desired state. |
| Valid values | dbms.reconciler.may_retry, a boolean |

| | |
|---|---|
| **Default value** | `false` |

## `dbms.reconciler.min_backoff`

*Table 219. dbms.reconciler.min_backoff*

| | |
|---|---|
| **Description** | Defines the minimum amount of time to wait before retrying after the dbms fails to reconcile a database to its desired state. |
| **Valid values** | dbms.reconciler.min_backoff, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) which is minimum `1s` |
| **Default value** | `2s` |

## `dbms.directories.cluster_state`

*Table 220. dbms.directories.cluster_state*

| | |
|---|---|
| **Description** | Directory to hold cluster state including Raft log. |
| **Valid values** | dbms.directories.cluster_state, a path. If relative it is resolved from dbms.directories.data |
| **Default value** | `cluster-state` |

## `dbms.directories.data`

*Table 221. dbms.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** | dbms.directories.data, a path. If relative it is resolved from dbms.directories.neo4j_home |
| **Default value** | `data` |

## `dbms.directories.dumps.root`

*Table 222. dbms.directories.dumps.root*

| | |
|---|---|
| **Description** | Root location where Neo4j will store database dumps optionally produced when dropping said databases. |
| **Valid values** | dbms.directories.dumps.root, a path. If relative it is resolved from dbms.directories.data |
| **Default value** | `dumps` |

## `dbms.directories.import`

*Table 223. dbms.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 | dbms.directories.import, a path. If relative it is resolved from dbms.directories.neo4j_home |

## `dbms.directories.lib`

*Table 224. dbms.directories.lib*

| | |
|---|---|
| Description | Path of the lib directory. |
| Valid values | dbms.directories.lib, a path. If relative it is resolved from dbms.directories.neo4j_home |
| Default value | `lib` |

## `dbms.directories.licenses`

*Table 225. dbms.directories.licenses*

| | |
|---|---|
| Description | Path of the licenses directory. |
| Valid values | dbms.directories.licenses, a path. If relative it is resolved from dbms.directories.neo4j_home |
| Default value | `licenses` |

## `dbms.directories.logs`

*Table 226. dbms.directories.logs*

| | |
|---|---|
| Description | Path of the logs directory. |
| Valid values | dbms.directories.logs, a path. If relative it is resolved from dbms.directories.neo4j_home |
| Default value | `logs` |

## `dbms.directories.metrics`

*Table 227. dbms.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 | dbms.directories.metrics, a path. If relative it is resolved from dbms.directories.neo4j_home |
| Default value | `metrics` |

## `dbms.directories.neo4j_home`

*Table 228. dbms.directories.neo4j_home*

| | |
|---|---|
| Description | Root relative to which directory settings are resolved. |
| Valid values | dbms.directories.neo4j_home, a path which is absolute |
| Default value | `Defaults to current working directory` |

### `dbms.directories.plugins`

*Table 229. dbms.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 | dbms.directories.plugins, a path. If relative it is resolved from dbms.directories.neo4j_home |
| Default value | `plugins` |

### `dbms.directories.run`

*Table 230. dbms.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 | dbms.directories.run, a path. If relative it is resolved from dbms.directories.neo4j_home |
| Default value | `run` |

### `dbms.directories.script.root`

*Table 231. dbms.directories.script.root*

| | |
|---|---|
| Description | Root location where Neo4j will store scripts for configured databases. |
| Valid values | dbms.directories.script.root, a path. If relative it is resolved from dbms.directories.data |
| Default value | `scripts` |

### `dbms.directories.transaction.logs.root`

*Table 232. dbms.directories.transaction.logs.root*

| | |
|---|---|
| Description | Root location where Neo4j will store transaction logs for configured databases. |
| Valid values | dbms.directories.transaction.logs.root, a path. If relative it is resolved from dbms.directories.data |
| Default value | `transactions` |

## 7.10.9. Import settings

### `dbms.import.csv.buffer_size`

*Table 233. dbms.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 | dbms.import.csv.buffer_size, a long which is minimum `1` |
| Default value | `2097152` |

### `dbms.import.csv.legacy_quote_escaping`

*Table 234. dbms.import.csv.legacy_quote_escaping*

| | |
|---|---|
| Description | Selects whether to conform to the standard https://tools.ietf.org/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 | dbms.import.csv.legacy_quote_escaping, a boolean |
| Default value | `true` |

## 7.10.10. Index settings

### `dbms.index.default_schema_provider`

Deprecated

*Table 235. dbms.index.default_schema_provider*

| | |
|---|---|
| Description | Index provider to use for newly created schema indexes. An index provider may store different value types in separate physical indexes. native-btree-1.0: All value types and arrays of all value types, even composite keys, are stored in one native index. lucene+native-3.0: Like native-btree-1.0 but single property strings are stored in Lucene. A native index has faster updates, less heap and CPU usage compared to a Lucene index. A native index has some limitations around key size and slower execution of CONTAINS and ENDS WITH string index queries, compared to a Lucene index. Deprecated: Which index provider to use will be a fully internal concern. |
| Valid values | dbms.index.default_schema_provider, a string |
| Default value | `native-btree-1.0` |

### `dbms.index.fulltext.default_analyzer`

*Table 236. dbms.index.fulltext.default_analyzer*

| | |
|---|---|
| Description | The name of the analyzer that the fulltext indexes should use by default. |
| Valid values | dbms.index.fulltext.default_analyzer, a string |
| Default value | `standard-no-stop-words` |

### `dbms.index.fulltext.eventually_consistent`

*Table 237. dbms.index.fulltext.eventually_consistent*

| | |
|---|---|
| **Description** | Whether or not fulltext indexes should be eventually consistent by default or not. |
| **Valid values** | dbms.index.fulltext.eventually_consistent, a boolean |
| **Default value** | `false` |

## `dbms.index.fulltext.eventually_consistent_index_update_queue_max_length`

*Table 238. dbms.index.fulltext.eventually_consistent_index_update_queue_max_length*

| | |
|---|---|
| **Description** | The eventually_consistent mode of the fulltext 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** | dbms.index.fulltext.eventually_consistent_index_update_queue_max_length, an integer which is in the range `1` to `50000000` |
| **Default value** | `10000` |

## `dbms.index_sampling.background_enabled`

*Table 239. dbms.index_sampling.background_enabled*

| | |
|---|---|
| **Description** | Enable or disable background index sampling. |
| **Valid values** | dbms.index_sampling.background_enabled, a boolean |
| **Default value** | `true` |

## `dbms.index_sampling.sample_size_limit`

*Table 240. dbms.index_sampling.sample_size_limit*

| | |
|---|---|
| **Description** | Index sampling chunk size limit. |
| **Valid values** | dbms.index_sampling.sample_size_limit, an integer which is in the range `1048576` to `2147483647` |
| **Default value** | `8388608` |

## `dbms.index_sampling.update_percentage`

*Table 241. dbms.index_sampling.update_percentage*

| | |
|---|---|
| **Description** | Percentage of index updates of total index size required before sampling of a given index is triggered. |
| **Valid values** | dbms.index_sampling.update_percentage, an integer which is minimum `0` |
| **Default value** | `5` |

## `dbms.index_searcher_cache_size`

Deprecated

*Table 242. dbms.index_searcher_cache_size*

| | |
|---|---|
| Description | The maximum number of open Lucene index searchers. |
| Valid values | dbms.index_searcher_cache_size, an integer which is minimum `1` |
| Default value | `2147483647` |

## 7.10.11. Logging settings

### `db.logs.query.obfuscate_errors`

Introduced in 4.4.48 Dynamic

*Table 243. db.logs.query.obfuscate_errors*

| | |
|---|---|
| Description | Obfuscates all sensitive information of errors before writing to the log. Note that only some errors are obfuscatable and that other errors will be missing error message and status description, while maintaining GQLSTATUS code. |
| Valid values | A boolean. |
| Default value | `false` |

### `dbms.logs.debug.format`

*Table 244. dbms.logs.debug.format*

| | |
|---|---|
| Description | Log format to use for debug log. |
| Valid values | dbms.logs.debug.format, one of [PLAIN, JSON]. If unset the value is inherited from dbms.logs.default_format |

### `dbms.logs.debug.level`

Dynamic

*Table 245. dbms.logs.debug.level*

| | |
|---|---|
| Description | Debug log level threshold. |
| Valid values | dbms.logs.debug.level, one of [DEBUG, INFO, WARN, ERROR, NONE] |
| Default value | `INFO` |

### `dbms.logs.debug.path`

*Table 246. dbms.logs.debug.path*

| | |
|---|---|
| Description | Path to the debug log file. |

| | |
|---|---|
| Valid values | dbms.logs.debug.path, a path. If relative it is resolved from dbms.directories.logs |
| Default value | `debug.log` |

## dbms.logs.debug.rotation.delay

Deprecated

*Table 247. dbms.logs.debug.rotation.delay*

| | |
|---|---|
| Description | Minimum time interval after last rotation of the debug log before it may be rotated again. |
| Valid values | dbms.logs.debug.rotation.delay, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `5m` |

## dbms.logs.debug.rotation.keep_number

*Table 248. dbms.logs.debug.rotation.keep_number*

| | |
|---|---|
| Description | Maximum number of history files for the debug log. |
| Valid values | dbms.logs.debug.rotation.keep_number, an integer which is minimum `1` |
| Default value | `7` |

## dbms.logs.debug.rotation.size

*Table 249. dbms.logs.debug.rotation.size*

| | |
|---|---|
| Description | Threshold for rotation of the debug log. |
| Valid values | dbms.logs.debug.rotation.size, 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`, `PiB`, `PB`, `EiB`, `EB`) which is in the range `0B` to `8388608.00TiB` |
| Default value | `20.00MiB` |

## dbms.logs.default_format

*Table 250. dbms.logs.default_format*

| | |
|---|---|
| Description | Default log format. Will apply to all logs unless overridden. |
| Valid values | dbms.logs.default_format, one of [PLAIN, JSON] |
| Default value | `PLAIN` |

## dbms.logs.gc.enabled

*Table 251. dbms.logs.gc.enabled*

| | |
|---|---|
| Description | Enable GC Logging. |
| Valid values | dbms.logs.gc.enabled, a boolean |

| | |
|---|---|
| Default value | `false` |

## dbms.logs.gc.options

*Table 252. dbms.logs.gc.options*

| | |
|---|---|
| Description | GC Logging Options. |
| Valid values | dbms.logs.gc.options, a string |
| Default value | `-Xlog:gc*,safepoint,age*=trace` |

## dbms.logs.gc.rotation.keep_number

*Table 253. dbms.logs.gc.rotation.keep_number*

| | |
|---|---|
| Description | Number of GC logs to keep. |
| Valid values | dbms.logs.gc.rotation.keep_number, an integer |
| Default value | `5` |

## dbms.logs.gc.rotation.size

*Table 254. dbms.logs.gc.rotation.size*

| | |
|---|---|
| Description | Size of each GC log that is kept. |
| Valid values | dbms.logs.gc.rotation.size, 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`, `PiB`, `PB`, `EiB`, `EB`) |
| Default value | `20.00MiB` |

## dbms.logs.http.enabled

*Table 255. dbms.logs.http.enabled*

| | |
|---|---|
| Description | Enable HTTP request logging. |
| Valid values | dbms.logs.http.enabled, a boolean |
| Default value | `false` |

## dbms.logs.http.format

*Table 256. dbms.logs.http.format*

| | |
|---|---|
| Description | Log format to use for http logs. |
| Valid values | dbms.logs.http.format, one of [PLAIN, JSON]. If unset the value is inherited from dbms.logs.default_format |

## dbms.logs.http.path

*Table 257. dbms.logs.http.path*

| | |
|---|---|
| Description | Path to HTTP request log. |
| Valid values | dbms.logs.http.path, a path. If relative it is resolved from dbms.directories.logs |
| Default value | `http.log` |

## `dbms.logs.http.rotation.keep_number`

*Table 258. dbms.logs.http.rotation.keep_number*

| | |
|---|---|
| Description | Number of HTTP logs to keep. |
| Valid values | dbms.logs.http.rotation.keep_number, an integer |
| Default value | `5` |

## `dbms.logs.http.rotation.size`

*Table 259. dbms.logs.http.rotation.size*

| | |
|---|---|
| Description | Size of each HTTP log that is kept. |
| Valid values | dbms.logs.http.rotation.size, 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`, `PiB`, `PB`, `EiB`, `EB`) which is in the range `0B` to `8388608.00TiB` |
| Default value | `20.00MiB` |

## `dbms.logs.query.allocation_logging_enabled`

Dynamic

*Table 260. dbms.logs.query.allocation_logging_enabled*

| | |
|---|---|
| Description | Log allocated bytes for the executed queries being logged. 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. Requires `dbms.track_query_allocation =true` |
| Valid values | dbms.logs.query.allocation_logging_enabled, a boolean |
| Default value | `true` |

## `dbms.logs.query.early_raw_logging_enabled`

Dynamic

*Table 261. dbms.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 | dbms.logs.query.early_raw_logging_enabled, a boolean |
| Default value | `false` |

## dbms.logs.query.enabled

Dynamic | Enterprise Edition

*Table 262. dbms.logs.query.enabled*

| | |
|---|---|
| Description | Log executed queries. Valid values are `OFF`, `INFO`, or `VERBOSE`.<br>`OFF`<br>no logging.<br>`INFO`<br>log queries at the end of execution, that take longer than the configured threshold, `dbms.logs.query.threshold`.<br>`VERBOSE`<br>log queries at the start and end of execution, regardless of `dbms.logs.query.threshold`.<br>Log entries are written to the query log (dbms.logs.query.path).<br>This feature is available in the Neo4j Enterprise Edition. |
| Valid values | dbms.logs.query.enabled, one of [OFF, INFO, VERBOSE] |
| Default value | VERBOSE |

## dbms.logs.query.format

*Table 263. dbms.logs.query.format*

| | |
|---|---|
| Description | Log format to use for the query log. |
| Valid values | dbms.logs.query.format, one of [PLAIN, JSON]. If unset the value is inherited from dbms.logs.default_format |

## dbms.logs.query.max_parameter_length

Dynamic

*Table 264. dbms.logs.query.max_parameter_length*

| | |
|---|---|
| Description | Sets a maximum character length use for each parameter in the log. This only takes effect if `dbms.logs.query.parameter_logging_enabled = true`. |
| Valid values | dbms.logs.query.max_parameter_length, an integer |
| Default value | 2147483647 |

## dbms.logs.query.obfuscate_literals

Dynamic

*Table 265. dbms.logs.query.obfuscate_literals*

| | |
|---|---|
| Description | Obfuscates all literals of the query before writing to the log. Note that node labels, relationship types and map property keys are still shown. Changing the setting will not affect queries that are cached. So, if you want the switch to have immediate effect, you must also call `CALL db.clearQueryCaches()`. |
| Valid values | dbms.logs.query.obfuscate_literals, a boolean |
| Default value | `false` |

## dbms.logs.query.page_logging_enabled

Dynamic

*Table 266. dbms.logs.query.page_logging_enabled*

| | |
|---|---|
| Description | Log page hits and page faults for the executed queries being logged. |
| Valid values | dbms.logs.query.page_logging_enabled, a boolean |
| Default value | `false` |

## dbms.logs.query.parameter_full_entities

Dynamic

*Table 267. dbms.logs.query.parameter_full_entities*

| | |
|---|---|
| Description | Log complete parameter entities including id, labels or relationship type, and properties. If false, only the entity id will be logged. This only takes effect if `dbms.logs.query.parameter_logging_enabled = true`. |
| Valid values | dbms.logs.query.parameter_full_entities, a boolean |
| Default value | `false` |

## dbms.logs.query.parameter_logging_enabled

Dynamic

*Table 268. dbms.logs.query.parameter_logging_enabled*

| | |
|---|---|
| Description | Log parameters for the executed queries being logged. |
| Valid values | dbms.logs.query.parameter_logging_enabled, a boolean |
| Default value | `true` |

## dbms.logs.query.path

*Table 269. dbms.logs.query.path*

| | |
|---|---|
| Description | Path to the query log file. |
| Valid values | dbms.logs.query.path, a path. If relative it is resolved from dbms.directories.logs |

| | |
|---|---|
| Default value | `query.log` |

## `dbms.logs.query.plan_description_enabled`

Dynamic

*Table 270. dbms.logs.query.plan_description_enabled*

| | |
|---|---|
| Description | Log query plan description table, useful for debugging purposes. |
| Valid values | dbms.logs.query.plan_description_enabled, a boolean |
| Default value | `false` |

## `dbms.logs.query.rotation.keep_number`

Dynamic

*Table 271. dbms.logs.query.rotation.keep_number*

| | |
|---|---|
| Description | Maximum number of history files for the query log. |
| Valid values | dbms.logs.query.rotation.keep_number, an integer which is minimum `1` |
| Default value | `7` |

## `dbms.logs.query.rotation.size`

Dynamic

*Table 272. dbms.logs.query.rotation.size*

| | |
|---|---|
| Description | The file size in bytes at which the query log will auto-rotate. If set to zero then no rotation will occur. Accepts a binary suffix `k`, `m` or `g`. |
| Valid values | dbms.logs.query.rotation.size, 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`, `PiB`, `PB`, `EiB`, `EB`) which is in the range `0B` to `8388608.00TiB` |
| Default value | `20.00MiB` |

## `dbms.logs.query.runtime_logging_enabled`

Dynamic

*Table 273. dbms.logs.query.runtime_logging_enabled*

| | |
|---|---|
| Description | Logs which runtime that was used to run the query. |
| Valid values | dbms.logs.query.runtime_logging_enabled, a boolean |
| Default value | `true` |

## `dbms.logs.query.threshold`

Dynamic

*Table 274. dbms.logs.query.threshold*

| | |
|---|---|
| Description | If the execution of 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 | dbms.logs.query.threshold, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `0s` |

## `dbms.logs.query.time_logging_enabled`

Dynamic

*Table 275. dbms.logs.query.time_logging_enabled*

| | |
|---|---|
| Description | Log detailed time information for the executed queries being logged, such as `(planning: 92, waiting: 0)`. |
| Valid values | dbms.logs.query.time_logging_enabled, a boolean |
| Default value | `false` |

## `dbms.logs.query.transaction.enabled`

Dynamic Enterprise Edition

*Table 276. dbms.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 start and end of transactions that take longer than the configured threshold, dbms.logs.query.transaction.threshold. VERBOSE: log start and end of all transactions. Log entries are written to the query log (dbms.logs.query.path). This feature is available in the Neo4j Enterprise Edition. |
| Valid values | dbms.logs.query.transaction.enabled, one of [OFF, INFO, VERBOSE] |
| Default value | `OFF` |

## `dbms.logs.query.transaction.threshold`

Dynamic

*Table 277. dbms.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 (dbms.logs.query.transaction.enabled) is set to `INFO`. Defaults to 0 seconds (all transactions are logged). |
| Valid values | dbms.logs.query.transaction.threshold, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `0s` |

## `dbms.logs.query.transaction_id.enabled`

Dynamic

*Table 278. dbms.logs.query.transaction_id.enabled*

| | |
|---|---|
| Description | Log transaction ID for the executed queries. |
| Valid values | dbms.logs.query.transaction_id.enabled, a boolean |
| Default value | `false` |

## `dbms.logs.security.format`

*Table 279. dbms.logs.security.format*

| | |
|---|---|
| Description | Log format to use for security log. |
| Valid values | dbms.logs.security.format, one of [PLAIN, JSON]. If unset the value is inherited from dbms.logs.default_format |

## `dbms.logs.security.level`

*Table 280. dbms.logs.security.level*

| | |
|---|---|
| Description | Security log level threshold. |
| Valid values | dbms.logs.security.level, one of [DEBUG, INFO, WARN, ERROR, NONE] |
| Default value | `INFO` |

## `dbms.logs.security.path`

*Table 281. dbms.logs.security.path*

| | |
|---|---|
| Description | Path to the security log file. |
| Valid values | dbms.logs.security.path, a path. If relative it is resolved from dbms.directories.logs |
| Default value | `security.log` |

## `dbms.logs.security.rotation.delay`

Deprecated

*Table 282. dbms.logs.security.rotation.delay*

| | |
|---|---|
| Description | Minimum time interval after last rotation of the security log before it may be rotated again. |
| Valid values | dbms.logs.security.rotation.delay, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `5m` |

## `dbms.logs.security.rotation.keep_number`

*Table 283. dbms.logs.security.rotation.keep_number*

| | |
|---|---|
| Description | Maximum number of history files for the security log. |
| Valid values | dbms.logs.security.rotation.keep_number, an integer which is minimum `1` |
| Default value | `7` |

## dbms.logs.security.rotation.size

*Table 284. dbms.logs.security.rotation.size*

| | |
|---|---|
| Description | Threshold for rotation of the security log. |
| Valid values | dbms.logs.security.rotation.size, 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`, `PiB`, `PB`, `EiB`, `EB`) which is in the range `0B` to `8388608.00TiB` |
| Default value | `20.00MiB` |

## dbms.logs.user.format

*Table 285. dbms.logs.user.format*

| | |
|---|---|
| Description | Log format to use for user log. |
| Valid values | dbms.logs.user.format, one of [PLAIN, JSON]. If unset the value is inherited from dbms.logs.default_format |

## dbms.logs.user.path

*Table 286. dbms.logs.user.path*

| | |
|---|---|
| Description | Path to the user log file. Note that if dbms.logs.user.stdout_enabled is enabled this setting will be ignored. |
| Valid values | dbms.logs.user.path, a path. If relative it is resolved from dbms.directories.logs |
| Default value | `neo4j.log` |

## dbms.logs.user.rotation.delay

Deprecated

*Table 287. dbms.logs.user.rotation.delay*

| | |
|---|---|
| Description | Minimum time interval after last rotation of the user log (*neo4j.log*) before it may be rotated again. Note that if dbms.logs.user.stdout_enabled is enabled this setting will be ignored. |
| Valid values | dbms.logs.user.rotation.delay, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `5m` |

## dbms.logs.user.rotation.keep_number

*Table 288. dbms.logs.user.rotation.keep_number*

| | |
|---|---|
| Description | Maximum number of history files for the user log (*neo4j.log*). Note that if dbms.logs.user.stdout_enabled is enabled this setting will be ignored. |
| Valid values | dbms.logs.user.rotation.keep_number, an integer which is minimum `1` |
| Default value | `7` |

### `dbms.logs.user.rotation.size`

*Table 289. dbms.logs.user.rotation.size*

| | |
|---|---|
| Description | Threshold for rotation of the user log (*neo4j.log*). If set to 0, log rotation is disabled. Note that if dbms.logs.user.stdout_enabled is enabled this setting will be ignored. |
| Valid values | dbms.logs.user.rotation.size, 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`, `PiB`, `PB`, `EiB`, `EB`) which is in the range `0B` to `8388608.00TiB` |
| Default value | `0B` |

### `dbms.logs.user.stdout_enabled`

*Table 290. dbms.logs.user.stdout_enabled*

| | |
|---|---|
| Description | Send user logs to the process stdout. If this is disabled then logs will instead be sent to the file *neo4j.log* located in the logs directory. |
| Valid values | dbms.logs.user.stdout_enabled, a boolean |
| Default value | `true` |

## 7.10.12. Memory settings

### `dbms.memory.heap.initial_size`

*Table 291. dbms.memory.heap.initial_size*

| | |
|---|---|
| Description | Initial heap size. By default it is calculated based on available system resources. |
| Valid values | dbms.memory.heap.initial_size, 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`, `PiB`, `PB`, `EiB`, `EB`) |

### `dbms.memory.heap.max_size`

*Table 292. dbms.memory.heap.max_size*

| | |
|---|---|
| Description | Maximum heap size. By default it is calculated based on available system resources. |
| Valid values | dbms.memory.heap.max_size, 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`, `PiB`, `PB`, `EiB`, `EB`) |

### `dbms.memory.off_heap.block_cache_size`

*Table 293. dbms.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 * dbms.memory.off_heap.max_cacheable_block_size * `dbms.memory.off_heap.block_cache_size` |
| Valid values | dbms.memory.off_heap.block_cache_size, an integer which is minimum `16` |
| Default value | `128` |

## `dbms.memory.off_heap.max_cacheable_block_size`

*Table 294. dbms.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 | dbms.memory.off_heap.max_cacheable_block_size, 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`, `PiB`, `PB`, `EiB`, `EB`) which is minimum `4` and is power of 2 |
| Default value | `512.00KiB` |

## `dbms.memory.off_heap.max_size`

*Table 295. dbms.memory.off_heap.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 dbms.tx_state.memory_allocation is set to 'OFF_HEAP'. |
| Valid values | dbms.memory.off_heap.max_size, 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`, `PiB`, `PB`, `EiB`, `EB`) which is minimum `0B` |
| Default value | `2.00GiB` |

## `dbms.memory.pagecache.directio`

*Table 296. dbms.memory.pagecache.directio*

| | |
|---|---|
| Description | Use direct I/O for page cache. Setting is supported only on Linux and only for a subset of record formats that use platform aligned page size. |
| Valid values | dbms.memory.pagecache.directio, a boolean |
| Default value | `false` |

## `dbms.memory.pagecache.flush.buffer.enabled`

Dynamic

*Table 297. dbms.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 | dbms.memory.pagecache.flush.buffer.enabled, a boolean |
| Default value | `false` |

## dbms.memory.pagecache.flush.buffer.size_in_pages

Dynamic

*Table 298. dbms.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 buffer size in pages (8KiB). To be able to utilize this buffer during page cache flushing, buffered flush should be enabled. |
| Valid values | dbms.memory.pagecache.flush.buffer.size_in_pages, an integer which is in the range `1` to `512` |
| Default value | `128` |

## dbms.memory.pagecache.scan.prefetchers

*Table 299. dbms.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 | dbms.memory.pagecache.scan.prefetchers, an integer which is in the range `0` to `255` |
| Default value | `4` |

## dbms.memory.pagecache.size

*Table 300. dbms.memory.pagecache.size*

| | |
|---|---|
| Description | The amount of memory to use for mapping the store files, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g'). 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. |
| Valid values | dbms.memory.pagecache.size, a string |

## dbms.memory.pagecache.swapper

Deprecated

*Table 301. dbms.memory.pagecache.swapper*

| | |
|---|---|
| Description | This setting is not used anymore. |
| Valid values | dbms.memory.pagecache.swapper, a string |

## dbms.memory.pagecache.warmup.enable

Enterprise Edition

*Table 302. dbms.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 | dbms.memory.pagecache.warmup.enable, a boolean |
| Default value | `true` |

## dbms.memory.pagecache.warmup.preload

*Table 303. dbms.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 | dbms.memory.pagecache.warmup.preload, a boolean |
| Default value | `false` |

## dbms.memory.pagecache.warmup.preload.allowlist

*Table 304. dbms.memory.pagecache.warmup.preload.allowlist*

| | |
|---|---|
| Description | Page cache warmup prefetch file allowlist regex. By default matches all files. |
| Valid values | dbms.memory.pagecache.warmup.preload.allowlist, a string |
| Default value | `.*` |

## dbms.memory.pagecache.warmup.preload.whitelist

Deprecated

*Table 305. dbms.memory.pagecache.warmup.preload.whitelist*

| | |
|---|---|
| Description | Page cache warmup prefetch file whitelist regex. By default matches all files. |
| Valid values | dbms.memory.pagecache.warmup.preload.whitelist, a string |
| Default value | `.*` |
| Replaced by | dbms.memory.pagecache.warmup.preload.allowlist |

## dbms.memory.pagecache.warmup.profile.interval

Enterprise Edition

*Table 306. dbms.memory.pagecache.warmup.profile.interval*

| | |
|---|---|
| Description | The profiling frequency for the page cache. Accurate profiles allow the page cache to do active warmup after a restart, reducing the mean time to performance. This feature is available in Neo4j Enterprise Edition. |
| Valid values | dbms.memory.pagecache.warmup.profile.interval, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `1m` |

## `dbms.memory.tracking.enable`

*Table 307. dbms.memory.tracking.enable*

| | |
|---|---|
| Description | Enable off heap and on heap memory tracking. Should not be set to `false` for clusters. |
| Valid values | dbms.memory.tracking.enable, a boolean |
| Default value | `true` |

## `dbms.memory.transaction.database_max_size`

Dynamic

*Table 308. dbms.memory.transaction.database_max_size*

| | |
|---|---|
| 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 | dbms.memory.transaction.database_max_size, 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`, `PiB`, `PB`, `EiB`, `EB`) which is minimum `10.00MiB` `0B` |
| Default value | `0B` |

## `dbms.memory.transaction.global_max_size`

Dynamic

*Table 309. dbms.memory.transaction.global_max_size*

| | |
|---|---|
| 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'. |
| Valid values | dbms.memory.transaction.global_max_size, 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`, `PiB`, `PB`, `EiB`, `EB`) which is minimum `10.00MiB` or i |
| Default value | `0B` |

## `dbms.memory.transaction.max_size`

Dynamic

*Table 310. dbms.memory.transaction.max_size*

| | |
|---|---|
| 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'. When `dbms.mode =CORE` or `dbms.mode =SINGLE` and `dbms.clustering.enable =true` this is '2G', in other cases this is 'unlimited'. |
| Valid values | dbms.memory.transaction.max_size, 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`, `PiB`, `PB`, `EiB`, `EB`) which is minimum `1.00MiB` or is `0B` and depends on dbms.mode. If dbms.mode one of `[CORE]` then it is maximum `2.00GiB` otherwise it depends on dbms.mode. If dbms.mode one of `[SINGLE]` then it depends on dbms.clustering.enable. If dbms.clustering.enable is `true` then it is maximum `2.00GiB` otherwise it is unconstrained. otherwise it is unconstrained. |
| Default value | `0B` |

### `dbms.tx_state.memory_allocation`

*Table 311. dbms.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 | dbms.tx_state.memory_allocation, one of [ON_HEAP, OFF_HEAP] |
| Default value | `OFF_HEAP` |

### `dbms.query_cache_size`

*Table 312. dbms.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` * `dbms.query_cache_size`. With 10 databases and `dbms.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 | dbms.query_cache_size, an integer which is minimum `0` |
| Default value | `1000` |

## 7.10.13. Metrics settings

### `metrics.bolt.messages.enabled`

Deprecated

*Table 313. metrics.bolt.messages.enabled*

| | |
|---|---|
| Description | Enable reporting metrics about Bolt Protocol message processing. |
| Valid values | metrics.bolt.messages.enabled, a boolean |
| Default value | `false` |
| Replaced by | metrics.filter |

## `metrics.csv.enabled`

*Table 314. metrics.csv.enabled*

| | |
|---|---|
| **Description** | Set to `true` to enable exporting metrics to CSV files. |
| **Valid values** | metrics.csv.enabled, a boolean |
| **Default value** | `true` |

## `metrics.csv.interval`

*Table 315. 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** | metrics.csv.interval, a duration (Valid units are: `ns`, `µ s`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) which is minimum `1ms` |
| **Default value** | `30s` |

## `metrics.csv.rotation.compression`

*Table 316. metrics.csv.rotation.compression*

| | |
|---|---|
| **Description** | Decides what compression to use for the csv history files. |
| **Valid values** | metrics.csv.rotation.compression, one of [NONE, ZIP, GZ] |
| **Default value** | `NONE` |

## `metrics.csv.rotation.keep_number`

*Table 317. metrics.csv.rotation.keep_number*

| | |
|---|---|
| **Description** | Maximum number of history files for the csv files. |
| **Valid values** | metrics.csv.rotation.keep_number, an integer which is minimum `1` |
| **Default value** | `7` |

## `metrics.csv.rotation.size`

*Table 318. 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** | metrics.csv.rotation.size, 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`, `PiB`, `PB`, `EiB`, `EB`) which is in the range `0B` to `8388608.00TiB` |
| **Default value** | `10.00MiB` |

## `metrics.cypher.replanning.enabled`

Deprecated

*Table 319. metrics.cypher.replanning.enabled*

| | |
|---|---|
| Description | Enable reporting metrics about number of occurred replanning events. instead. |
| Valid values | metrics.cypher.replanning.enabled, a boolean |
| Default value | `false` |
| Replaced by | metrics.filter |

## `metrics.enabled`

*Table 320. metrics.enabled*

| | |
|---|---|
| Description | Enable metrics. Setting this to `false` will to turn off all metrics. |
| Valid values | metrics.enabled, a boolean |
| Default value | `true` |

## `metrics.filter`

*Table 321. 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 | metrics.filter, a ',' separated list with elements of type '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,*causal_clustering.core.is_leader,*causal_clustering.core.last_leader_message,*causal_clustering.core.replication_attempt,*causal_clustering.core.replication_fail,*check_point.duration,*check_point.total_time,*cypher.replan_events,*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*` |

## `metrics.graphite.enabled`

*Table 322. metrics.graphite.enabled*

| | |
|---|---|
| Description | Set to `true` to enable exporting metrics to Graphite. |
| Valid values | metrics.graphite.enabled, a boolean |
| Default value | `false` |

## `metrics.graphite.interval`

*Table 323. metrics.graphite.interval*

| | |
|---|---|
| Description | The reporting interval for Graphite. That is, how often to send updated metrics to Graphite. |
| Valid values | metrics.graphite.interval, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `30s` |

## `metrics.graphite.server`

*Table 324. metrics.graphite.server*

| | |
|---|---|
| Description | The hostname or IP address of the Graphite server. |
| Valid values | metrics.graphite.server, a socket address in the format 'hostname:port', 'hostname' or ':port'. If missing port or hostname it is acquired from dbms.default_listen_address |
| Default value | `:2003` |

## `metrics.jmx.enabled`

*Table 325. metrics.jmx.enabled*

| | |
|---|---|
| Description | Set to `true` to enable the JMX metrics endpoint. |
| Valid values | metrics.jmx.enabled, a boolean |
| Default value | `true` |

## `metrics.jvm.buffers.enabled`

Deprecated

*Table 326. metrics.jvm.buffers.enabled*

| | |
|---|---|
| Description | Enable reporting metrics about the buffer pools. |
| Valid values | metrics.jvm.buffers.enabled, a boolean |
| Default value | `false` |
| Replaced by | metrics.filter |

## `metrics.jvm.file.descriptors.enabled`

Deprecated

*Table 327. metrics.jvm.file.descriptors.enabled*

| | |
|---|---|
| Description | Enable reporting metrics about the number of open file descriptors. |
| Valid values | metrics.jvm.file.descriptors.enabled, a boolean |

| | |
|---|---|
| Default value | `false` |
| Replaced by | metrics.filter |

## `metrics.jvm.gc.enabled`

Deprecated

*Table 328. metrics.jvm.gc.enabled*

| | |
|---|---|
| Description | Enable reporting metrics about the duration of garbage collections. |
| Valid values | metrics.jvm.gc.enabled, a boolean |
| Default value | `false` |
| Replaced by | metrics.filter |

## `metrics.jvm.heap.enabled`

Deprecated

*Table 329. metrics.jvm.heap.enabled*

| | |
|---|---|
| Description | Enable reporting metrics about the heap memory usage. |
| Valid values | metrics.jvm.heap.enabled, a boolean |
| Default value | `false` |
| Replaced by | metrics.filter |

## `metrics.jvm.memory.enabled`

Deprecated

*Table 330. metrics.jvm.memory.enabled*

| | |
|---|---|
| Description | Enable reporting metrics about the memory usage. |
| Valid values | metrics.jvm.memory.enabled, a boolean |
| Default value | `false` |
| Replaced by | metrics.filter |

## `metrics.jvm.pause_time.enabled`

Deprecated

*Table 331. metrics.jvm.pause_time.enabled*

| | |
|---|---|
| Description | Enable reporting metrics about the VM pause time. |

| | |
|---|---|
| Valid values | metrics.jvm.pause_time.enabled, a boolean |
| Default value | `false` |
| Replaced by | metrics.filter |

## `metrics.jvm.threads.enabled`

Deprecated

*Table 332. metrics.jvm.threads.enabled*

| | |
|---|---|
| Description | Enable reporting metrics about the current number of threads running. |
| Valid values | metrics.jvm.threads.enabled, a boolean |
| Default value | `false` |
| Replaced by | metrics.filter |

## `metrics.namespaces.enabled`

*Table 333. metrics.namespaces.enabled*

| | |
|---|---|
| Description | Enable metrics namespaces that separates the global and database specific metrics. If enabled all database specific metrics will have field names starting with <metrics_prefix>.database.<database_name> and all global metrics will start with <metrics_prefix>.dbms. For example, neo4j.page_cache.hits will become neo4j.dbms.page_cache.hits and neo4j.system.log.rotation_events will become neo4j.database.system.log.rotation_events. |
| Valid values | metrics.namespaces.enabled, a boolean |
| Default value | `false` |

## `metrics.neo4j.causal_clustering.enabled`

Deprecated

*Table 334. metrics.neo4j.causal_clustering.enabled*

| | |
|---|---|
| Description | Enable reporting metrics about Causal Clustering mode. |
| Valid values | metrics.neo4j.causal_clustering.enabled, a boolean |
| Default value | `false` |
| Replaced by | metrics.filter |

## `metrics.neo4j.checkpointing.enabled`

Deprecated

*Table 335. metrics.neo4j.checkpointing.enabled*

| | |
|---|---|
| Description | Enable reporting metrics about Neo4j check pointing; when it occurs and how much time it takes to complete. |
| Valid values | metrics.neo4j.checkpointing.enabled, a boolean |
| Default value | `false` |
| Replaced by | metrics.filter |

## `metrics.neo4j.counts.enabled`

Deprecated

*Table 336. metrics.neo4j.counts.enabled*

| | |
|---|---|
| Description | Enable reporting metrics about approximately how many entities are in the database; nodes, relationships, properties, etc. |
| Valid values | metrics.neo4j.counts.enabled, a boolean |
| Default value | `false` |
| Replaced by | metrics.filter |

## `metrics.neo4j.data.counts.enabled`

Deprecated

*Table 337. metrics.neo4j.data.counts.enabled*

| | |
|---|---|
| Description | Enable reporting metrics about number of entities in the database. |
| Valid values | metrics.neo4j.data.counts.enabled, a boolean |
| Default value | `false` |
| Replaced by | metrics.filter |

## `metrics.neo4j.database_operation_count.enabled`

Deprecated

*Table 338. metrics.neo4j.database_operation_count.enabled*

| | |
|---|---|
| Description | Enable reporting metrics for Neo4j dbms operations; how many times databases have been created, started, stopped or dropped, and how many attempted operations have failed and recovered later. |
| Valid values | metrics.neo4j.database_operation_count.enabled, a boolean |
| Default value | `false` |
| Replaced by | metrics.filter |

## `metrics.neo4j.logs.enabled`

Deprecated

*Table 339. metrics.neo4j.logs.enabled*

| | |
|---|---|
| Description | Enable reporting metrics about the Neo4j transaction logs. |
| Valid values | metrics.neo4j.logs.enabled, a boolean |
| Default value | `false` |
| Replaced by | metrics.filter |

## `metrics.neo4j.pagecache.enabled`

Deprecated

*Table 340. metrics.neo4j.pagecache.enabled*

| | |
|---|---|
| Description | Enable reporting metrics about the Neo4j page cache; page faults, evictions, flushes, exceptions, etc. |
| Valid values | metrics.neo4j.pagecache.enabled, a boolean |
| Default value | `false` |
| Replaced by | metrics.filter |

## `metrics.neo4j.pools.enabled`

Deprecated

*Table 341. metrics.neo4j.pools.enabled*

| | |
|---|---|
| Description | Enable reporting metrics about Neo4j memory pools. |
| Valid values | metrics.neo4j.pools.enabled, a boolean |
| Default value | `false` |
| Replaced by | metrics.filter |

## `metrics.neo4j.server.enabled`

Deprecated

*Table 342. metrics.neo4j.server.enabled*

| | |
|---|---|
| Description | Enable reporting metrics about Server threading info. |
| Valid values | metrics.neo4j.server.enabled, a boolean |
| Default value | `false` |
| Replaced by | metrics.filter |

## `metrics.neo4j.size.enabled`

Deprecated

*Table 343. metrics.neo4j.size.enabled*

| | |
|---|---|
| Description | Enable reporting metrics about the store size of each database. |
| Valid values | metrics.neo4j.size.enabled, a boolean |
| Default value | `false` |
| Replaced by | metrics.filter |

## `metrics.neo4j.tx.enabled`

Deprecated

*Table 344. metrics.neo4j.tx.enabled*

| | |
|---|---|
| Description | Enable reporting metrics about transactions; number of transactions started, committed, etc. |
| Valid values | metrics.neo4j.tx.enabled, a boolean |
| Default value | `false` |
| Replaced by | metrics.filter |

## `metrics.prefix`

*Table 345. metrics.prefix*

| | |
|---|---|
| Description | A common prefix for the reported metrics field names. |
| Valid values | metrics.prefix, a string |
| Default value | `neo4j` |

## `metrics.prometheus.enabled`

*Table 346. metrics.prometheus.enabled*

| | |
|---|---|
| Description | Set to `true` to enable the Prometheus endpoint. |
| Valid values | metrics.prometheus.enabled, a boolean |
| Default value | `false` |

## `metrics.prometheus.endpoint`

*Table 347. metrics.prometheus.endpoint*

| | |
|---|---|
| Description | The hostname and port to use as Prometheus endpoint. |

| | |
|---|---|
| Valid values | metrics.prometheus.endpoint, a socket address in the format `hostname:port`, `hostname`, or `:port`. If missing, port and hostname are acquired from `dbms.default_listen_address`. |
| Default value | `localhost:2004` |

## 7.10.14. Neo4j Browser and client settings

### `browser.allow_outgoing_connections`

*Table 348. browser.allow_outgoing_connections*

| | |
|---|---|
| Description | Configure the policy for outgoing Neo4j Browser connections. |
| Valid values | browser.allow_outgoing_connections, a boolean |
| Default value | `true` |

### `browser.credential_timeout`

*Table 349. 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 | browser.credential_timeout, 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 350. 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 | browser.post_connect_cmd, a string |
| Default value | |

### `browser.remote_content_hostname_whitelist`

*Table 351. browser.remote_content_hostname_whitelist*

| | |
|---|---|
| Description | Whitelist of hosts for the Neo4j Browser to be allowed to fetch content from. |
| Valid values | browser.remote_content_hostname_whitelist, a string |
| Default value | `guides.neo4j.com,localhost` |

### `browser.retain_connection_credentials`

*Table 352. browser.retain_connection_credentials*

| | |
|---|---|
| Description | Configure the Neo4j Browser to store or not store user credentials. |
| Valid values | browser.retain_connection_credentials, a boolean |
| Default value | `true` |

### `browser.retain_editor_history`

*Table 353. browser.retain_editor_history*

| | |
|---|---|
| Description | Configure the Neo4j Browser to store or not store user editor history. |
| Valid values | browser.retain_editor_history, a boolean |
| Default value | `true` |

### `clients.allow_telemetry`

*Table 354. clients.allow_telemetry*

| | |
|---|---|
| Description | Configure client applications such as Browser and Bloom to send Product Analytics data. |
| Valid values | clients.allow_telemetry, a boolean |
| Default value | `true` |

## 7.10.15. Security settings

### `dbms.security.allow_csv_import_from_file_urls`

*Table 355. 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 | dbms.security.allow_csv_import_from_file_urls, a boolean |
| Default value | `true` |

### `dbms.security.auth_cache_max_capacity`

*Table 356. dbms.security.auth_cache_max_capacity*

| | |
|---|---|
| Description | The maximum capacity for authentication and authorization caches (respectively). |
| Valid values | dbms.security.auth_cache_max_capacity, an integer |
| Default value | `10000` |

### `dbms.security.auth_cache_ttl`

*Table 357. 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 | dbms.security.auth_cache_ttl, 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`

*Table 358. 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 | dbms.security.auth_cache_use_ttl, a boolean |
| Default value | `true` |

## `dbms.security.auth_enabled`

*Table 359. dbms.security.auth_enabled*

| | |
|---|---|
| Description | Enable auth requirement to access Neo4j. |
| Valid values | dbms.security.auth_enabled, a boolean |
| Default value | `true` |

## `dbms.security.auth_lock_time`

*Table 360. 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 | dbms.security.auth_lock_time, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) which is minimum `0s` |
| Default value | `5s` |

## `dbms.security.auth_max_failed_attempts`

*Table 361. 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 | dbms.security.auth_max_failed_attempts, an integer which is minimum `0` |
| Default value | `3` |

## dbms.security.authentication_providers

*Table 362. 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 | dbms.security.authentication_providers, a ',' separated list with elements of type 'a string'. |
| Default value | `native` |

## dbms.security.authorization_providers

*Table 363. 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 | dbms.security.authorization_providers, a ',' separated list with elements of type 'a string'. |
| Default value | `native` |

## dbms.security.causal_clustering_status_auth_enabled

*Table 364. dbms.security.causal_clustering_status_auth_enabled*

| | |
|---|---|
| Description | Require authorization for access to the Causal Clustering status endpoints. |
| Valid values | dbms.security.causal_clustering_status_auth_enabled, a boolean |
| Default value | `true` |

## dbms.security.http_access_control_allow_origin

*Table 365. 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 | dbms.security.http_access_control_allow_origin, a string |
| Default value | * |

## dbms.security.http_auth_allowlist

*Table 366. dbms.security.http_auth_allowlist*

| | |
|---|---|
| Description | Defines an allowlist of http paths where Neo4j authentication is not required. |
| Valid values | dbms.security.http_auth_allowlist, a ',' separated list with elements of type 'a string'. |
| Default value | `/,/browser.*` |

## `dbms.security.http_auth_whitelist`

Deprecated

*Table 367. dbms.security.http_auth_whitelist*

| | |
|---|---|
| Description | Defines a whitelist of http paths where Neo4j authentication is not required. |
| Valid values | dbms.security.http_auth_whitelist, a ',' separated list with elements of type 'a string'. |
| Default value | `/,/browser.*` |
| Replaced by | dbms.security.http_auth_allowlist |

## `dbms.security.http_strict_transport_security`

*Table 368. 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 | dbms.security.http_strict_transport_security, a string |

## `dbms.security.ldap.authentication.attribute`

Dynamic

*Table 369. 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 | dbms.security.ldap.authentication.attribute, a string which 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`

*Table 370. 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 which this Neo4j instance is deployed. |
| Valid values | dbms.security.ldap.authentication.cache_enabled, a boolean |
| Default value | `true` |

## `dbms.security.ldap.authentication.mechanism`

*Table 371. 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 | dbms.security.ldap.authentication.mechanism, a string |
| Default value | `simple` |

## `dbms.security.ldap.authentication.search_for_attribute`

*Table 372. 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 | dbms.security.ldap.authentication.search_for_attribute, a boolean |
| Default value | `false` |

## `dbms.security.ldap.authentication.use_samaccountname`

Deprecated

*Table 373. dbms.security.ldap.authentication.use_samaccountname*

| | |
|---|---|
| Description | Perform authentication by searching for an unique attribute of a user. |
| Valid values | dbms.security.ldap.authentication.use_samaccountname, a boolean |
| Default value | `false` |
| Replaced by | dbms.security.ldap.authentication.search_for_attribute |

## `dbms.security.ldap.authentication.user_dn_template`

Dynamic

*Table 374. 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 | dbms.security.ldap.authentication.user_dn_template, a string which 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`

Dynamic

*Table 375. 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 | dbms.security.ldap.authorization.access_permitted_group, a string |
| Default value | |

## `dbms.security.ldap.authorization.group_membership_attributes`

Dynamic

*Table 376. 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. |
| Valid values | dbms.security.ldap.authorization.group_membership_attributes, a ',' separated list with elements of type 'a string'. which Can not be empty |
| Default value | `memberOf` |

## `dbms.security.ldap.authorization.group_to_role_mapping`

Dynamic

*Table 377. 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:<br>`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 | dbms.security.ldap.authorization.group_to_role_mapping, a string which must be semicolon separated list of key-value pairs or empty |
| Default value | |

## dbms.security.ldap.authorization.system_password

*Table 378. 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 | dbms.security.ldap.authorization.system_password, a secure string |

## dbms.security.ldap.authorization.system_username

*Table 379. 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 | dbms.security.ldap.authorization.system_username, a string |

## dbms.security.ldap.authorization.use_system_account

*Table 380. 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 | dbms.security.ldap.authorization.use_system_account, a boolean |

| | |
|---|---|
| Default value | `false` |

## `dbms.security.ldap.authorization.user_search_base`

Dynamic

*Table 381. 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 | dbms.security.ldap.authorization.user_search_base, a string which Can not be empty |
| Default value | `ou=users,dc=example,dc=com` |

## `dbms.security.ldap.authorization.user_search_filter`

Dynamic

*Table 382. 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 | dbms.security.ldap.authorization.user_search_filter, a string |
| Default value | `(&(objectClass=*)(uid={0}))` |

## `dbms.security.ldap.connection_timeout`

*Table 383. 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 | dbms.security.ldap.connection_timeout, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `30s` |

## `dbms.security.ldap.host`

*Table 384. 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 | dbms.security.ldap.host, a string |

| | |
|---|---|
| Default value | `localhost` |

## `dbms.security.ldap.read_timeout`

*Table 385. 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 | dbms.security.ldap.read_timeout, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `30s` |

## `dbms.security.ldap.referral`

*Table 386. 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 | dbms.security.ldap.referral, a string |
| Default value | `follow` |

## `dbms.security.ldap.use_starttls`

*Table 387. 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 | dbms.security.ldap.use_starttls, a boolean |
| Default value | `false` |

## `dbms.security.log_successful_authentication`

*Table 388. 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 | dbms.security.log_successful_authentication, a boolean |
| Default value | `true` |

## `dbms.security.oidc.<provider>.audience`

Dynamic

*Table 389. dbms.security.oidc.<provider>.audience*

| | |
|---|---|
| Description | Expected values of the Audience (aud) claim in the id token. |
| Valid values | dbms.security.oidc.<provider>.audience, a ',' separated list with elements of type 'a string'. which Can not be empty |

## dbms.security.oidc.<provider>.auth_endpoint

Dynamic

*Table 390. 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 | dbms.security.oidc.<provider>.auth_endpoint, a URI |

## dbms.security.oidc.<provider>.auth_flow

Dynamic

*Table 391. dbms.security.oidc.<provider>.auth_flow*

| | |
|---|---|
| Description | The OIDC flow to use. This is exposed to clients via the discovery endpoint. |
| Valid values | dbms.security.oidc.<provider>.auth_flow, one of [PKCE, IMPLICIT] |
| Default value | PKCE |

## dbms.security.oidc.<provider>.auth_params

Dynamic

*Table 392. dbms.security.oidc.<provider>.auth_params*

| | |
|---|---|
| Description | Optional additional configuration options used for the authentication request. The map is a semicolon separated list of key-value pairs. For example: `k1=v1;k2=v2`. |
| Valid values | dbms.security.oidc.<provider>.auth_params, A simple key value map pattern k1=v1;k2=v2 |
| Default value | {} |

## dbms.security.oidc.<provider>.authorization.group_to_role_mapping

Dynamic

*Table 393. 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:<br>`dbms.security.oidc.<provider>.authorization.group_to_role_mapping=\`<br>`"Neo4j Read Only" = reader; \`<br>`"Neo4j Read-Write" = publisher; \`<br>`"Neo4j Schema Manager" = architect; \`<br>`"Neo4j Administrator" = admin` |
| Valid values | dbms.security.oidc.<provider>.authorization.group_to_role_mapping, a string which must be semicolon separated list of key-value pairs or empty |

## dbms.security.oidc.<provider>.claims.groups

Dynamic

*Table 394. 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. |
| Valid values | dbms.security.oidc.<provider>.claims.groups, a string |

## dbms.security.oidc.<provider>.claims.username

Dynamic

*Table 395. 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 | dbms.security.oidc.<provider>.claims.username, a string |
| Default value | sub |

## dbms.security.oidc.<provider>.client_id

Dynamic

*Table 396. dbms.security.oidc.<provider>.client_id*

| | |
|---|---|
| Description | Client id needed if token contains multiple Audience (aud) claims. |
| Valid values | dbms.security.oidc.<provider>.client_id, a string |

## dbms.security.oidc.<provider>.config

Dynamic

*Table 397. dbms.security.oidc.<provider>.config*

| | |
|---|---|
| Description | Additional configuration options that the clients may require to authenticate. The map is a semicolon separated list of key-value pairs. For example: `k1=v1;k2=v2`. |
| Valid values | dbms.security.oidc.<provider>.config, A simple key value map pattern k1=v1;k2=v2 |
| Default value | `{}` |

## dbms.security.logs.oidc.jwt_claims_at_debug_level_enabled

*Table 398. 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`.<br>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 | dbms.security.logs.oidc.jwt_claims_at_debug_level_enabled, a boolean |
| Default value | `false` |

## dbms.security.oidc.<provider>.display_name

*Table 399. 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 | dbms.security.oidc.<provider>.display_name, a string |

## dbms.security.oidc.<provider>.get_groups_from_user_info

Dynamic

*Table 400. 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 | dbms.security.oidc.<provider>.get_groups_from_user_info, a boolean |
| Default value | `false` |

## dbms.security.oidc.<provider>.get_username_from_user_info

Dynamic

*Table 401. 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 | dbms.security.oidc.<provider>.get_username_from_user_info, a boolean |
| Default value | `false` |

## dbms.security.oidc.<provider>.issuer

Dynamic

*Table 402. dbms.security.oidc.<provider>.issuer*

| | |
|---|---|
| Description | The expected value of the iss claim in the id token. |
| Valid values | dbms.security.oidc.<provider>.issuer, a string |

## dbms.security.oidc.<provider>.jwks_uri

Dynamic

*Table 403. 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 | dbms.security.oidc.<provider>.jwks_uri, a URI |

## dbms.security.oidc.<provider>.params

Dynamic

*Table 404. dbms.security.oidc.<provider>.params*

| | |
|---|---|
| Description | Generic parameters that the clients may require to authenticate. The map is a semicolon separated list of key-value pairs. For example: `k1=v1;k2=v2`. |
| Valid values | dbms.security.oidc.<provider>.params, A simple key value map pattern k1=v1;k2=v2 |
| Default value | `{}` |

## dbms.security.oidc.<provider>.redirect_uri

Dynamic Deprecated

*Table 405. dbms.security.oidc.<provider>.redirect_uri*

| | |
|---|---|
| Description | The redirect URI the IdP should return the user to when authenticated. |
| Valid values | dbms.security.oidc.<provider>.redirect_uri, a URI |

## dbms.security.oidc.<provider>.token_endpoint

Dynamic

*Table 406. 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 | dbms.security.oidc.<provider>.token_endpoint, a URI |

## dbms.security.oidc.<provider>.token_params

Dynamic

*Table 407. dbms.security.oidc.<provider>.token_params*

| | |
|---|---|
| Description | Optional additional configuration options used for the token request. The map is a semicolon separated list of key-value pairs. For example: `k1=v1;k2=v2`. |
| Valid values | dbms.security.oidc.<provider>.token_params, A simple key value map pattern k1=v1;k2=v2 |
| Default value | `{}` |

## dbms.security.oidc.<provider>.user_info_uri

Dynamic

*Table 408. dbms.security.oidc.<provider>.user_info_uri*

| | |
|---|---|
| Description | The identity providers user info uri. |
| Valid values | dbms.security.oidc.<provider>.user_info_uri, a URI |

## dbms.security.oidc.<provider>.well_known_discovery_uri

Dynamic

*Table 409. dbms.security.oidc.<provider>.well_known_discovery_uri*

| | |
|---|---|
| Description | The 'well known' OpenID Connect Discovery endpoint used to fetch identity provider settings. |
| Valid values | dbms.security.oidc.<provider>.well_known_discovery_uri, a URI |

## dbms.security.procedures.allowlist

*Table 410. 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 '*'. If this setting is left empty no procedures will be loaded. |
| Valid values | dbms.security.procedures.allowlist, a ',' separated list with elements of type 'a string'. |
| Default value | * |

## dbms.security.procedures.default_allowed

Deprecated

*Table 411. dbms.security.procedures.default_allowed*

| | |
|---|---|
| Description | The default role that can execute all procedures and user-defined functions that are not covered by the `dbms.security.procedures.roles` setting. This setting (if not empty string) will be translated to 'GRANT EXECUTE BOOSTED PROCEDURE *' and 'GRANT EXECUTE BOOSTED FUNCTION *' for that role. If `dbms.security.procedures.roles`is not empty, any procedure or function that this role is not mapped to will result in a 'DENY EXECUTE BOOSTED PROCEDURE name' and 'DENY EXECUTE BOOSTED FUNCTION name' for this role. Any privilege mapped in this way cannot be revoked, instead the config must be changed and will take effect after a restart. |
| Valid values | dbms.security.procedures.default_allowed, a string |
| Default value | |
| Replaced by | `EXECUTE PROCEDURE`, `EXECUTE BOOSTED PROCEDURE`, `EXECUTE FUNCTION` and `EXECUTE BOOSTED FUNCTION` privileges |

## dbms.security.procedures.roles

Deprecated

*Table 412. dbms.security.procedures.roles*

| | |
|---|---|
| Description | This provides a finer level of control over which roles can execute procedures than the `dbms.security.procedures.default_allowed` setting. For example: `dbms.security.procedures.roles=apoc.convert.*:reader;apoc.load.json*:writer;apoc.trigger.add:TriggerHappy` will allow the role `reader` to execute all procedures in the `apoc.convert` namespace, the role `writer` to execute all procedures in the `apoc.load` namespace that starts with `json` and the role `TriggerHappy` to execute the specific procedure `apoc.trigger.add`. Procedures not matching any of these patterns will be subject to the `dbms.security.procedures.default_allowed` setting. This setting (if not empty string) will be translated to 'GRANT EXECUTE BOOSTED PROCEDURE name' and 'GRANT EXECUTE BOOSTED FUNCTION name' privileges for the mapped roles. Any privilege mapped in this way cannot be revoked, instead the config must be changed and will take effect after a restart. |
| Valid values | dbms.security.procedures.roles, a string |
| Default value | |
| Replaced by | `EXECUTE PROCEDURE`, `EXECUTE BOOSTED PROCEDURE`, `EXECUTE FUNCTION` and `EXECUTE BOOSTED FUNCTION` privileges |

## dbms.security.procedures.unrestricted

*Table 413. 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 | dbms.security.procedures.unrestricted, a ',' separated list with elements of type 'a string'. |
| Default value | |

## dbms.security.procedures.whitelist

Deprecated

*Table 414. dbms.security.procedures.whitelist*

| | |
|---|---|
| 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 '*'. If this setting is left empty no procedures will be loaded. |
| Valid values | dbms.security.procedures.whitelist, a ',' separated list with elements of type 'a string'. |
| Default value | * |
| Replaced by | dbms.security.procedures.allowlist |

## `dbms.netty.ssl.provider`

*Table 415. dbms.netty.ssl.provider*

| | |
|---|---|
| Description | Netty SSL provider. |
| Valid values | dbms.netty.ssl.provider, one of [JDK, OPENSSL, OPENSSL_REFCNT] |
| Default value | `JDK` |

## `systemdb.secrets.key.name`

Dynamic

*Table 416. systemdb.secrets.key.name*

| | |
|---|---|
| Description | Name of the 256 length AES encryption key, which is used for the symmetric encryption. |
| Valid values | systemdb.secrets.key.name, a string |
| Default value | `aesKey` |

## `systemdb.secrets.keystore.password`

Dynamic

*Table 417. systemdb.secrets.keystore.password*

| | |
|---|---|
| Description | Password for accessing the keystore holding a 256 length AES encryption key, which is used for the symmetric encryption. |
| Valid values | systemdb.secrets.keystore.password, a secure string |

## `systemdb.secrets.keystore.path`

Dynamic

*Table 418. systemdb.secrets.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 | systemdb.secrets.keystore.path, a path |

# 7.10.16. Transaction settings

## `dbms.lock.acquisition.timeout`

Dynamic

*Table 419. dbms.lock.acquisition.timeout*

| | |
|---|---|
| **Description** | The maximum time interval within which lock should be acquired. Zero (default) means timeout is disabled. |
| **Valid values** | dbms.lock.acquisition.timeout, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| **Default value** | `0s` |

## `dbms.shutdown_transaction_end_timeout`

*Table 420. dbms.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** | dbms.shutdown_transaction_end_timeout, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| **Default value** | `10s` |

## `dbms.transaction.bookmark_ready_timeout`

Dynamic

*Table 421. dbms.transaction.bookmark_ready_timeout*

| | |
|---|---|
| **Description** | The maximum amount of time to wait for the database state represented by the bookmark. |
| **Valid values** | dbms.transaction.bookmark_ready_timeout, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) which is minimum `1s` |
| **Default value** | `30s` |

## `dbms.transaction.concurrent.maximum`

Dynamic

*Table 422. dbms.transaction.concurrent.maximum*

| | |
|---|---|
| **Description** | The maximum number of concurrently running transactions. If set to 0, limit is disabled. |
| **Valid values** | dbms.transaction.concurrent.maximum, an integer |
| **Default value** | `1000` |

## `dbms.transaction.monitor.check.interval`

*Table 423. dbms.transaction.monitor.check.interval*

| | |
|---|---|
| Description | Configures the time interval between transaction monitor checks. Determines how often monitor thread will check transaction for timeout. |
| Valid values | dbms.transaction.monitor.check.interval, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `2s` |

## `dbms.transaction.sampling.percentage`

Dynamic

*Table 424. dbms.transaction.sampling.percentage*

| | |
|---|---|
| Description | Transaction sampling percentage. |
| Valid values | dbms.transaction.sampling.percentage, an integer which is in the range `1` to `100` |
| Default value | `5` |

## `dbms.transaction.timeout`

Dynamic

*Table 425. dbms.transaction.timeout*

| | |
|---|---|
| Description | The maximum time interval of a transaction within which it should be completed. |
| Valid values | dbms.transaction.timeout, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `0s` |

## `dbms.transaction.tracing.level`

Dynamic

*Table 426. dbms.transaction.tracing.level*

| | |
|---|---|
| Description | Transaction creation tracing level. |
| Valid values | dbms.transaction.tracing.level, one of [DISABLED, SAMPLE, ALL] |
| Default value | `DISABLED` |

## `dbms.rest.transaction.idle_timeout`

*Table 427. dbms.rest.transaction.idle_timeout*

| | |
|---|---|
| Description | Timeout for idle transactions in the REST endpoint. |
| Valid values | dbms.rest.transaction.idle_timeout, a duration (Valid units are: `ns`, `µs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) |
| Default value | `1m` |

# 7.10.17. Transaction log settings

## `dbms.recovery.fail_on_missing_files`

*Table 428. dbms.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** | dbms.recovery.fail_on_missing_files, a boolean |
| **Default value** | `true` |

## `dbms.tx_log.buffer.size`

*Table 429. dbms.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 buffer will be full. |
| **Valid values** | dbms.tx_log.buffer.size, a long which is minimum `131072` |
| **Default value** | `By default the size of byte buffer is based on number of available cpu's with minimal buffer size of 512KB. Every another 4 cpu's will add another 512KB into the buffer size. Maximal buffer size in this default scheme is 4MB taking into account that we can have one transaction log writer per database in multi-database env.For example, runtime with 4 cpus will have buffer size of 1MB; runtime with 8 cpus will have buffer size of 1MB 512KB; runtime with 12 cpus will have buffer size of 2MB.` |

## `dbms.tx_log.preallocate`

Dynamic

*Table 430. dbms.tx_log.preallocate*

| | |
|---|---|
| **Description** | Specify if Neo4j should try to preallocate the logical log file in advance. It optimizes the filesystem by ensuring there is room to accommodate newly generated files and avoid file-level fragmentation. |
| **Valid values** | dbms.tx_log.preallocate, a boolean |
| **Default value** | `true` |

## `dbms.tx_log.rotation.retention_policy`

Dynamic

*Table 431. dbms.tx_log.rotation.retention_policy*

| | |
|---|---|
| **Description** | Tell Neo4j how long logical transaction logs should be kept to backup the database.For example, "10 days" will prune logical logs that only contain transactions older than 10 days.Alternatively, "100k txs" will keep the 100k latest transactions from each database and prune any older transactions. |

| | |
|---|---|
| Valid values | dbms.tx_log.rotation.retention_policy, a string which matches the pattern `^(true\|keep_all\|false\|keep_none\|(\d+[KkMmGg]?( (files\|size\|txs\|entries\|hours\|days))))$` (Must be `true` or `keep_all`, `false` or `keep_none`, or of format `<number><optional unit> <type>`. Valid units are `K`, `M` and `G`. Valid types are `files`, `size`, `txs`, `entries`, `hours` and `days`. For example, `100M size` will limit logical log space on disk to 100MB per database,and `200K txs` will limit the number of transactions kept to 200 000 per database.) |
| Default value | `7 days` |

## `dbms.tx_log.rotation.size`

Dynamic

*Table 432. dbms.tx_log.rotation.size*

| | |
|---|---|
| Description | Specifies at which file size the logical log will auto-rotate. Minimum accepted value is 128 KiB. |
| Valid values | dbms.tx_log.rotation.size, 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`, `PiB`, `PB`, `EiB`, `EB`) which is minimum `128.00KiB` |
| Default value | `250.00MiB` |

[1] Applicable to all operating systems where Neo4j Desktop is supported.
[2] For details about *neo4j.conf*, see The neo4j.conf file.
[3] The data directory is internal to Neo4j and its structure is subject to change between versions without notice.
[4] For more information, see APOC User Guide → Installation.
[5] To view *neo4j.log* in Docker, use `docker logs <containerID/name>`.
[6] To view the neo4j.log for Debian and RPM, use `journalctl --unit=neo4j`.
[7] When Neo4j is used in embedded mode, the default value is `false`.
[8] The default value for `dbms.connector.https.enabled` is `false`.

# Chapter 8. Manage databases

This chapter describes the following:


- Introduction
- Administration and configuration
- Queries
- Error handling
- Databases in a Causal Cluster
- Connecting remote databases


## 8.1. Introduction

### 8.1.1. Concepts

With Neo4j 4.4 you can create and use more than one active database at the same time.

*DBMS*

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 Causal Cluster.

*Instance*

A Neo4j instance is a Java process that is running the Neo4j server code.

*Transaction domain*

A transaction domain is a collection of graphs that can be updated within the context of a single transaction.

*Execution context*

An execution context is a runtime environment for the execution of a request. In practical terms, a request may be a query, a transaction, or an internal function or procedure.

*Database*

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. In logical terms, a database is a container for one or more graphs.

A database defines a *transaction domain* and an *execution context*. 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.

A default installation of Neo4j 4.4 contains two databases:

- `system` - the system database, containing metadata on the DBMS and security configuration.
- `neo4j` - the default database, a single database for user data. This has a default name of `neo4j`. A

different name can be configured before starting Neo4j for the first time.

*Graph*

This is a data model within a database. In Neo4j 4.0 there is only one graph within each database, and many administrative commands that refer to a specific graph do so using the database name.

In Neo4j Fabric, it is possible to refer to multiple graphs within the same transaction and Cypher query.

The following image illustrates a default installation, including the `system` database and a single database named `neo4j` for user data:

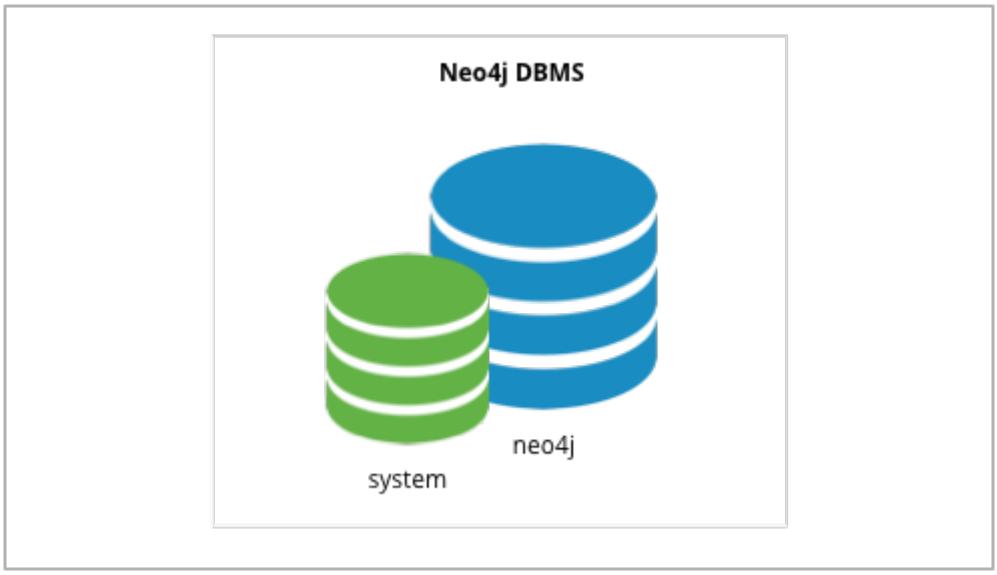



*Figure 2. A default Neo4j installation.*

*Editions*

**Note:**

The edition of Neo4j determines the number of possible databases:

- Installations of Community Edition can have exactly **one** user database.
- Installations of Enterprise Edition can have any number of user databases.

All installations include the `system` database.

## 8.1.2. The `system` database

All installations include a built-in database named `system`, which contains meta-data 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 functions, as described in detail in Cypher Manual → Database management.

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.

Security administration is described in detail in Cypher Manual → Access Control .

The following image illustrates an installation of Neo4j with multiple active databases, named `marketing`, `sales`, and `hr`:

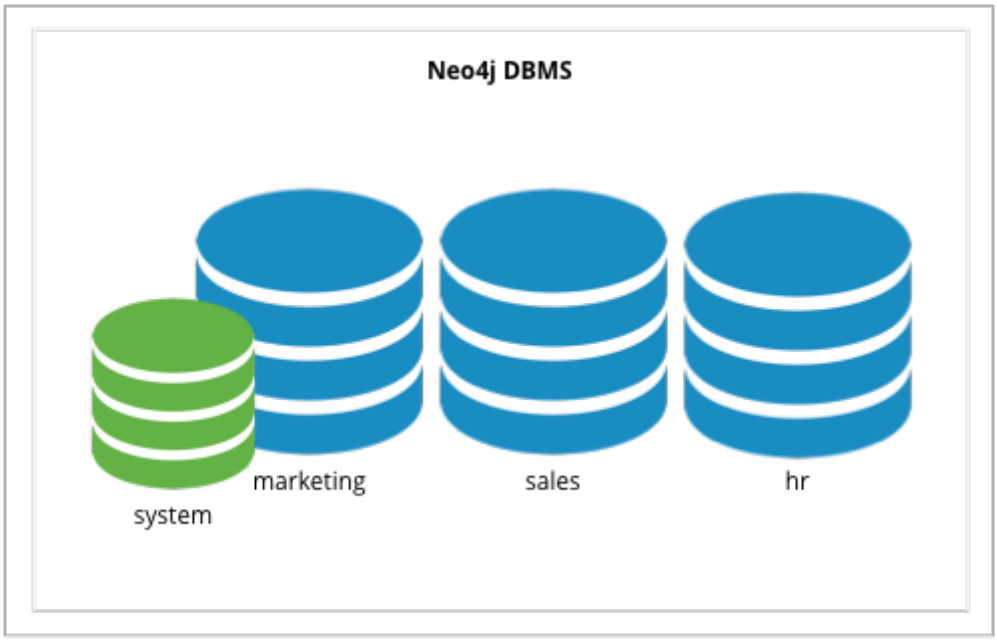



*Figure 3. A multiple database Neo4j installation.*

## 8.1.3. The default and home database

If a user connects to Neo4j without specifying a database, they will be connected to a home database. When choosing a home database the server will first use the home database configured for that user. If the connecting user does not have a home database configured, the server will use the default database, which every Neo4j instance has.

The default database is configurable. For details, see configuration parameters.

The following image illustrates an installation of Neo4j containing the three databases for user data, named `marketing`, `sales` and `hr`, and the `system` database. The default database is `sales`:

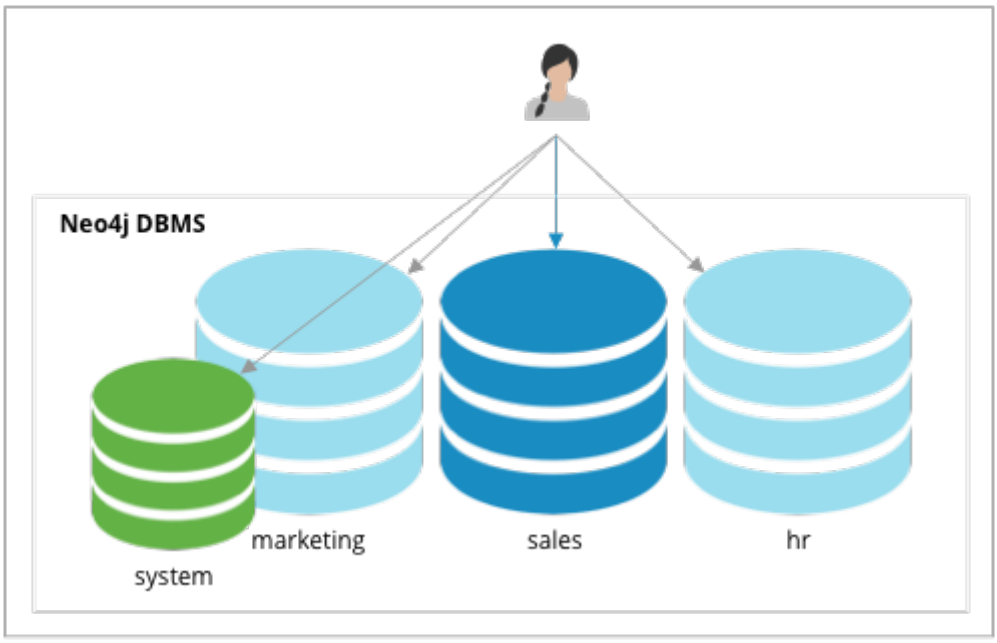



*Figure 4. A multiple database Neo4j installation, with a default database.*

### 8.1.4. Per-user home databases

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 Cypher Manual → User Management.

## 8.2. Administration and configuration

### 8.2.1. Administrative commands

> **Note:**
>
> Administrative commands should not be used during a rolling upgrade. For more information, see Upgrade and Migration Guide → Upgrade a Causal Cluster.
>
> For detailed information on Cypher administrative commands, see Cypher Manual → Database management.

Before using administrative commands, it is important to understand the difference between stopped databases, and dropped databases:

- Databases that are stopped with the `STOP` command are completely shutdown, and may be started again through the `START` command. In a Causal 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.

- Dropped databases are completely removed and are not intended to be used again at all.

The following Cypher commands are used on the `system` database to manage multiple databases:

| Command | Description |
|---|---|
| `CREATE DATABASE name` | Create and start a new database. Enterprise Edition |
| `DROP DATABASE name` | Drop (remove) an existing database. Enterprise Edition |
| `ALTER DATABASE name` | Alter (modify) an existing database. Enterprise Edition |
| `START DATABASE name` | Start a database that has been stopped. |
| `STOP DATABASE name` | Shut down a database. |
| `SHOW DATABASE name` | Show the status of a specific database. |
| `SHOW DATABASES` | Show the name and status of all the databases. |
| `SHOW DEFAULT DATABASE` | Show the name and status of the default database. |
| `SHOW HOME DATABASE` | Show the name and status of the home database for the current user. |

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 are case-insensitive and normalized to lowercase.
- Names that begin with an underscore and with the prefix `system` are reserved for internal use.



**Note:**

All of the above commands are executed as Cypher commands, and the database name is subject to the standard Cypher restrictions on valid identifiers. In particular, the `-` (dash) and `.` (dot) characters are not legal in Cypher variables, and therefore names with dashes must be enclosed within back-ticks. For example, `CREATE DATABASE \`main-db\``. Database names are the only identifier for which dots don't need to be escaped. For example, `main.db` is a valid database name.

It is possible to create an alias to refer to an existing database to avoid these restrictions. For more information, see Cypher Manual → Creating database aliases.

For detailed information on Cypher administrative commands, see Cypher Manual → Access Control.

For examples of using the Cypher administrative commands to manage multiple active databases, see Queries.

## 8.2.2. Configuration parameters

Configuration parameters are defined in the neo4j.conf file.

The following configuration parameters are applicable for managing databases:

| Parameter name | Description |
| --- | --- |
| `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.<br><br>**Default value:** `neo4j`<br><br>**Note:**<br><br>In a clustered setup, the value of `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. |
| `dbms.max_databases` | Maximum number of databases that can be used in a Neo4j single instance or Causal Cluster. The number includes all the online and offline databases. The value is an integer with a minimum value of 2. Enterprise Edition<br><br>**Default value:** `100`<br><br>**Note:**<br><br>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. |
| `dbms.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.<br><br>**Default value:** `false` |

| Parameter name | Description |
| --- | --- |
| `dbms.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.<br><br>**Note:**<br><br>Regardless of settings of `dbms.databases.default_to_read_only`, `dbms.databases.read_only` and `dbms.databases.writable` the `system` database will never will be read-only and will always accept write queries.<br><br>**Note:**<br><br>Another way of preventing writes is to set the database access to read-only using the ALTER DATABASE command.<br><br>Example configuration:<br><br>`dbms.databases.read_only=["foo", "bar"]` |
| `dbms.databases.writable` | List of database names for which to accept write queries. This set can contain also not yet existing databases.<br>The value of this setting is ignored if `dbms.databases.default_to_read_only` is set to `false`.<br>If a database name is present in both sets, the database will be read-only and prevent write queries.<br><br>**Tip:**<br><br>If most of your databases would read-only with a few exceptions, it can be easier to set `config_dbms.databases.default_to_read_only` to `true`, and then put the names of the non read-only databases into `dbms.databases.writeable`.<br><br>Example configuration:<br><br>`dbms.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 `dbms.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.

## 8.3. Queries

**Tip:**

For detailed information on Cypher administrative commands, see Cypher Manual → Database management.

**Note:**

All commands and example queries in this section are run in the Neo4j Cypher Shell command-line interface (CLI).

Note that the `cypher-shell` queries are not case-sensitive, but must end with a semicolon.

### 8.3.1. Show the status of a specific database

*Example 35.* `SHOW DATABASE`

```
neo4j@system> SHOW DATABASE neo4j;
```

In standalone mode:

```
+-------------------------------------------------------------------------------------------------------
--------------------------+
| name    | aliases | access       | address          | role         | requestedStatus |
currentStatus | error | default | home  |
+-------------------------------------------------------------------------------------------------------
--------------------------+
| "neo4j" | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""    | TRUE    | TRUE  |
+-------------------------------------------------------------------------------------------------------
--------------------------+

1 row available after 100 ms, consumed after another 6 ms
```

Or in a Causal Cluster:

```
+-------------------------------------------------------------------------------------------------------
--------------------------+
| name    | aliases | access       | address          | role       | requestedStatus | currentStatus
| error | default   | home  |
+-------------------------------------------------------------------------------------------------------
--------------------------+
| "neo4j" | []      | "read-write" | "localhost:7687" | "leader"   | "online"        | "online"
| ""    | TRUE      | TRUE  |
| "neo4j" | []      | "read-write" | "localhost:7688" | "follower" | "online"        | "online"
```

```
| ""     | TRUE      | TRUE  |
| "neo4j" | []      | "read-write" | "localhost:7689" | "follower" | "online"        | "online"
| ""     | TRUE      | TRUE  |
+-------------------------------------------------------------------------------------------------------
---------------------------+

3 row available after 100 ms, consumed after another 6 ms
```

## 8.3.2. Show the status of all databases

*Example 36.* `SHOW DATABASES`

```
neo4j@system> SHOW DATABASES;
```

In standalone mode:

```
+-------------------------------------------------------------------------------------------------------
----------------------------+
| name     | aliases | access       | address          | role         | requestedStatus |
currentStatus | error | default | home  |
+-------------------------------------------------------------------------------------------------------
----------------------------+
| "neo4j"  | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""     | TRUE    | TRUE  |
| "system" | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""     | FALSE   | FALSE |
+-------------------------------------------------------------------------------------------------------
----------------------------+

2 rows available after 5 ms, consumed after another 1 ms
```

Or in a Causal Cluster:

```
+-------------------------------------------------------------------------------------------------------
----------------------------+
| name     | aliases | access       | address          | role       | requestedStatus | currentStatus
| error | default   | home  |
+-------------------------------------------------------------------------------------------------------
----------------------------+
| "neo4j"  | []      | "read-write" | "localhost:7687" | "leader"   | "online"        | "online"
| ""     | TRUE      | TRUE  |
| "neo4j"  | []      | "read-write" | "localhost:7688" | "follower" | "online"        | "online"
| ""     | TRUE      | TRUE  |
| "neo4j"  | []      | "read-write" | "localhost:7689" | "follower" | "online"        | "online"
| ""     | TRUE      | TRUE  |
| "system" | []      | "read-write" | "localhost:7687" | "follower" | "online"        | "online"
| ""     | FALSE     | FALSE |
| "system" | []      | "read-write" | "localhost:7688" | "leader"   | "online"        | "online"
| ""     | FALSE     | FALSE |
| "system" | []      | "read-write" | "localhost:7689" | "follower" | "online"        | "online"
| ""     | FALSE     | FALSE |
+-------------------------------------------------------------------------------------------------------
----------------------------+

6 rows available after 5 ms, consumed after another 1 ms
```

Switching between `online` and `offline` states is achieved using the `START DATABASE` and `STOP DATABASE` commands.

## 8.3.3. Show the status of the default database

The config setting `dbms.default_database` defines which database is created and started by default when Neo4j starts. The default value of this setting is `neo4j`.

*Example 37.* `SHOW DEFAULT DATABASE`

```
neo4j@system> SHOW DEFAULT DATABASE;
```

In standalone mode:

```
+-----------------------------------------------------------------------------------------------------
----------+
| name    | aliases | access       | address          | role         | requestedStatus |
currentStatus | error |
+-----------------------------------------------------------------------------------------------------
----------+
| "neo4j" | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""    |
+-----------------------------------------------------------------------------------------------------
----------+

1 row available after 57 ms, consumed after another 2 ms
```

Or in a Causal Cluster:

```
+-----------------------------------------------------------------------------------------------------
----------+
| name    | aliases | access       | address          | role       | requestedStatus | currentStatus
| error   |
+-----------------------------------------------------------------------------------------------------
----------+
| "neo4j" | []      | "read-write" | "localhost:7687" | "follower" | "online"        | "online"
| ""      |
| "neo4j" | []      | "read-write" | "localhost:7688" | "leader"   | "online"        | "online"
| ""      |
| "neo4j" | []      | "read-write" | "localhost:7689" | "follower" | "online"        | "online"
| ""      |
+-----------------------------------------------------------------------------------------------------
----------+

3 row available after 57 ms, consumed after another 2 ms
```

You can change the default database by using `dbms.default_database`, and restarting the server.



**Note:**

In Community Edition, the default database is the only database available, other than the `system` database.

## 8.3.4. Create a database

*Example 38.* `CREATE DATABASE`

```
neo4j@system> CREATE DATABASE sales;
```

```
0 rows available after 108 ms, consumed after another 0 ms
```

```
neo4j@system> SHOW DATABASES;
```

In standalone mode:

```
+-------------------------------------------------------------------------------------------------------
----------------------------+
| name     | aliases | access       | address          | role         | requestedStatus |
currentStatus | error | default | home  |
+-------------------------------------------------------------------------------------------------------
----------------------------+
| "neo4j"  | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""     | TRUE    | TRUE  |
| "system" | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""     | FALSE   | FALSE |
| "sales"  | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""     | FALSE   | FALSE |
+-------------------------------------------------------------------------------------------------------
----------------------------+

3 rows available after 4 ms, consumed after another 1 ms
```

Or in a Causal Cluster:

```
+-------------------------------------------------------------------------------------------------------
----------------------------+
| name     | aliases | access       | address          | role       | requestedStatus | currentStatus
| error | default   | home  |
+-------------------------------------------------------------------------------------------------------
----------------------------+
| "neo4j"  | []      | "read-write" | "localhost:7687" | "leader"   | "online"        | "online"
| ""     | TRUE      | TRUE  |
| "neo4j"  | []      | "read-write" | "localhost:7688" | "follower" | "online"        | "online"
| ""     | TRUE      | TRUE  |
| "neo4j"  | []      | "read-write" | "localhost:7689" | "follower" | "online"        | "online"
| ""     | TRUE      | TRUE  |
| "system" | []      | "read-write" | "localhost:7687" | "follower" | "online"        | "online"
| ""     | FALSE     | FALSE |
| "system" | []      | "read-write" | "localhost:7688" | "leader"   | "online"        | "online"
| ""     | FALSE     | FALSE |
| "system" | []      | "read-write" | "localhost:7689" | "follower" | "online"        | "online"
| ""     | FALSE     | FALSE |
| "sales"  | []      | "read-write" | "localhost:7687" | "follower" | "online"        | "online"
| ""     | FALSE     | FALSE |
| "sales"  | []      | "read-write" | "localhost:7688" | "follower" | "online"        | "online"
| ""     | FALSE     | FALSE |
| "sales"  | []      | "read-write" | "localhost:7689" | "leader"   | "online"        | "online"
| ""     | FALSE     | FALSE |
+-------------------------------------------------------------------------------------------------------
----------------------------+

9 rows available after 4 ms, consumed after another 1 ms
```

## 8.3.5. Switch a database

*Example 39.* `:use <database-name>`

```
neo4j@system> :use sales
neo4j@sales>
```

## 8.3.6. Create or replace a database

*Example 40.* `CREATE OR REPLACE DATABASE`

```
neo4j@sales> match (n) return count(n) as countNode;
```

```
+-----------+
| countNode |
+-----------+
| 115       |
+-----------+

1 row available after 12 ms, consumed after another 0 ms
```

```
neo4j@system> CREATE OR REPLACE DATABASE sales;
```

```
0 rows available after 64 ms, consumed after another 0 ms
```

```
neo4j@system> SHOW DATABASES;
```

In standalone mode:

```
+-----------------------------------------------------------------------------------------------------------
----------------------------+
| name     | aliases | access       | address          | role         | requestedStatus |
currentStatus | error | default | home  |
+-----------------------------------------------------------------------------------------------------------
----------------------------+
| "neo4j"  | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""     | TRUE    | TRUE  |
| "system" | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""     | FALSE   | FALSE |
| "sales"  | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""     | FALSE   | FALSE |
+-----------------------------------------------------------------------------------------------------------
----------------------------+

3 rows available after 2 ms, consumed after another 2 ms
```

Or in a Causal Cluster:

```
+-----------------------------------------------------------------------------------------------------------
----------------------------+
| name     | aliases | access       | address          | role       | requestedStatus | currentStatus
| error | default   | home  |
+-----------------------------------------------------------------------------------------------------------
----------------------------+
| "neo4j"  | []      | "read-write" | "localhost:7687" | "leader"   | "online"        | "online"
| ""     | TRUE      | TRUE  |
| "neo4j"  | []      | "read-write" | "localhost:7688" | "follower" | "online"        | "online"
| ""     | TRUE      | TRUE  |
| "neo4j"  | []      | "read-write" | "localhost:7689" | "follower" | "online"        | "online"
| ""     | TRUE      | TRUE  |
| "system" | []      | "read-write" | "localhost:7687" | "follower" | "online"        | "online"
| ""     | FALSE     | FALSE |
| "system" | []      | "read-write" | "localhost:7688" | "leader"   | "online"        | "online"
| ""     | FALSE     | FALSE |
| "system" | []      | "read-write" | "localhost:7689" | "follower" | "online"        | "online"
| ""     | FALSE     | FALSE |
| "sales"  | []      | "read-write" | "localhost:7687" | "follower" | "online"        | "online"
```

```
| ""    | FALSE     | FALSE |
| "sales"  | []      | "read-write" | "localhost:7688" | "follower" | "online"        | "online"
| ""    | FALSE     | FALSE |
| "sales"  | []      | "read-write" | "localhost:7689" | "leader"   | "online"        | "online"
| ""    | FALSE     | FALSE |
+-------------------------------------------------------------------------------------------------------
----------------------------+

9 rows available after 2 ms, consumed after another 2 ms
```

```
neo4j@system> :use sales
neo4j@sales> match (n) return count(n) as countNode;
```

```
+-----------+
| countNode |
+-----------+
| 0         |
+-----------+

1 row available after 15 ms, consumed after another 1 ms
```

## 8.3.7. Stop a database

*Example 41.* `STOP DATABASE`

```
neo4j@system> STOP DATABASE sales;
```

```
0 rows available after 18 ms, consumed after another 6 ms
```

```
neo4j@system> SHOW DATABASES;
```

In standalone mode:

```
+-------------------------------------------------------------------------------------------------------
----------------------------+
| name     | aliases | access       | address          | role         | requestedStatus |
currentStatus | error | default | home  |
+-------------------------------------------------------------------------------------------------------
----------------------------+
| "neo4j"  | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""    | TRUE    | TRUE  |
| "system" | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""    | FALSE   | FALSE |
| "sales"  | []      | "read-write" | "localhost:7687" | "standalone" | "offline"       | "offline"
| ""    | FALSE   | FALSE |
+-------------------------------------------------------------------------------------------------------
----------------------------+

3 rows available after 2 ms, consumed after another 1 ms
```

Or in a Causal Cluster:

```
+-------------------------------------------------------------------------------------------------------
----------------------------+
| name     | aliases | access       | address          | role       | requestedStatus | currentStatus
| error | default   | home  |
+-------------------------------------------------------------------------------------------------------
```

```
----------------------------+
| "neo4j"  | []      | "read-write" | "localhost:7687" | "leader"   | "online"        | "online"
| ""    | TRUE      | TRUE  |
| "neo4j"  | []      | "read-write" | "localhost:7688" | "follower" | "online"        | "online"
| ""    | TRUE      | TRUE  |
| "neo4j"  | []      | "read-write" | "localhost:7689" | "follower" | "online"        | "online"
| ""    | TRUE      | TRUE  |
| "system" | []      | "read-write" | "localhost:7687" | "follower" | "online"        | "online"
| ""    | FALSE     | FALSE |
| "system" | []      | "read-write" | "localhost:7688" | "leader"   | "online"        | "online"
| ""    | FALSE     | FALSE |
| "system" | []      | "read-write" | "localhost:7689" | "follower" | "online"        | "online"
| ""    | FALSE     | FALSE |
| "sales"  | []      | "read-write" | "localhost:7687" | "unknown"  | "offline"       | "offline"
| ""    | FALSE     | FALSE |
| "sales"  | []      | "read-write" | "localhost:7688" | "unknown"  | "offline"       | "offline"
| ""    | FALSE     | FALSE |
| "sales"  | []      | "read-write" | "localhost:7689" | "unknown"  | "offline"       | "offline"
| ""    | FALSE     | FALSE |
+-----------------------------------------------------------------------------------------------------------
----------------------------+

9 rows available after 2 ms, consumed after another 1 ms
```

```
neo4j@system> :use sales
```

```
Unable to get a routing table for database 'sales' because this database is unavailable
neo4j@sales[UNAVAILABLE]>
```

## 8.3.8. Start a database

*Example 42.* `START DATABASE`

```
neo4j@sales[UNAVAILABLE]> :use system
neo4j@system> START DATABASE sales;
```

```
0 rows available after 21 ms, consumed after another 1 ms
```

```
neo4j@system> SHOW DATABASES;
```

In standalone mode:

```
+-----------------------------------------------------------------------------------------------------------
----------------------------+
| name     | aliases | access       | address          | role         | requestedStatus |
currentStatus | error | default | home  |
+-----------------------------------------------------------------------------------------------------------
----------------------------+
| "neo4j"  | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""    | TRUE    | TRUE  |
| "system" | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""    | FALSE   | FALSE |
| "sales"  | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""    | FALSE   | FALSE |
+-----------------------------------------------------------------------------------------------------------
----------------------------+

3 rows available after 2 ms, consumed after another 1 ms
```

Or in a Causal Cluster:

```
+-----------------------------------------------------------------------------------------------------------
----------------------------+
| name     | aliases | access       | address          | role       | requestedStatus | currentStatus
| error | default   | home  |
+-----------------------------------------------------------------------------------------------------------
----------------------------+
| "neo4j"  | []      | "read-write" | "localhost:7687" | "leader"   | "online"        | "online"
| ""    | TRUE      | TRUE  |
| "neo4j"  | []      | "read-write" | "localhost:7688" | "follower" | "online"        | "online"
| ""    | TRUE      | TRUE  |
| "neo4j"  | []      | "read-write" | "localhost:7689" | "follower" | "online"        | "online"
| ""    | TRUE      | TRUE  |
| "system" | []      | "read-write" | "localhost:7687" | "follower" | "online"        | "online"
| ""    | FALSE     | FALSE |
| "system" | []      | "read-write" | "localhost:7688" | "leader"   | "online"        | "online"
| ""    | FALSE     | FALSE |
| "system" | []      | "read-write" | "localhost:7689" | "follower" | "online"        | "online"
| ""    | FALSE     | FALSE |
| "sales"  | []      | "read-write" | "localhost:7687" | "follower" | "online"        | "online"
| ""    | FALSE     | FALSE |
| "sales"  | []      | "read-write" | "localhost:7688" | "follower" | "online"        | "online"
| ""    | FALSE     | FALSE |
| "sales"  | []      | "read-write" | "localhost:7689" | "leader"   | "online"        | "online"
| ""    | FALSE     | FALSE |
+-----------------------------------------------------------------------------------------------------------
----------------------------+

9 rows available after 2 ms, consumed after another 1 ms
```

## 8.3.9. Drop or remove a database

*Example 43.* `DROP DATABASE`

```
neo4j@system> DROP DATABASE sales;
```

```
0 rows available after 82 ms, consumed after another 1 ms
```

```
neo4j@system> SHOW DATABASES;
```

```
+-----------------------------------------------------------------------------------------------------------
----------------------------+
| name     | aliases | access       | address          | role         | requestedStatus |
currentStatus | error | default | home  |
+-----------------------------------------------------------------------------------------------------------
----------------------------+
| "neo4j"  | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""    | TRUE    | TRUE  |
| "system" | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""    | FALSE   | FALSE |
+-----------------------------------------------------------------------------------------------------------
----------------------------+

2 rows available after 6 ms, consumed after another 0 ms
```

# 8.4. Error handling

When running the database management queries, such as `CREATE DATABASE`, it is possible to encounter errors.

## 8.4.1. Observing errors

Because database management operations are performed asynchronously, these errors may not returned immediately upon query execution. Instead, you must monitor the output of `SHOW DATABASE`; particularly the `error` and `currentStatus` columns.

*Example 44. 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   | aliases | access       | address          | role         | requestedStatus | currentStatus
| error                     | default | home  |
+-----------------------------------------------------------------------------------------------------------
-----------------------------------------------+
| "foo"  | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "dirty"
| "File system permissions" | FALSE   | FALSE |
+-----------------------------------------------------------------------------------------------------------
-----------------------------------------------+

1 rows available after 4 ms, consumed after another 1 ms
```

In a Causal Cluster:

```
+-----------------------------------------------------------------------------------------------------------
-------------------------------------------+
| name   | aliases | access       | address          | role       | requestedStatus | currentStatus |
error                     | default | home  |
+-----------------------------------------------------------------------------------------------------------
-------------------------------------------+
| "foo"  | []      | "read-write" | "localhost:7687" | "leader"   | "online"        | "online"      |
""                        | FALSE   | FALSE |
| "foo"  | []      | "read-write" | "localhost:7688" | "follower" | "online"        | "online"      |
""                        | FALSE   | FALSE |
| "foo"  | []      | "read-write" | "localhost:7689" | "follower" | "online"        | "dirty"       |
"File system permissions" | FALSE   | FALSE |
+-----------------------------------------------------------------------------------------------------------
-------------------------------------------+

3 row available after 100 ms, consumed after another 6 ms
```

## 8.4.2. 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 `error`

column in the `SHOW DATABASE` query results may vary significantly.

However, databases may only be in one of a select number of states:

| Current state | Description |
|---|---|
| `initial` | The database has not yet been created. |
| `online` | The database is running. |
| `offline` | The database is not running. |
| `store copying` | The database is currently being updated from another instance of Neo4j. |
| `dropped` | The database has been deleted. |
| `dirty` | This state implies an error has occurred. The database's underlying store files may be invalid. For more information, consult the server's logs. |
| `quarantined` | The database is effectively stopped and its state may not be changed until no longer quarantined. |
| `unknown` | This instance of Neo4j doesn't know the state of this database. |

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`.



Note:

While `dropped` is a valid database state, it is only transiently observable, as database records are removed from `SHOW DATABASE` results once the `DROP` operation is complete.

## 8.4.3. 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.

*Example 45. 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   | aliases | access       | address          | role         | requestedStatus | currentStatus
| error                     | default | home  |
+-------------------------------------------------------------------------------------------------------------
-----------------------------------------------+
| "foo"  | []      | "read-write" | "localhost:7687" | "standalone" | "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     | aliases | access       | address          | role         | requestedStatus |
currentStatus | error | default | home  |
+-------------------------------------------------------------------------------------------------------------
-----------------------------+
| "foo"    | []      | "read-write" | "localhost:7687" | "standalone" | "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 Cypher manual → Database management.

> **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.

## 8.4.4. 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. |

| Name | Type | Description |
| --- | --- | --- |
| `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  | aliases | access       | address          | role       | requestedStatus | currentStatus | error
| default | home  |
+-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------+
| "foo" | []      | "read-write" | "localhost:7688" | "unknown"  | "online"        | "quarantined" | "By
neo4j at 2020-10-15T15:10:41.348Z: No reason given" | FALSE   | FALSE |
| "foo" | []      | "read-write" | "localhost:7689" | "follower" | "online"        | "online"      | ""
| FALSE   | FALSE |
| "foo" | []      | "read-write" | "localhost:7687" | "leader"   | "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 that Neo4j instance 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.

# 8.5. Databases in a Causal Cluster

Multiple databases in a Causal Cluster are managed the same way as a single instance. Administrators can use the same Cypher commands described in Administrative commands to manage databases. This is based on two main principles:

- All databases are available on all members of a cluster - this applies to Core servers and Read Replicas.
- Administrative commands must be executed on the `system` database, on the Leader member of the cluster.

## 8.5.1. Change the default database

You can use the procedure `dbms.cluster.setDefaultDatabase("newDefaultDatabaseName")` to change the default database of a Causal cluster.

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. On the Leader member of the cluster, run `CALL dbms.cluster.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>`.

## 8.5.2. Run Cypher administrative commands from Cypher Shell on a Causal Cluster

For the following examples consider a Causal Cluster environment formed by 5 members, 3 Core servers, and 2 Read Replicas:

*Example 46. View the members of a Causal Cluster*

```
neo4j@neo4j> CALL dbms.cluster.overview();
```

```
+-----------------------------------------------------------------------------------------------------------------
-------------------------------------------------------+
| id       | addresses                                                                                          |
databases                  | groups |
+-----------------------------------------------------------------------------------------------------------------
-------------------------------------------------------+
| "8c...3d" | ["bolt://core01:7687", "http://core01:7474", "https://core01:7473"] | {neo4j:
"FOLLOWER", system: "FOLLOWER"}         | []     |
| "8f...28" | ["bolt://core02:7687", "http://core02:7474", "https://core02:7473"] | {neo4j: "LEADER",
system: "LEADER"}                   | []     |
| "e0...4d" | ["bolt://read01:7687", "http://read01:7474", "https://read01:7473"] | {neo4j:
"READ_REPLICA", system: "READ_REPLICA"}     | []     |
```

```
| "1a...64" | ["bolt://core03:7687", "http://core03:7474", "https://core03:7473"] | {neo4j:
"FOLLOWER", system: "FOLLOWER"}         | []     |
| "59...87" | ["bolt://read02:7687", "http://read02:7474", "https://read02:7473"] | {neo4j:
"READ_REPLICA", system: "READ_REPLICA"}     | []     |
+-------------------------------------------------------------------------------------------------------
-----------------------------------------------------+

5 rows available after 5 ms, consumed after another 0 ms
```

The leader is currently the instance exposing port `7681` for the `bolt` protocol, and `7471/7481` for the `http/https` protocol.

Administrators can connect and execute Cypher commands in the following ways:

*Example 47. Using the `bolt://` scheme to connect to the Leader:*

```
$ bin/cypher-shell -a bolt://localhost:7681 -d system -u neo4j -p neo4j1
```

```
Connected to Neo4j 4.0.0 at bolt://localhost:7681 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@system> SHOW DATABASES YIELD name, currentStatus AS status, default;
```

```
+-------------------------------+
| name     | status   | default |
+-------------------------------+
| "neo4j"  | "online" | TRUE    |
| "system" | "online" | FALSE   |
+-------------------------------+

2 rows available after 34 ms, consumed after another 0 ms
```

```
neo4j@system> CREATE DATABASE data001;
```

```
0 rows available after 378 ms, consumed after another 12 ms
Added 1 nodes, Set 4 properties, Added 1 labels
neo4j@system> SHOW DATABASES YIELD name, currentStatus AS status, default;
+-------------------------------+
| name      | status   | default |
+-------------------------------+
| "neo4j"   | "online" | TRUE    |
| "system"  | "online" | FALSE   |
| "data001" | "online" | FALSE   |
+-------------------------------+

3 rows available after 2 ms, consumed after another 1 ms
```

*Example 48. Using the `neo4j://` scheme to connect to any Core member:*

```
$ bin/cypher-shell -a neo4j://localhost:7683 -d system -u neo4j -p neo4j1
```

```
Connected to Neo4j 4.0.0 at neo4j://localhost:7683 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@system> SHOW DATABASES YIELD name, currentStatus AS status, default;
```

```
+--------------------------------+
| name      | status   | default |
+--------------------------------+
| "neo4j"   | "online" | TRUE    |
| "system"  | "online" | FALSE   |
| "data001" | "online" | FALSE   |
+--------------------------------+

3 rows available after 0 ms, consumed after another 0 ms
```

```
neo4j@system> CREATE DATABASE data002;
```

```
0 rows available after 8 ms, consumed after another 1 ms
Added 1 nodes, Set 4 properties, Added 1 labels
```

```
neo4j@system> SHOW DATABASES YIELD name, currentStatus AS status, default;
```

```
+--------------------------------+
| name      | status   | default |
+--------------------------------+
| "neo4j"   | "online" | TRUE    |
| "system"  | "online" | FALSE   |
| "data001" | "online" | FALSE   |
| "data002" | "online" | FALSE   |
+--------------------------------+

4 rows available after 33 ms, consumed after another 0 ms
```



The `neo4j://` scheme is the equivalent to the `bolt+routing:` scheme available in earlier versions of Neo4j, but it can be used seamlessly with a standalone and clustered DBMS.

## 8.6. Connecting remote databases

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 configuration can be done with Administration 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.

## 8.6.1. Setup example

In this example, *Alice* is an administrator and *Carol* is a user who needs access to a database managed by *Bob*:

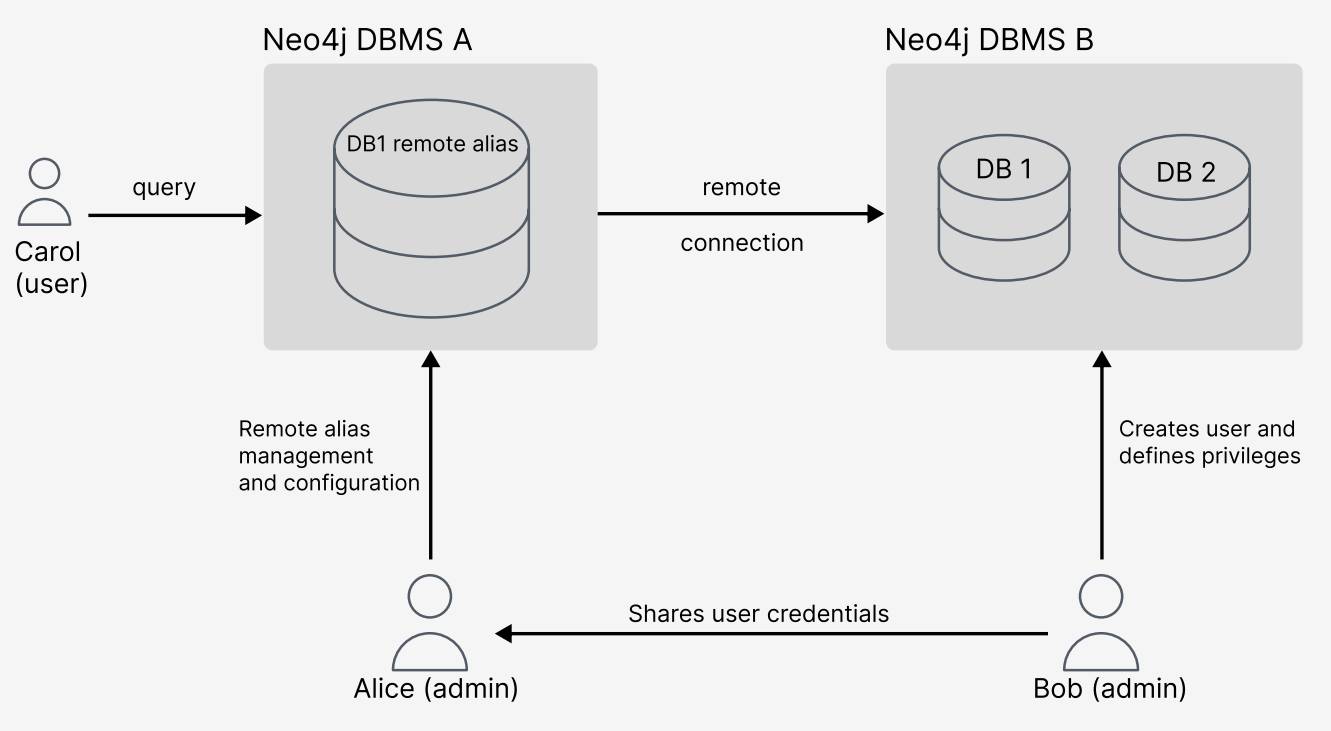



*Figure 5. Overview of the required remote database alias setup*

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 Cypher manual → DBMS administration 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 Cypher manual → Access control for further information).

## 8.6.2. Fabric vs remote alias database

Here is a comparison between a Fabric database and a remote alias database, so you can take a more informed decision about which one to use:

| | Fabric | Remote alias database |
|---|---|---|
| Authentication | Identical user credentials in both DBMSs. | Authenticates with defined user on remote DBMS for anyone using the alias. |
| Credentials | In memory. | Stored encrypted. |
| Configuration | Manually set per instance. | Secret key for encryption set per instance.<br>Created aliases are synchronized across cluster instances. |
| Restarting when creating or altering endpoints | Yes | No |

## 8.6.3. Configuration of 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.



**Note:**

If a remote alias target is a cluster, it is recommended to enable Server-side routing to handle leader changes.

```
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
dbms.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
dbms.connector.bolt.tls_level=REQUIRED
```

## 8.6.4. Configuration of 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.

### Set configuration

The following configuration is necessary for creating a remote database alias:

| Configuration | Description |
|---|---|
| `systemdb.secrets.keystore.path` | The absolute path to the keystore file, including the file name. |
| `systemdb.secrets.keystore.password` | The password to the keystore file. Use Command expansion to set the password. |
| `systemdb.secrets.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:

```
systemdb.secrets.keystore.path=/home/secure-folder/keystore-name.pkcs12
systemdb.secrets.keystore.password=$(conf/password.sh)
systemdb.secrets.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
```

## 8.6.5. Managing 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 like this:

```
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.

## 8.6.6. Changing 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.

## 8.6.7. User connection 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 `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.

# Chapter 9. Clustering

This chapter describes the following:

- Introduction — An overview of the Causal Clustering architecture.
- Deploy a cluster — The basics of configuring and deploying a new cluster.
- Seed a cluster — How to deploy a Causal Cluster with pre-existing data.
- Discovery — How members of a cluster discover each other.
- Intra-cluster encryption — How to secure the cluster communication.
- Internals — A few internals regarding the operation of the cluster.
- Settings reference — A summary of the most important Causal Cluster settings.
- Clustering glossary — A glossary of terms used in the clustering documentation.

Further information:

- For instructions on setting up clustering when running Neo4j in a Docker container, see Clustering on Docker.
- For an example of managing multiple databases in a cluster, see Multiple databases in a cluster.
- For instructions on how you to upgrade your Neo4j cluster, see Upgrade and Migration Guide → Upgrade a Causal Cluster.
- For a summary of the facilities that are available for monitoring a Neo4j cluster, see Monitoring (and specifically, Monitoring a cluster).
- For a tutorial on setting up a test cluster locally on a single machine, see Set up a local Causal Cluster.
- For advanced concepts, including the implementation of the Raft Protocol, see Advanced Causal Clustering

## 9.1. Introduction

### 9.1.1. Overview

Neo4j's Causal Clustering provides three main features:

1. **Safety:** Core Servers provide a fault tolerant platform for transaction processing which will remain available while a simple majority of those Core Servers are functioning.
2. **Scale:** Read Replicas provide a massively scalable platform for graph queries that enables very large graph workloads to be executed in a widely distributed topology.
3. **Causal consistency:** when invoked, a client application is guaranteed to read at least its own writes.

Together, this allows the end-user system to be fully functional and both read and write to the database in the event of multiple hardware and network failures and makes reasoning about database interactions straightforward.

In the remainder of this section we will provide an overview of how causal clustering works in production, including both operational and application aspects.

## 9.1.2. Operational view

From an operational point of view, it is useful to view the cluster as being composed of servers with two different roles, referred to as *Primary* and *Secondary* servers.

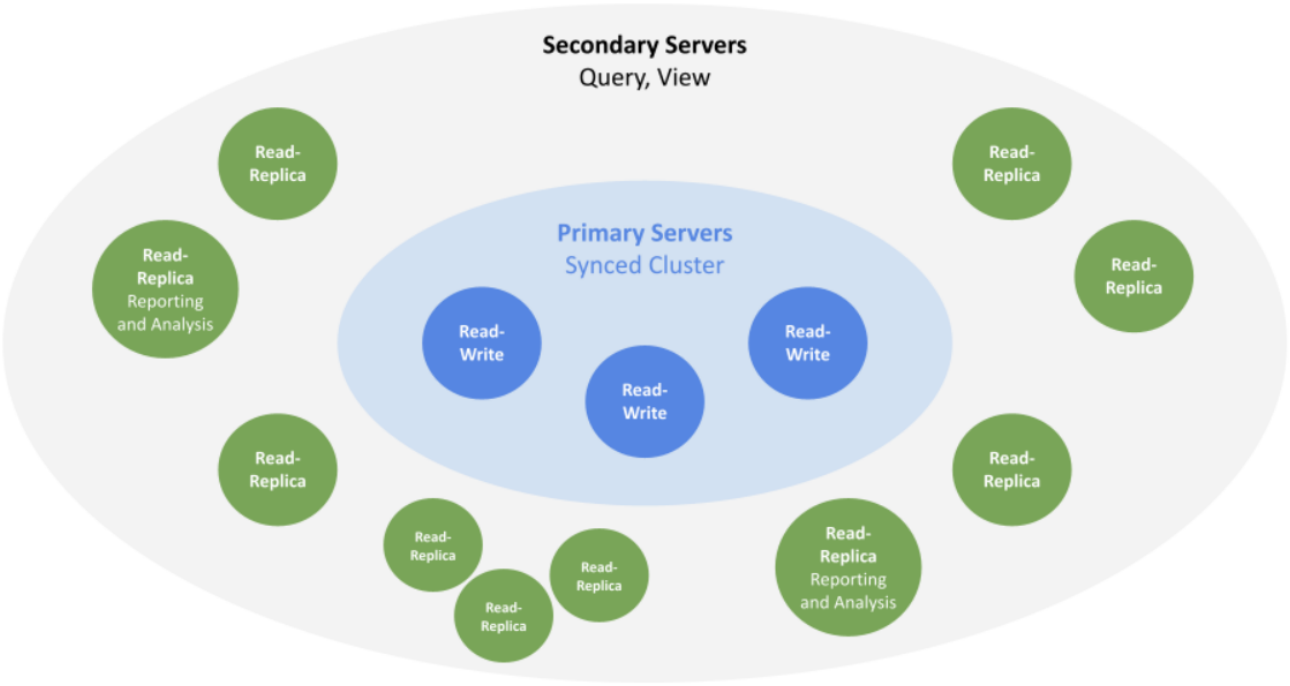



*Figure 6. Causal Cluster Architecture*

The two roles are foundational in any production deployment but are managed at different scales from one another and undertake different roles in managing the fault tolerance and scalability of the overall cluster.

## 9.1.3. Primary servers

The Primary servers are based on two types of instances:

- **Single instance** is an instance that operates without redundancy within the set of Primary servers and allows read and write operations. Redundancy is achieved by adding Secondary servers, which guarantee causal consistency but they **do not** safeguard data as Primary servers do. Therefore, clusters based on a Single instance as Primary server are good for read scalability, but they are not fault tolerant. If a fault occurs on the Single instance, there is a potential risk of data loss: it is the responsibility of the application or of the tooling around the cluster to eliminate or minimize such risk.
- **Core instance** is an instance that allows read and write operations and its main responsibility is to safeguard data. Core instances do so by replicating all transactions using the Raft protocol. Raft ensures that the data is safely durable before confirming a transaction commit to the end user application. In practice, this means once a majority of Core instances in a cluster (N/2+1) have accepted the transaction, it is safe to acknowledge the commit to the end user application.

  The safety requirement has an impact on write latency. Implicitly, writes are acknowledged by the fastest majority, but as the number of Core instances in the cluster grows, so does the size of the

majority needed to acknowledge a write.

In practice, this means that there are relatively few machines in a typical Core instance cluster, enough to provide sufficient fault tolerance for the specific deployment. This is calculated with the formula M = 2F + 1, where M is the number of Core instances required to tolerate F faults. For example:

- In order to tolerate two failed Core instances, you need to deploy a cluster of five Core instances.
- The smallest fault tolerant cluster, a cluster that can tolerate one fault, must have three Core instances.
- It is also possible to create a Causal Cluster consisting of only two Core instances. However, that cluster is not fault-tolerant. If one of the two servers fails, the remaining server becomes read-only.



**Note:**

With Core instances, should the cluster suffer enough Core failures, it can no longer process writes and becomes read-only to preserve safety.



**Note:**

In version 4.4 of Neo4j Causal Cluster, Primary servers cannot be mixed: either one Single instance is the Primary server or a set of Core instances are the Primary servers.

## 9.1.4. Secondary servers

In version 4.4 of Neo4j Causal Cluster, Secondary servers can only be one type of instance, called **Read Replica instances**.

Read Replica instances are asynchronously replicated from Primary Servers via transaction log shipping. They will periodically poll an upstream server for new transactions and have these shipped over. Many Read Replicas can be fed data from a relatively small number of Primary Servers, allowing for a large fan out of the query workload for scale.

Read Replica instances should typically be run in relatively large numbers and treated as disposable. Losing a Read Replica does not impact the cluster's availability, aside from the loss of its fraction of graph query throughput. It does not affect the fault tolerance capabilities of the cluster.



**Note:**

Read Replicas are read-only, but you should not create one (or convert the `dbms.mode` of a `CORE` to `READ_REPLICA`) if you simply desire a neo4j instance which temporarily cannot execute write queries. Instead you should use the `dbms.databases.default_to_read_only` config setting to prevent writes to databases on the instance in question.

The main responsibility of Read Replica instances is to scale out read workloads. Read Replica instances act like caches for the graph data and are fully capable of executing arbitrary (read-only) queries and procedures.

When the Primary server is a Single instance, Secondary servers may be part of a Disaster Recovery strategy. Due to its asynchronous nature, Read Replica instances may not provide all transactions committed on the Primary server, but they may be set as a new Primary server in case the Single instance is no longer available. The change of a Read Replica instance into a Single instance is a manual operation that must be executed by a Database Administrator or by some tooling and it requires careful checks, in order to identify the most up-to-date instance and the status of the other instances.

### 9.1.5. 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 will 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 Core Servers (where data is safe) and read those writes from a Read Replica (where graph operations are scaled out). For example, causal consistency guarantees that the write which created a user account will be present when that same user subsequently attempts to log in.

*Figure 7. Causal Cluster setup with causal consistency via Neo4j drivers*

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 Core Servers and Read Replicas to provide high quality of service.

Since Neo4j clusters are causally consistent, in the remainder of this chapter, the terms *causal cluster* or *cluster* are used to denote Neo4j installations consisting of primary and secondary servers.

## 9.2. Deploy a cluster

### 9.2.1. Introduction

This section describes how to set up a new cluster. Two scenarios are covered:

1. A four-instance cluster with one Single instance as Primary server and three Read Replica instances as Secondary servers. This scenario is ideal for reporting and analytical workloads.
2. A three-instance cluster with three Core instances as Primary servers. This scenario is ideal for transactional workloads.

Additionally, the process to turn a Secondary server into a standalone instance by detaching it from an existing cluster is also described.

## 9.2.2. Configure a cluster with Single and Read Replica instances

The following configuration settings are important to consider when deploying a new cluster with a Single instance as a Primary server. See also Settings reference for more detailed descriptions and examples.



**Important:**

This configuration is optimized for best scalability and it is recommended to be used for reporting and analytical workloads. Clusters configured in this way do not provide automatic failover and fault tolerance. In case of fault, if a cluster is not supported by appropriate external tooling, data may be lost.



**Note:**

In the current version of Neo4j, the clustering-related parameters use the `causal_clustering` namespace. This will be replaced with a more suitable namespace in an upcoming release.

*Table 433. Important settings for clusters with Single instance as Primary server*

| Option name | Servers | Description |
|---|---|---|
| `dbms.default_advertised_address` | All (Primary and Secondary) | 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. |
| `dbms.mode` | Primary | The operating mode of the server instance. The Primary server is set as `SINGLE`. |
| | Secondary | The operating mode of the server instance. The Secondary servers are set as `READ_REPLICA`. |
| `dbms.clustering.enable=true` | Primary | Allows a single instance to form a cluster and is only evaluated when `dbms.mode=SINGLE`. |

| Option name | Servers | Description |
| --- | --- | --- |
| `causal_clustering.initial_discovery_members` | Secondary | This setting needs to be specified on Read Replica instances and contains the network address for at least the primary instance, but can also include Secondary servers. This parameter must be set to the same value on all cluster members. The behavior of this setting can be modified by configuring the setting `causal_clustering.discovery_type`. This is described in detail in Discovery. |

The following example shows how to set up a cluster with a Single instance as Primary server and three Read Replica instances as Secondary servers.

*Example 49. Configure a cluster with a Single instance as Primary server*

In this example, one Primary server named single.example.com and three Secondary servers, `read_replica01.example.com`, `read_replica02.example.com` and `read_replica03.example.com` are configured. All instances have Neo4j Enterprise Edition installed. To form a cluster, the *neo4j.conf* needs to be configured on each server. The Primary server, set as Single instance, is configured as such:

*neo4j.conf on single.example.com:*

```
dbms.mode=SINGLE
dbms.clustering.enable=true
dbms.default_advertised_address=single.example.com
```

The *neo4j.conf* on the Secondary servers, set as Read Replica instances, is identical across all instances:

*neo4j.conf on read_replica01.example.com, read_replica02.example.com and read_replica03.example.com:*

```
dbms.mode=READ_REPLICA
dbms.default_advertised_address=read_replica<xx>.example.com
causal_clustering.initial_discovery_members=single.example.com:5000
```

Once all *neo4j.conf* files have been configured, the instances can be started and the cluster is ready. After the cluster has started, it is possible to connect to any of the instances and run `CALL dbms.cluster.overview()` to check the status of the cluster. This shows information about each member of the cluster:

```
CALL dbms.cluster.overview();
```

```
+----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------+
| id                                     | addresses
| databases                                          | groups |
+----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------+
| "8e4133d7-4de1-469e-88ac-864571cb0a92" | ["bolt://read_replica1.example.com:7687",
```

```
"http://read_replica1.example.com:7474"] | {neo4j: "READ_REPLICA", system: "READ_REPLICA"} | []     |
| "eb6a4e88-9a5f-405b-b230-5bbbd681ec9e" | ["bolt://read_replica2.example.com:7687",
"http://read_replica2.example.com:7474"] | {neo4j: "READ_REPLICA", system: "READ_REPLICA"} | []     |
| "274e36db-d96f-4736-8a99-68851b1bbb0b" | ["bolt://read_replica3.example.com:7687",
"http://read_replica3.example.com:7474"] | {neo4j: "READ_REPLICA", system: "READ_REPLICA"} | []     |
| "6fd05bc6-760e-4644-bf02-05117a5d777d" | ["bolt://single.example.com:7687",
"http://single.example.com:7474"]              | {neo4j: "LEADER", system: "LEADER"}             |
[]     |
+-------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
4 rows available after 8 ms, consumed after another 3 ms
```

## 9.2.3. Configure a cluster with Core instances

The following configuration settings are important to consider when deploying a new cluster with Core instances as Primary servers. See also Settings reference for more detailed descriptions and examples.



**Important:**

This configuration is optimized for fault tolerance, automatic failover and best scalability, and it is recommended to be used for transactional workloads. In many cases and when they are correctly configured, these clusters safeguard data and they do not require any particular external tooling.

*Table 434. Important settings for clusters with Core instances as Primary servers*

| Option name | Servers | Description |
|---|---|---|
| `dbms.default_listen_address` | All (Primary and Secondary) | 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.default_advertised_address` | All (Primary and Secondary) | 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. |
| `dbms.mode` | Primary | The operating mode of the server instance. The Primary servers are set as `CORE`. |
| | Secondary | The operating mode of the server instance. The Secondary servers are set as `READ_REPLICA`. |
| `causal_clustering.minimum_core_cluster_size_at_formation` | Primary | The minimum number of Core instances in the cluster at formation. A cluster will not form without the number of Cores defined by this setting, and this should in general be configured to the full and fixed amount. |
| `causal_clustering.minimum_core_cluster_size_at_runtime` | Primary | The minimum number of Core instances which will exist in the consensus group. |

| Option name | Servers | Description |
| --- | --- | --- |
| `causal_clustering.initial_discovery_members` | All (Primary and Secondary) | The network addresses of an initial set of Core cluster members that are available to bootstrap this Core or Read Replica instance. 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 Core Servers.<br><br>The behavior of this setting can be modified by configuring the setting `causal_clustering.discovery_type`. This is described in detail in Discovery. |



**Caution:**

**Listen configuration**

Listening on 0.0.0.0 makes the ports publicly available. Make sure you understand the security implications and strongly consider setting up encryption.

The following example shows how to set up a simple cluster with three Core servers:

*Example 50. Configure a Core-only cluster*

In this example, three Core instances named `core01.example.com`, `core02.example.com` and `core03.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 `dbms.default_advertised_address`:

*neo4j.conf on core01.example.com:*

```
dbms.default_listen_address=0.0.0.0
dbms.default_advertised_address=core01.example.com
dbms.mode=CORE
causal_clustering.initial_discovery_members=core01.example.com:5000,core02.example.com:5000,core03.ex
ample.com:5000
```

*neo4j.conf on core02.example.com:*

```
dbms.default_listen_address=0.0.0.0
dbms.default_advertised_address=core02.example.com
dbms.mode=CORE
causal_clustering.initial_discovery_members=core01.example.com:5000,core02.example.com:5000,core03.ex
ample.com:5000
```

*neo4j.conf on core03.example.com:*

```
dbms.default_listen_address=0.0.0.0
dbms.default_advertised_address=core03.example.com
dbms.mode=CORE
causal_clustering.initial_discovery_members=core01.example.com:5000,core02.example.com:5000,core03.ex
```

```
ample.com:5000
```

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 `CALL dbms.cluster.overview()` to check the status of the cluster. This shows information about each member of the cluster:

```
CALL dbms.cluster.overview();
```

```
+----------------------------------------------------------------------------------------------------
-----------------------------------+
| id                                   | addresses                                    | databases
| groups |
+----------------------------------------------------------------------------------------------------
-----------------------------------+
| "8e07406b-90b3-4311-a63f-85c45af63583" | ["bolt://core1:7687", "http://core1:7474"] | {neo4j:
"LEADER", system: "FOLLOWER"}   | []     |
| "aeb6debe-d3ea-4644-bd68-304236f3813b" | ["bolt://core3:7687", "http://core3:7474"] | {neo4j:
"FOLLOWER", system: "FOLLOWER"} | []     |
| "b99ff25e-dc64-4c9c-8a50-ebc1aa0053cf" | ["bolt://core2:7687", "http://core2:7474"] | {neo4j:
"FOLLOWER", system: "LEADER"}   | []     |
+----------------------------------------------------------------------------------------------------
-----------------------------------+
```



**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 follow the messages in *neo4j.log*.

## 9.2.4. Add a Core Server to an existing cluster

Core Servers are added to an existing cluster by starting a new Neo4j instance with the appropriate configuration. The new server will join the existing cluster and become available once it has copied the data from its peers. It may take some time for the new instance to perform the copy if the existing cluster contains large amounts of data.

The setting `causal_clustering.initial_discovery_members` shall be updated on all the servers in the cluster to include the new server.

*Example 51. Add a Core Server to an existing cluster*

In this example, a Core Server, `core04.example.com`, is added to the cluster created in Configure a Core-only cluster.

Configure the following entries in *neo4j.conf*:

*neo4j.conf on core04.example.com:*

```
dbms.default_listen_address=0.0.0.0
dbms.default_advertised_address=core04.example.com
```

```
dbms.mode=CORE
causal_clustering.minimum_core_cluster_size_at_formation=3
causal_clustering.minimum_core_cluster_size_at_runtime=3
causal_clustering.initial_discovery_members=core01.example.com:5000,core02.example.com:5000,core03.ex
ample.com:5000,core04.example.com:5000
```

Note that the configuration is very similar to that of the previous servers. In this example, the new server is not intended to be a permanent member of the cluster, thus it is not included in `causal_clustering.initial_discovery_members` on the other Core members of the cluster.

Now start the new Core Server and let it add itself to the existing cluster.

## 9.2.5. Add a Secondary server to an existing cluster

In the 4.4 version of Neo4j, all Secondary servers are Read Replica instances. The initial configuration for Read Replica instances is provided via *neo4j.conf*, as mentioned above in Configure a cluster with Single and Read Replica instances. Since Read Replicas do not participate in cluster quorum decisions, their configuration is shorter; they only need to know the addresses of at least one primary instance which they can bind to in order to discover the cluster.



**Note:**

It is recommended to specify the addresses for *all* existing primary instances in a cluster when adding a Read Replica. They can then select an appropriate Primary server from which to copy data.

*Example 52. Add a Secondary server to an existing cluster with a Single instance as Primary server*

In this example, a Read Replica instance, `replica04.example.com`, is added to the cluster created in Configure a cluster with a Single instance as Primary server.

Configure the following entries in *neo4j.conf*:

*neo4j.conf on replica01.example.com:*

```
dbms.default_advertised_address=read_replica04.example.com
dbms.mode=READ_REPLICA
causal_clustering.initial_discovery_members=single.example.com:5000
```

Now start the new Read Replica and let it add itself to the existing cluster.

*Example 53. Add a Secondary server to an existing cluster with Core servers as Primary servers*

In this example, a Read Replica, `replica05.example.com`, is added to the cluster created in Configure a Core-only cluster.

Configure the following entries in *neo4j.conf*:

*neo4j.conf on replica05.example.com:*

```
dbms.default_advertised_address=read_replica05.example.com
dbms.mode=READ_REPLICA
causal_clustering.initial_discovery_members=core01.example.com:5000,core02.example.com:5000,core03.example.com:5000
```

Now start the new Read Replica and let it add itself to the existing cluster.



**Note:**

When adding a Secondary server to an existing cluster, only *Primary* servers need to be listed in `causal_clustering.initial_discovery_members`. It is not necessary to include existing Secondary servers, i.e. other Read Replica instances.

## 9.2.6. Detach a Secondary server from an existing cluster

It is possible to turn a Secondary server into a standalone instance that thus contains a snapshot of the data in the cluster. This can, in theory, be done for a Core Server as well, but this is **not** recommended for performance and safety reasons. As mentioned above, in the 4.4 version of Neo4j, all Secondary servers are Read Replica instances.

*Example 54. Detach a Read Replica and turn it into a stand alone instance*

In this example, a Read Replica, `replica01.example.com`, is detached from a cluster. See Add a Secondary server to an existing cluster above on how to add a Read Replica to a cluster.

First, check if the Read Replica is as up-to-date as desired. Use `SHOW DATABASE` to see where the different members of the cluster are in terms of committed transactions compared to the leader.

```
neo4j@system> SHOW DATABASE test00 YIELD name,serverID,address,role,lastCommittedTxn,replicationLag;
```

Note that `SHOW DATABASES` uses `serverID` as it lists databases and there may be more than one database per server, while `dbms.cluster.overview()` uses only `id` as it is only concerned with servers.

```
+-------------------------------------------------------------------------------------------------------------------+
| name     | serverID                               | address          | role           | lastCommittedTxn | replicationLag |
+-------------------------------------------------------------------------------------------------------------------+
| "test00" | "aeb6debe-d3ea-4644-bd68-304236f3813b" | "core3:7687"     | "leader"       | 21423            | 0              |
| "test00" | "8e07406b-90b3-4311-a63f-85c45af63583" | "core1:7687"     | "follower"     | 21422            | -1             |
| "test00" | "b99ff25e-dc64-4c9c-8a50-ebc1aa0053cf" | "core2:7687"     | "follower"     | 21423            | 0              |
| "test00" | "0bf3f6c1-0f48-47c2-a943-18fa8362c918" | "replica4:7687"  | "read_replica" | 21409            | -14            |
| "test00" | "0e9c1b28-c8c0-4c65-a1f2-39d326411280" | "replica6:7687"  | "read_replica" | 21421            | -2             |
| "test00" | "82524236-3058-48a2-b198-6580003475af" | "replica5:7687"  | "read_replica" | 21413            | -10            |
+-------------------------------------------------------------------------------------------------------------------+
```

```
----------------------+
```

Based on the results, decide which Read Replica to detach and proceed to shut it down.

Once the Read Replica is shut down, configure the following entry in *neo4j.conf*:

*neo4j.conf on replica01.example.com:*

```
dbms.mode=SINGLE
```

Start the instance again. It is now a standalone instance containing the data committed to it at the time of shutdown.



**Note:**

There is always a chance that the Read Replica is behind the Core Servers at any time (see above on how to check the state of your cluster members). If a transaction is being processed at the time of the shutdown of the Read Replica, this transaction is eventually reflected in the remaining Cluster, but not on the detached Read Replica. A way to ensure that a Read Replica contains a snapshot of a database in the cluster at a point in time, is to pause the read Replica before shutting it down. See `dbms.cluster.readReplicaToggle()` for more information.

# 9.3. Seed a cluster

## 9.3.1. Introduction

Regardless of whether you are just playing around with Neo4j or setting up a production environment, you likely have some existing data that you want to transfer into your newly created Causal cluster. Neo4j supports seeding a cluster from a database dump, a database backup, or from another data source (with the Import tool). For more information about the different backup options and how to use the Neo4j Import tool, see Backup and restore options and Neo4j Admin tool.

It is possible to seed a cluster with a single database or multiple, including a full DBMS. Any seeding that includes restoring the `system` database needs to be done offline, but any other databases can be seeded online.



**Note:**

The databases that you want to seed and the Neo4j cluster must be of the same version.

The process for seeding a cluster is essentially the same for clusters with Single and Read Replica instances as for clusters with Core (and optional Read Replica) instances. However, using a designated seeder is only applicable to clusters with Core instances. The seeding is usually performed on primary instances only but it is possible to seed a Read Replica instance, yet it is not necessary unless for performance reasons.

## 9.3.2. Seed a cluster from a database dump (offline)

This could be an offline backup (i.e. a dump) from a standalone Neo4j instance or a cluster member (e.g., an existing Read Replica instance). The following example seeds a newly created cluster with an example DBMS consisting of the `system` database and the default database, `neo4j` from a dump. If you want to seed a single user database, follow the steps in Seed a cluster from a database backup (online) further on.

> **Tip:**
>
> This scenario is useful in disaster recovery where some servers have retained their data during a catastrophic event.

> **Caution:**
>
> Moving files and directories manually in or out of a Neo4j installation is not recommended and considered unsupported.

1. Create a new Neo4j Core-only cluster following the instructions in Configure a cluster with Core instances but **do not start** any of the members. (If you have started any of the cluster members, stop and unbind each started member.)
2. Use `neo4j-admin load` to seed each of the Core members in the cluster.

   The examples assume that you are restoring one user database with the default name of `neo4j` and the `system` database, containing the replicated configuration state. Modify the command line arguments to match your exact setup.

   ```
   neo4j-01$ ./bin/neo4j-admin load --from=/path/to/system.dump --database=system
   neo4j-01$ ./bin/neo4j-admin load --from=/path/to/neo4j.dump --database=neo4j
   neo4j-02$ ./bin/neo4j-admin load --from=/path/to/system.dump --database=system
   neo4j-02$ ./bin/neo4j-admin load --from=/path/to/neo4j.dump --database=neo4j
   neo4j-03$ ./bin/neo4j-admin load --from=/path/to/system.dump --database=system
   neo4j-03$ ./bin/neo4j-admin load --from=/path/to/neo4j.dump --database=neo4j
   ```

3. Start each cluster member.

   ```
   neo4j-01$ ./bin/neo4j start
   neo4j-02$ ./bin/neo4j start
   neo4j-03$ ./bin/neo4j start
   ```

   The cluster forms and the replicated Neo4j DBMS deployment comes online.

> **Note:**
>
> The system database contains information about the databases that should exist in your Neo4j DBMS. If a database does not exist in your system database (because it has not been created previously), it must be created with `CREATE DATABASE <database-name>` even if it has been restored/loaded/imported.

## 9.3.3. Seed a cluster from a database backup (online)

> **Tip:**
>
> These scenarios are useful when you want to restore a database in a running cluster.

If you have a running Neo4j database that you want to seed in a running cluster, use `neo4j-admin backup` to create a database backup. This could be a backup from a standalone Neo4j instance or another cluster member (e.g., an existing Read Replica).

Neo4j supports two types of seeding in a running cluster. You can either transfer the database backup to each Core instance or transfer it only to one Core instance and then use the `CREATE DATABASE` Cypher command to seed the cluster. For more information on the `CREATE DATABASE` syntax and options, see Cypher Manual → Creating databases.

> **Caution:**
>
> Moving files and directories manually in or out of a Neo4j installation is not recommended and considered unsupported.

### Restore a database on each Core instance

Transfer the database backup to each Core instance in the cluster using the `neo4j-admin restore` command and then use `CREATE DATABASE` to restore it. This example uses a user database called `movies1`.

1. To ensure that the `movies1` database does not exist in the cluster, on one of the Core members, use Cypher Shell and run `DROP DATABASE movies1`. Use the `system` database to connect. The command is automatically routed to the appropriate Core instance and from there to the other cluster members.

   ```
   DROP DATABASE movies1;
   ```

   > **Note:**
   >
   > Dropping a database also deletes the users and roles associated with it.

   > **Important:**
   >
   > If you cannot drop the database because your seeds include the `system` database (which cannot be dropped), you must run `neo4j-admin unbind` . However, this removes the cluster state of the Core instance and in turn the instance needs to be restarted in order to join the cluster. Thus, you are no longer restoring a database in a running cluster. See Seed a cluster from a database dump (offline) instead for instructions on how to seed an offline cluster.

2. Restore the database on each Core member in the cluster.

```
neo4j@core1$ ./bin/neo4j-admin restore --from=/path/to/movies1-backup-dir --database=movies1
neo4j@core2$ ./bin/neo4j-admin restore --from=/path/to/movies1-backup-dir --database=movies1
neo4j@core3$ ./bin/neo4j-admin restore --from=/path/to/movies1-backup-dir --database=movies1
```

However, restoring a database does not automatically create it.

3. On one of the Core instances, run `CREATE DATABASE movies1` against the `system` database to create the `movies1` database. The command is automatically routed to the appropriate Core instance and from there to the other cluster members.

```
CREATE DATABASE movies1;
```

```
0 rows
ready to start consuming query after 701 ms, results consumed after another 0 ms
```

4. Verify that the `movies1` database is online on all members.

```
SHOW DATABASES;
```

```
+-----------------------------------------------------------------------------------------------------------
----------------------+
| name     | aliases | access       | address      | role       | requestedStatus | currentStatus |
error | default | home  |
+-----------------------------------------------------------------------------------------------------------
----------------------+
| "neo4j"  | []      | "read-write" | "core1:7687" | "leader"   | "online"        | "online"      | ""
| TRUE    | TRUE  |
| "neo4j"  | []      | "read-write" | "core3:7687" | "follower" | "online"        | "online"      | ""
| TRUE    | TRUE  |
| "neo4j"  | []      | "read-write" | "core2:7687" | "follower" | "online"        | "online"      | ""
| TRUE    | TRUE  |
| "movies1"| []      | "read-write" | "core1:7687" | "leader"   | "online"        | "online"      | ""
| FALSE   | FALSE |
| "movies1"| []      | "read-write" | "core3:7687" | "follower" | "online"        | "online"      | ""
| FALSE   | FALSE |
| "movies1"| []      | "read-write" | "core2:7687" | "follower" | "online"        | "online"      | ""
| FALSE   | FALSE |
| "system" | []      | "read-write" | "core1:7687" | "follower" | "online"        | "online"      | ""
| FALSE   | FALSE |
| "system" | []      | "read-write" | "core3:7687" | "follower" | "online"        | "online"      | ""
| FALSE   | FALSE |
| "system" | []      | "read-write" | "core2:7687" | "leader"   | "online"        | "online"      | ""
| FALSE   | FALSE |
+-----------------------------------------------------------------------------------------------------------
----------------------+

9 rows available after 3 ms, consumed after another 1 ms
```

## Restore a database using a designated seeder

With a seeder, you transfer the database backup to one Core instance in the cluster using the `neo4j-admin restore` command. Then you use that member as a designated seeder to create the backed-up database on the other cluster members.

This example uses a user database called `movies1` and a cluster that consists of three Core instances. The `movies1` database **does not** exist on any of the cluster members.

If a database with the same name as your backup already exists in your cluster, see step 1 in Restore a

database on each Core instance for details on how to drop it.

1. Restore the `movies1` database on one of the Core instances. In this example, the `core1` member is used.

```
neo4j@core1$ ./bin/neo4j-admin restore --from=/path/to/movies1-backup-dir --database=movies1
```

2. Find the server ID of `core1` by logging in to Cypher Shell and running `dbms.cluster.overview()`. Use any database to connect.

```
CALL dbms.cluster.overview();
```

```
+-----------------------------------------------------------------------------------------------------
----------------------------------+
| id                                   | addresses                                    | databases
| groups |
+-----------------------------------------------------------------------------------------------------
----------------------------------+
| "8e07406b-90b3-4311-a63f-85c45af63583" | ["bolt://core1:7687", "http://core1:7474"] | {neo4j:
"LEADER", system: "FOLLOWER"}   | []     |
| "aeb6debe-d3ea-4644-bd68-304236f3813b" | ["bolt://core3:7687", "http://core3:7474"] | {neo4j:
"FOLLOWER", system: "FOLLOWER"} | []     |
| "b99ff25e-dc64-4c9c-8a50-ebc1aa0053cf" | ["bolt://core2:7687", "http://core2:7474"] | {neo4j:
"FOLLOWER", system: "LEADER"}   | []     |
+-----------------------------------------------------------------------------------------------------
----------------------------------+
```

3. On one of the Core instances, use the `system` database and create the database `movies1` using the server ID of `core1`. The command is automatically routed to the appropriate Core instance and from there to the other cluster members. If the `movies1` database is of considerable size, the execution of the command can take some time.

```
CREATE DATABASE movies1 OPTIONS {existingData: 'use', existingDataSeedInstance: '8e07406b-90b3-4311-
a63f-85c45af63583'};
```

```
0 rows
ready to start consuming query after 701 ms, results consumed after another 0 ms
```

4. Verify that the `movies1` database is online on all cluster members.

```
SHOW DATABASES;
```

```
+-----------------------------------------------------------------------------------------------------
----------------------+
| name     | aliases | access       | address      | role       | requestedStatus | currentStatus |
error | default | home  |
+-----------------------------------------------------------------------------------------------------
----------------------+
| "neo4j"  | []      | "read-write" | "core1:7687" | "leader"   | "online"        | "online"      | ""
| TRUE    | TRUE  |
| "neo4j"  | []      | "read-write" | "core3:7687" | "follower" | "online"        | "online"      | ""
| TRUE    | TRUE  |
| "neo4j"  | []      | "read-write" | "core2:7687" | "follower" | "online"        | "online"      | ""
| TRUE    | TRUE  |
| "movies1"| []      | "read-write" | "core1:7687" | "leader"   | "online"        | "online"      | ""
| FALSE   | FALSE |
| "movies1"| []      | "read-write" | "core3:7687" | "follower" | "online"        | "online"      | ""
| FALSE   | FALSE |
| "movies1"| []      | "read-write" | "core2:7687" | "follower" | "online"        | "online"      | ""
```

```
| FALSE   | FALSE |
| "system" | []      | "read-write" | "core1:7687" | "follower" | "online"        | "online"      | ""
| FALSE   | FALSE |
| "system" | []      | "read-write" | "core3:7687" | "follower" | "online"        | "online"      | ""
| FALSE   | FALSE |
| "system" | []      | "read-write" | "core2:7687" | "leader"   | "online"        | "online"      | ""
| FALSE   | FALSE |
+-------------------------------------------------------------------------------------------------------------
---------------------+

9 rows available after 3 ms, consumed after another 1 ms
```

### 9.3.4. Seed a cluster using the import tool

To create a cluster based on imported data, it is recommended to first import the data into a standalone Neo4j DBMS and then use an offline backup to seed the cluster.

1. Import the data.
   a. Deploy a standalone Neo4j DBMS.
   b. Import the data using the import tool.
2. Use `neo4j-admin dump` to create an offline backup of the `neo4j` database.
3. Seed a new cluster using the instructions in Seed a cluster from a database dump (offline).

   Skip the `system` database in this scenario since it is not needed.

## 9.4. Discovery

### 9.4.1. Overview

In order to form or connect to a running cluster, a Core Server or a Read Replica needs to know the addresses of some of the Core Servers. This information is used to bind to the Core Servers in order to run the discovery protocol and get the full information about the cluster. The way in which this is best done depends on the configuration in each specific case.

> **Note:**
>
> A Single instance used as a Primary server does not need to be configured for discovery. However, `discovery_advertised_address` and `discovery_listen_address` can be configured if other addresses than the default are desired.

If the addresses of the other cluster members are known upfront, they can be listed explicitly. This is convenient, but has limitations:

- If Core instances are replaced and the new members have different addresses, the list will become 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 Causal Cluster.

Additional mechanisms for using DNS are provided for the cases where it is not practical or possible to explicitly list the addresses of cluster members to discover.

The discovery configuration is just used for initial discovery and a running cluster will continuously exchange information about changes to the topology. The behavior of the initial discovery is determined by the parameters `causal_clustering.discovery_type` and `causal_clustering.initial_discovery_members`, and is described in the following sections.

## Discovery using a list of server addresses

If the addresses of the other cluster members are known upfront, they can be listed explicitly. We use the default `causal_clustering.discovery_type=LIST` and hard code the addresses in the configuration of each machine. This alternative is illustrated by Configure a Core-only cluster.

## Discovery using DNS with multiple records

When using initial discovery with DNS, a DNS record lookup is performed when an instance starts up. Once an instance has joined a cluster, further membership changes are communicated amongst Core instances as part of the discovery service.

The following DNS-based mechanisms can be used to get the addresses of Core instances for discovery:

`causal_clustering.discovery_type=DNS`

With this configuration, the initial discovery members will be resolved from *DNS A* records to find the IP addresses to contact. The value of `causal_clustering.initial_discovery_members` should be set to a single domain name and the port of the discovery service. For example: `causal_clustering.initial_discovery_members=cluster01.example.com:5000`. The domain name should return an A record for every Core instance when a DNS lookup is performed. Each A record returned by DNS should contain the IP address of the Core instance. The configured Primary server will use all the IP addresses from the A records to join or form a cluster.

The discovery port must be the same on all Core instances when using this configuration. If this is not possible, consider using the discovery type `SRV` instead.

`causal_clustering.discovery_type=SRV`

With this configuration, the initial discovery members will be resolved from *DNS SRV* records to find the IP addresses/hostnames and discovery service ports to contact. The value of `causal_clustering.initial_discovery_members` should be set to a single domain name and the port set to `0`. For example: `causal_clustering.initial_discovery_members=cluster01.example.com:0`. The domain name should return a single SRV record when a DNS lookup is performed. The SRV record returned by DNS should contain the IP address or hostname, and the discovery port, for the Core Servers to be discovered. The configured Primary server will use 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 Primary server is running as a Kubernetes service. Then, the addresses of the Core instances 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 `causal_clustering.discovery_type=K8S`.
- Set `causal_clustering.kubernetes.label_selector` to the label selector for the Causal Cluster services. For more information, see the Kubernetes official documentation.
- Set `causal_clustering.kubernetes.service_port_name` 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, `causal_clustering.initial_discovery_members` is not used and any value assigned to it will be ignored.



Note:

- The pod running Neo4j must use a service account which has permission to list services. For further information, see the Kubernetes documentation on RBAC authorization or ABAC authorization.
- The configured `causal_clustering.discovery_advertised_address` must exactly match the Kubernetes-internal DNS name, which will be of the form `<service-name>.<namespace>.svc.cluster.local`.

As with DNS-based methods, the Kubernetes record lookup is only performed at startup.

## 9.5. Intra-cluster encryption



Caution:

Securing client to server communication is not covered in this chapter (e.g. Bolt, HTTPS, Backup).

### 9.5.1. Introduction

The security solution for cluster communication is based on standard SSL/TLS technology (referred to jointly as SSL). Encryption is in fact just one aspect of security, with the other cornerstones being authentication and integrity. A secure solution will be 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 will cover 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).

It should be noted that 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.



**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
```

## 9.5.2. Example deployment

The following steps will create an example deployment, and each step is expanded in further detail below.

- Generate and install cryptographic objects
- Configure Causal Clustering with the SSL policy
- Validate the secure operation of the cluster

### Generate and install cryptographic objects

The generation of cryptographic objects is for the most part outside the scope of this manual. It will generally require having a PKI with a Certificate Authority (CA) within the organization and they should be able to advise here. Please note that the information in this manual relating to the PKI is mainly for illustrative purposes.

When the certificates and private keys have been obtained they can be installed on each of the servers. Each server will have 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 will thus be trusted. This means that the server now has the capability of establishing trust with other servers.



**Caution:**

Please be sure to exercise caution when using CA certificates in the `trusted` directory, as any certificates signed by that CA will then be trusted to join the cluster. For this reason, never use a public 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.

In this example we will deploy a mutual authentication setup, 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 that you 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 55. Generate and install cryptographic objects*

In this example we assume that the private key and certificate file are named *private.key* and *public.crt*, respectively. If you want to use different names you may override the policy configuration for the key and certificate names/locations. We want to use the default configuration for this server so we 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 Causal 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. We will use the following parameters in our configuration:

*Table 435. Example settings*

| Setting suffix | Value | Comment |
| --- | --- | --- |
| `client_auth` | `REQUIRE` | Setting this to `REQUIRE` effectively enables mutual authentication for servers. |

| Setting suffix | Value | Comment |
| --- | --- | --- |
| `ciphers` | `TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384` | We can enforce a particular single strong cipher and remove any doubt about which cipher gets negotiated and chosen. The cipher chosen above offers Perfect Forward Secrecy (PFS) which is generally desirable. It also uses Advanced Encryption Standard (AES) for symmetric encryption which has great support for acceleration in hardware and thus allows performance to generally be negligibly affected. |
| `tls_versions` | `TLSv1.2` | Since we control the entire cluster we can enforce the latest TLS standard without any concern for backwards compatibility. It has no known security vulnerabilities and uses the most modern algorithms for key exchanges, etc. |

In the following example we will create and configure an SSL policy that we will use in our cluster.

*Example 56. Configure the cluster SSL policy*

In this example we assume that the directory structure has been created, and certificate files have been installed, as per the previous example.

We add the following content to our *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
```

Any user data communicated between instances will now be secured. Please note that an instance which is not correctly setup would not be able to communicate with the others.

Note that 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.

## 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 57. Validate the secure operation of the cluster*

In this example we will use the `nmap` tool to validate the secure operation of our 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 our configuration. This can prove that TLS is in fact enabled and that the only the intended cipher suites are enabled. All servers and all applicable ports should be tested.

For testing purposes we could also attempt to utilize a separate testing instance of Neo4j which, for example, has an untrusted certificate in place. The expected result of this test is that the test server is not able to participate in replication of user data. The debug logs will generally indicate an issue by printing an SSL or certificate-related exception.

# 9.6. Internals of clustering

## 9.6.1. Elections and leadership

The Core instances used as Primary servers in a cluster use the Raft protocol to ensure consistency and safety. See Advanced Causal Clustering for more information on the Raft protocol. 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 Core instance currently playing the *Leader* role for the respective database. 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 Core instance 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 Cores to vote for it. If it can get a majority of the votes, then it assumes the leader role. Cores will 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).

## 9.6.2. Leadership balancing

Write transactions will always be 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 instances. By default, Neo4j avoids this by automatically transferring database leaderships so that they are evenly

distributed throughout the cluster. Additionally, Neo4j will automatically transfer database leaderships away from instances where those databases are configured to be read-only using dbms.databases.read_only or similar.

### 9.6.3. Multi-database and the reconciler

Databases operate as independent entities in a Neo4j DBMS, both in standalone and in a cluster. Since a cluster can consist of multiple independent server instances, the effects of administrative operations like creating a new database happen asynchronously and independently for each server. However, the immediate effect of an administrative operation is to safely commit the desired state in the `system` database.

The desired state committed in the `system` database gets replicated and is picked up by an internal component called the reconciler. It runs on every instance and takes the appropriate actions required locally on that instance for reaching the desired state; creating, starting, stopping, and dropping databases.

Every database runs in an independent Raft group and since there are two databases in a fresh cluster, `system` and `neo4j`, this means that it also has two Raft groups. Every Raft group also has an independent leader and thus a particular Core instance could be the leader for one database and a follower for another.



**Note:**

This does not apply to clusters where a Single instance is the Primary server. In such clusters, the Single instance is the leader of all databases and there is no Raft at all.

### 9.6.4. Server-side routing

Server-side routing is a complement to the client-side routing.

In a Causal Cluster deployment of Neo4j, Cypher queries may be directed to a cluster member that is unable to run the given query. With server-side routing enabled, such queries will be rerouted internally to a cluster member that is expected to be able to run it. 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 core 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 the Cypher `USE` clause), server-side routing will be 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 (`dbms.routing.listen_address`) and advertised address (`dbms.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 will not perform any routing when the `bolt://` URI scheme is used, instead connecting directly to the specified host.

On the cluster-side you must fulfil the following pre-requisites to make server-side routing available:

- Set `dbms.routing.enabled=true` on each member of the cluster.
- Configure `dbms.routing.listen_address`, and provide the advertised address using `dbms.routing.advertised_address` on each member.
- Optionally, you can set `dbms.routing.default_router=SERVER` on each member of the cluster.

The final pre-requisite 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 will benefit from these settings automatically.

The table below shows the criteria by which server-side routing is performed:

*Table 436. Server-side routing criteria*

| CLIENT - Neo4j Driver (Bolt Protocol) | | | | SERVER - Neo4j Cluster member | | |
|---|---|---|---|---|---|---|
| URI scheme | Client-side routing | Request server-side routing | Transaction type | Server - Instance > Role (per database) | Server-side routing enabled | Routes the query |
| `neo4j://` | ✓ | ✓ | write | Primary - Single | ✓ | ✗ |
| `neo4j://` | ✓ | ✓ | read | Primary - Single | ✓ | ✗ |
| `neo4j://` | ✓ | ✓ | write | Primary - Core > leader | ✓ | ✗ |
| `neo4j://` | ✓ | ✓ | read | Primary - Core > leader | ✓ | ✗ |
| `neo4j://` | ✓ | ✓ | write | Primary - Core > follower | ✓ | ✓ |
| `neo4j://` | ✓ | ✓ | read | Primary - Core > follower | ✓ | ✗ |
| `neo4j://` | ✓ | ✓ | write | Secondary - Read Replica | ✓ | ✓ |
| `neo4j://` | ✓ | ✓ | read | Secondary - Read Replica | ✓ | ✗ |
| `bolt://` | ✗ | ✗ | write | Primary - Single | ✓ | ✗ |
| `bolt://` | ✗ | ✗ | read | Primary - Single | ✓ | ✗ |

| CLIENT - Neo4j Driver (Bolt Protocol) | | | | SERVER - Neo4j Cluster member | | |
|---|---|---|---|---|---|---|
| `bolt://` | ✗ | ✗ | write | Primary - Core > leader | ✓ | ✗ |
| `bolt://` | ✗ | ✗ | read | Primary - Core > leader | ✓ | ✗ |
| `bolt://` | ✗ | ✗ | write | Primary - Core > follower | ✓ | ✗ |
| `bolt://` | ✗ | ✗ | read | Primary - Core > follower | ✓ | ✗ |
| `bolt://` | ✗ | ✗ | write | Secondary - Read Replica | ✓ | ✗ |
| `bolt://` | ✗ | ✗ | read | Secondary - Read Replica | ✓ | ✗ |

*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`
- `dbms.routing.listen_address`
- `dbms.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.*`

*Server-side routing encryption*

Encryption of server-side routing communication is configured by the cluster SSL policy. For more information, see Cluster Encryption.

## 9.6.5. Store copy

Store copies are initiated when an instance does not have an up-to-date copy of the database. For example, this is the case when a new instance is joining a cluster (without a seed). It can also happen as a consequence of falling behind the rest of the cluster, for reasons such as connectivity issues or having been shut down. Upon re-establishing connection with the cluster, an instance recognizes that it is too far behind and fetches a new copy from the rest of the cluster.

A store copy is a major operation, which may disrupt the availability of instances in the cluster. Store copies should not be a frequent occurrence in a well-functioning cluster, but rather be an exceptional operation that happens due to specific causes, e.g. network outages or planned maintenance outages. If store copies happen during regular operation, then the configuration of the cluster, or the workload directed at it, might have to be reviewed so that all instances can keep up, and that there is enough of a buffer of Raft logs and transaction logs to handle smaller transient issues.

The protocol used for store copies is robust and configurable. The network requests are directed at an upstream member according to configuration and they are retried despite transient failures. The maximum amount of time to retry every request can be configured with

`causal_clustering.store_copy_max_retry_time_per_request`. If a request fails and the maximum retry time has elapsed then it stops retrying and the store copy fails.

Use `causal_clustering.catch_up_client_inactivity_timeout` to configure the inactivity timeout for any particular request.



Note:

The `causal_clustering.catch_up_client_inactivity_timeout` configuration is for all requests from the catchup client, including the pulling of transactions.

The default upstream strategy is not applicable to Single instances and it differs for Core and Read Replica instances. Core instances always send the initial request to the leader to get the most up-to-date information about the store. The strategy for the file and index requests for Core instances is to vary every other request to a random Read Replica instance and every other to a random Core instance.

Read Replica instances use the same strategy for store copies as it uses for pulling transactions. The default is to pull from a random Core instance.

If you are running a multi-datacenter cluster, then upstream strategies for both Core and Read Replica instances can be configured. Remember that for Read Replica instances, this also affects from where transactions are pulled. See more in Configure for multi-data center operations.

## Using the Replica instance in case of failure

In case of failure (e.g. a partial failure of a cluster due to the loss of an instance, but not of the majority), you may transform a Read Replica instance into a Core instance as a way to restore the cluster's core availability. However, keep in mind that this is not advised as it could cause data loss and complications in the Raft group.

To avoid that, the *read_replica* instance must not be initialized as a **single** instance, nor be introduced in a different or new cluster. This action would cause an override of the raft state, thus preventing the replica from successfully joining the targeted cluster.

After performing that change, follow these instructions to unbind the Replica instance and update the discovery configurations amongst cluster members:

1. Ensure that the converted *read_replica* currently belongs to the same cluster that it will be re-introduced back to, as a *core*. This can be done by performing `CALL dbms.cluster.overview()` and verifying the instance's address and cluster mode.
2. Stop and unbind the *read_replica* instance.
3. Update the cluster mode configuration in *neo4j.conf*, from `dbms.mode=READ_REPLICA` to `dbms.mode=CORE`.
4. Stop Neo4j on the removed core instances that are not intended to serve as core members.
5. Unbind those instances from the cluster by performing `neo4j-admin unbind` while they are stopped. This action will prevent such instances from subsequently attempting to rejoin the running cluster.

At this point, the previous *read_replica* (now *core*) instance may be introduced into the running cluster. To

persist this change in the cluster's architecture, the following configuration updates are advised:

- On the previous *read_replica* (now *core*) instance, set `causal_clustering.discovery_advertised_address` and `causal_clustering.discovery_listen_address` as appropriate.
- Update the `causal_clustering.initial_discovery_members` configuration with the currently valid list of discovery addresses for each member of the cluster. This should replace the addresses of any removed *core(s)* with the discovery addresses of the previous *read_replica* (now *core*) instance.

> **Note:**
>
> In cases where `causal_clustering.discovery_type` is other than `LIST`, make sure to update the corresponding address resolution addresses records. For example, DNS A records for discovery types DNS and SRV, and any Kubernetes service address alternate to reflect the inclusion of the *read_replica* discovery address.

### 9.6.6. On-disk state

The on-disk state of cluster instances is different from that of standalone instances. The biggest difference is the existence of an additional cluster state. Most of the files there are relatively small, but the Raft logs can become quite large depending on the configuration and workload.

It is important to understand that once a database has been extracted from a cluster and used in a standalone deployment, it must not be put back into an operational cluster.

> **Warning:**
>
> If you try to reinsert a modified database back into the cluster, then the logs and stores will mismatch. Operators should not try to merge standalone databases into the cluster in the optimistic hope that their data will become replicated. That does not happen and instead, it likely leads to unpredictable cluster behavior.

## 9.7. Settings reference

### 9.7.1. Common server settings

> **Note:**
>
> In the current version of Neo4j, the clustering-related parameters use the `causal_clustering` namespace. This will be replaced with a more suitable namespace in an upcoming release.

| Parameter | Instance type | Explanation |
| --- | --- | --- |
| `dbms.clustering.enable` | Single | This setting allows a `SINGLE` instance to form a cluster with one or more `READ_REPLICA` instances. Must be set to `TRUE` on the `SINGLE` instance only, as the setting is ignored on `CORE` and `READ_REPLICA` instances. |
| `dbms.mode` | All | This setting configures the operating mode of the database. In version 4.4, there are three possible modes:<br>1. `SINGLE` and `READ_REPLICA` instances<br>2. `CORE` only instances<br>3. `CORE` and `READ_REPLICA` instances<br>**Example:** `dbms.mode=READ_REPLICA` defines a Read Replica instance |
| `dbms.read_only` | - | This setting is not supported. |
| `causal_clustering.minimum_core_cluster_size_at_formation` | Core | Minimum number of Core instances as Primary servers required to form a Core cluster.<br>**Example:** `causal_clustering.minimum_core_cluster_size_at_formation=3` specifies that the cluster will form when at least three Core instances have discovered each other. |
| `causal_clustering.minimum_core_cluster_size_at_runtime` | Core | The minimum size of the dynamically adjusted voting set (which only Core members may be a part of).<br>Adjustments to the voting set happen automatically as the availability of Core instances changes, due to explicit operations such as starting or stopping a member, or unintended issues such as network partitions. Please note that this dynamic scaling of the voting set is generally desirable, as under some circumstances it can increase the number of instance failures which may be tolerated.<br>A majority of the voting set must be available before members are voted in or out.<br>**Example:** `causal_clustering.minimum_core_cluster_size_at_runtime=3` specifies that the cluster should not try to dynamically adjust below three Core instances in the voting set. |

| Parameter | Instance type | Explanation |
| --- | --- | --- |
| `causal_clustering.discovery_type` | Core and Read Replica | 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`, `DNS`, `SRV`, and `K8S`.<br><br>`LIST`<br>Treat `causal_clustering.initial_discovery_members` as a list of addresses of Core instances to contact for discovery.<br><br>`DNS`<br>Treat `causal_clustering.initial_discovery_members` as a domain name to resolve via DNS. Expect DNS resolution to provide A records with hostnames or IP addresses of Core instances to contact for discovery, on the port specified by `causal_clustering.initial_discovery_members`.<br><br>`SRV`<br>Treat `causal_clustering.initial_discovery_members` as a domain name to resolve via DNS. Expect DNS resolution to provide SRV records with hostnames or IP addresses and ports, of Core instances to contact for discovery.<br><br>`K8S`<br>Access the Kubernetes list service API to derive addresses of Core instances to contact for discovery. Requires `causal_clustering.kubernetes.label_selector` to be a Kubernetes label selector for Kubernetes services running a Core each and `causal_clustering.kubernetes.service_port_name` to be a service port name identifying the discovery port of Core services. The value of `causal_clustering.initial_discovery_members` is ignored for this option.<br><br>The value of this setting determines how `causal_clustering.initial_discovery_members` is interpreted. Detailed information about discovery and discovery configuration options is given in Discovery using DNS with multiple records.<br><br>**Example:** `causal_clustering.discovery_type=DNS` combined with `causal_clustering.initial_discovery_members=cluster01.example.com:5000` fetch all DNS A records for *cluster01.example.com* and attempt to reach Neo4j instances listening on port `5000` for each A record's IP address. |

| Parameter | Instance type | Explanation |
| --- | --- | --- |
| `causal_clustering.initial_discovery_members` | Core and Read Replica | The network addresses of an initial set of Core instance members that are available to bootstrap this Core or Read Replica instance. 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`.<br><br>It is good practice to set this parameter to the same value on all Core instances.<br><br>The behavior of this setting can be modified by configuring the setting `causal_clustering.discovery_type`. This is described in detail in Discovery using DNS with multiple records.<br><br>Example: `causal_clustering.discovery_type=LIST` combined with `core01.example.com:5000,core02.example.com:5000,core03.example.com:5000` will attempt to reach Neo4j instances listening on *core01.example.com*, core01.example.com and core01.example.com; all on port `5000`. |
| `causal_clustering.discovery_advertised_address` | All | 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 instance is included in the `initial_discovery_members` of other cluster members, the value there must **exactly** match this advertised address.<br><br>Example: `causal_clustering.discovery_advertised_address=192.168.33.21:5001` indicates that other cluster members can communicate with this instance using the discovery protocol at host `192.168.33.20` and port `5001`. |
| `causal_clustering.raft_advertised_address` | Core | The address/port setting that specifies where the Neo4j instance advertises to other members of the cluster that it listens for Raft messages within the Core cluster.<br><br>Example: `causal_clustering.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`. |
| `causal_clustering.transaction_advertised_address` | All | The address/port setting that specifies where the instance advertises where it listens for requests for transactions in the transaction-shipping catchup protocol.<br><br>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`. |

| Parameter | Instance type | Explanation |
| --- | --- | --- |
| `causal_clustering.discovery_listen_address` | All | The address/port setting that specifies which network interface and port the Neo4j instance binds to for the cluster discovery protocol.<br><br>Example: `causal_clustering.discovery_listen_address=0.0.0.0:5001` will listen for cluster membership communication on any network interface at port `5001`. |
| `causal_clustering.raft_listen_address` | Core | 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 `causal_clustering.raft_advertised_address`.<br><br>Example: `causal_clustering.raft_listen_address=0.0.0.0:7000` listens for cluster communication on any network interface at port `7000`. |
| `causal_clustering.transaction_listen_address` | All | 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 `causal_clustering.transaction_advertised_address`.<br><br>Example: `causal_clustering.transaction_listen_address=0.0.0.0:6001` listens for cluster communication on any network interface at port `6001`. |
| `causal_clustering.store_copy_max_retry_time_per_request` | Core and Read Replica | Condition for when store copy should eventually fail. A request is allowed to retry for any amount of attempts as long as the configured time has not been met. For very large stores or other reason that might make transferring of files slow this could be increased.<br><br>Example: `causal_clustering.store_copy_max_retry_time_per_request=60min` |

## 9.7.2. Multi-data center settings

| Parameter | Explanation |
| --- | --- |
| `causal_clustering.multi_dc_license` | Enables multi-data center features. Requires appropriate licensing.<br><br>Example: `causal_clustering.multi_dc_license=true` will enable the multi-data center features. |

| Parameter | Explanation |
| --- | --- |
| `causal_clustering.server_groups` | A list of group names for the server used when configuring load balancing and replication policies.<br><br>**Example:** `causal_clustering.server_groups=us,us-east` will add the current instance to the groups `us` and `us-east`. |
| `causal_clustering.leadership_priority_group.<database>` | The group of servers which should be preferred when selecting leaders for the specified database. If the instance currently acting as leader for this database is not a member of the configured server group, then the cluster will attempt to transfer leadership to an instance which *is* a member. It is not guaranteed that leadership will always be held by a server in the desired group. For example, if no member of the desired group is available or has up-to-date store contents. The cluster will seek to preserve availability, over respecting the `leadership_priority_group` setting.<br><br>To set a default `leadership_priority_group` for all databases that do not have an explicitly set `leadership_priority_group`, the `<database>` can be omitted. See `causal_clustering.leadership_priority_group`.<br><br>**Example:** `causal_cluster.leadership_priority_group.foo=us` will ensure that if the leader for `foo` is not held by a server configured with `causal_clustering.server_groups=us`, the cluster will attempt to transfer leadership to a server which is. |
| `causal_clustering.upstream_selection_strategy` | An ordered list in descending preference of the strategy which Read Replicas use to choose upstream database server from which to pull transactional updates.<br><br>**Example:** `causal_clustering.upstream_selection_strategy=connect-randomly-within-server-group,typically-connect-to-random-read-replica` will configure the behavior so that the Read Replica will first try to connect to any other instance in the group(s) specified in `causal_clustering.server_groups`. Should we fail to find any live instances in those groups, then we will connect to a random Read Replica. A value of `user-defined` will enable custom strategy definitions using the setting `causal_clustering.user_defined_upstream_strategy`. |
| `causal_clustering.user_defined_upstream_strategy` | Defines the configuration of upstream dependencies. Can only be used if `causal_clustering.upstream_selection_strategy` is set to `user-defined`.<br><br>**Example:** `causal_clustering.user_defined_upstream_strategy=groups(north2); groups(north); halt()` will look for servers in the `north2`. If none are available it will look in the `north` server group. Finally, if we cannot resolve any servers in any of the previous groups, then rule chain will be stopped via `halt()`. |
| `causal_clustering.load_balancing.plugin` | The load balancing plugin to use. One pre-defined plugin named `server_policies` is available by default.<br><br>**Example:** `causal_clustering.load_balancing.plugin=server_policies` will enable custom policy definitions. |

| Parameter | Explanation |
| --- | --- |
| `causal_clustering.load_balancing.config.server_policies.<policy-name>` | Defines a custom policy under the name `<policy-name>`. Note that load balancing policies are cluster-global configurations and should be defined the exact same way on all core machines.<br><br>**Example:**<br>`causal_clustering.load_balancing.config.server_policies.north1_only=groups(north1)->min(2); halt();` defines a load balancing policy named `north1_only`.<br>Queries are sent only to servers in the `north1` server group, provided there are two of them available. If there are less than two servers in `north1`, the chain is halted.<br><br>By default, the load balancer sends read requests only to replicas/followers, which means these two servers must be of that kind. To allow reads on the leader, set to `causal_clustering.cluster_allow_reads_on_leader` to `true`. |

# 9.8. Clustering glossary

| Term | Description |
| --- | --- |
| 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. |
| Causal cluster | A collection of servers running Neo4j that are configured to communicate with each other. The servers can be either *Primary* or *Secondary*, where the Primary servers allow read and write operations and the Secondary servers allow only read operations. As long as it uses bookmarks, the cluster guarantees that a client application can read at least its own writes. See also Primary server, Secondary server, and Causal consistency. |
| 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. |
| 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 | Each database has a designated leader within the cluster and it can only be located on a primary server. The leader 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 server | A primary server can be either a single instance or a core instance, both of which allow read and write operations. A single instance as primary is beneficial for read scalability but is not fault tolerant. A Core instance safeguards data and a cluster of at least three Core instances is fault tolerant. It participates in fault tolerant writes as it is part of the majority required to acknowledge and commit write transactions. |
| Read scaling | Adding Secondary servers to the cluster can offload read queries from the Primary servers and thus reduce the load and aid write performance of the cluster. |

| Term | Description |
| --- | --- |
| Secondary server | An asynchronously replicated instance that provides read scaling within the cluster. Secondary servers are made up of Read Replica instances. |
| Seed | A file used to create 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. |

# Chapter 10. Fabric

This chapter describes the following:


- Introduction
- Configuration
- Queries
- Further Considerations
- Sharding data with the `copy` command


## 10.1. Introduction

### 10.1.1. Overview

> **Note:**
>
> *Composite databases* are introduced in Neo4j 5 as an implementation of the Fabric technology. As such, in Neo4j 5, Composite databases replace what is known as Fabric in Neo4j 4.x. For more information, see Composite databases in version 5 of the Neo4j Operations Manual.

Fabric, introduced in Neo4j 4.0, is a way to store and retrieve data in multiple databases, whether they are on the same Neo4j DBMS or in multiple DBMSs, using a single Cypher query. Fabric achieves a number of desirable objectives:

- a unified view of local and distributed data, accessible via a single client connection and user session
- increased scalability for read/write operations, data volume and concurrency
- predictable response time for queries executed during normal operations, a failover or other infrastructure changes
- High Availability and No Single Point of Failure for large data volume.

In practical terms, Fabric provides the infrastructure and tooling for:

- **Data Federation**: the ability to access data available in distributed sources in the form of **disjointed graphs**.
- **Data Sharding**: the ability to access data available in distributed sources in the form of a **common graph partitioned on multiple databases**.

With Fabric, a Cypher query can store and retrieve data in multiple federated and sharded graphs.

If you want to connect a single database to your DBMS that is shared over all instances of your DBMS, consider using Remote Database Aliases instead.

## 10.1.2. Fabric concepts

### The fabric database

A Fabric setup includes a *Fabric virtual database*, which acts as the entry point to a federated or sharded graph infrastructure. This database is the execution context in which multi-graph queries can be executed. Drivers and client applications access and use the Fabric execution context by naming it as the selected database for a session. For more information, see *Databases and execution context* in the Neo4j Driver manuals.

The Fabric virtual database (execution context) differs from normal databases in that it cannot store any data, and only relays data stored elsewhere. The Fabric virtual database can be configured on a standalone Neo4j DBMS only, i.e. on a Neo4j DBMS where the configuration setting `dbms.mode` must be set to `SINGLE`.



**Note:**

The Neo4j Admin commands cannot be applied to the Fabric virtual database. They must be run directly on the databases that are part of the Fabric setup.

### Fabric graphs

In a Fabric virtual database, data is organized in the form of graphs. Graphs are seen by client applications as local logical structures, where physically data is stored in one or more databases. Databases accessed as Fabric graphs can be local, i.e in the same Neo4j DBMS, or they can be located in external Neo4j DBMSes. The databases are also accessible by client applications from regular local connections in their respective Neo4j DBMSs.

## 10.1.3. Deployment examples

Fabric constitutes an extremely versatile environment that provides scalability and availability with no single point of failure in various topologies. Users and developers may use applications that can work on a standalone DBMS as well on a very complex and largely distributed infrastructure without the need to apply any change to the queries accessing the Fabric graphs.

### Development deployment

In its simplest deployment, Fabric can be used on a single instance, where Fabric graphs are associated to local databases. This approach is commonly used by software developers to create applications that will be deployed on multiple Neo4j DBMSs, or by power users who intend to execute Cypher queries against local disjoint graphs.

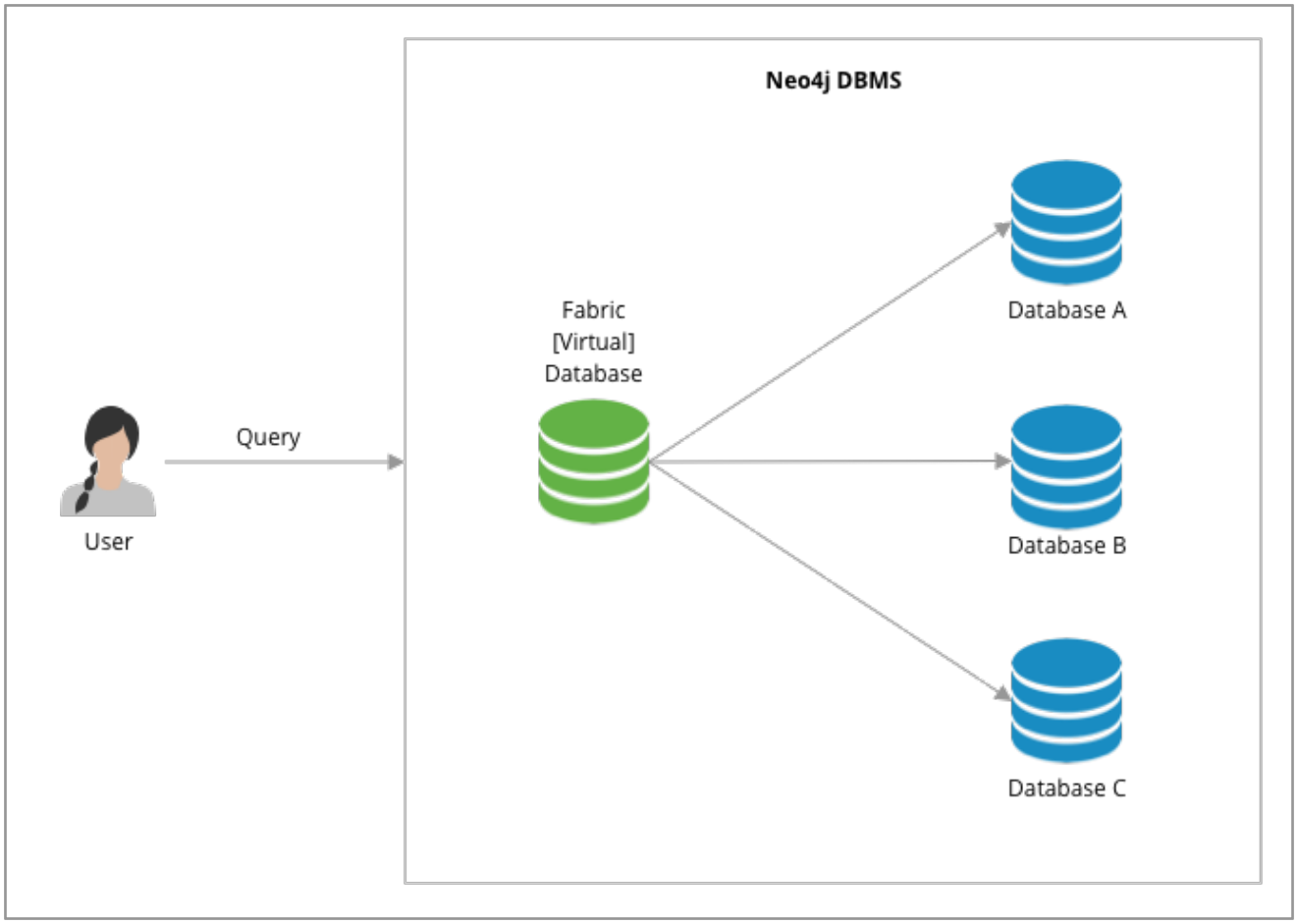



*Figure 8. Fabric deployment in a single instance*

## Cluster deployment with no single point of failure

In this deployment Fabric guarantees access to disjoint graphs in high availability with no single point of failure. High availability is reached by creating redundant entry points for the Fabric Database (i.e. two standalone Neo4j DBMSs with the same Fabric configuration) and a minimum Causal Cluster of three members for data storage and retrieval. This approach is suitable for production environments and it can be used by power users who intend to execute Cypher queries against disjoint graphs.

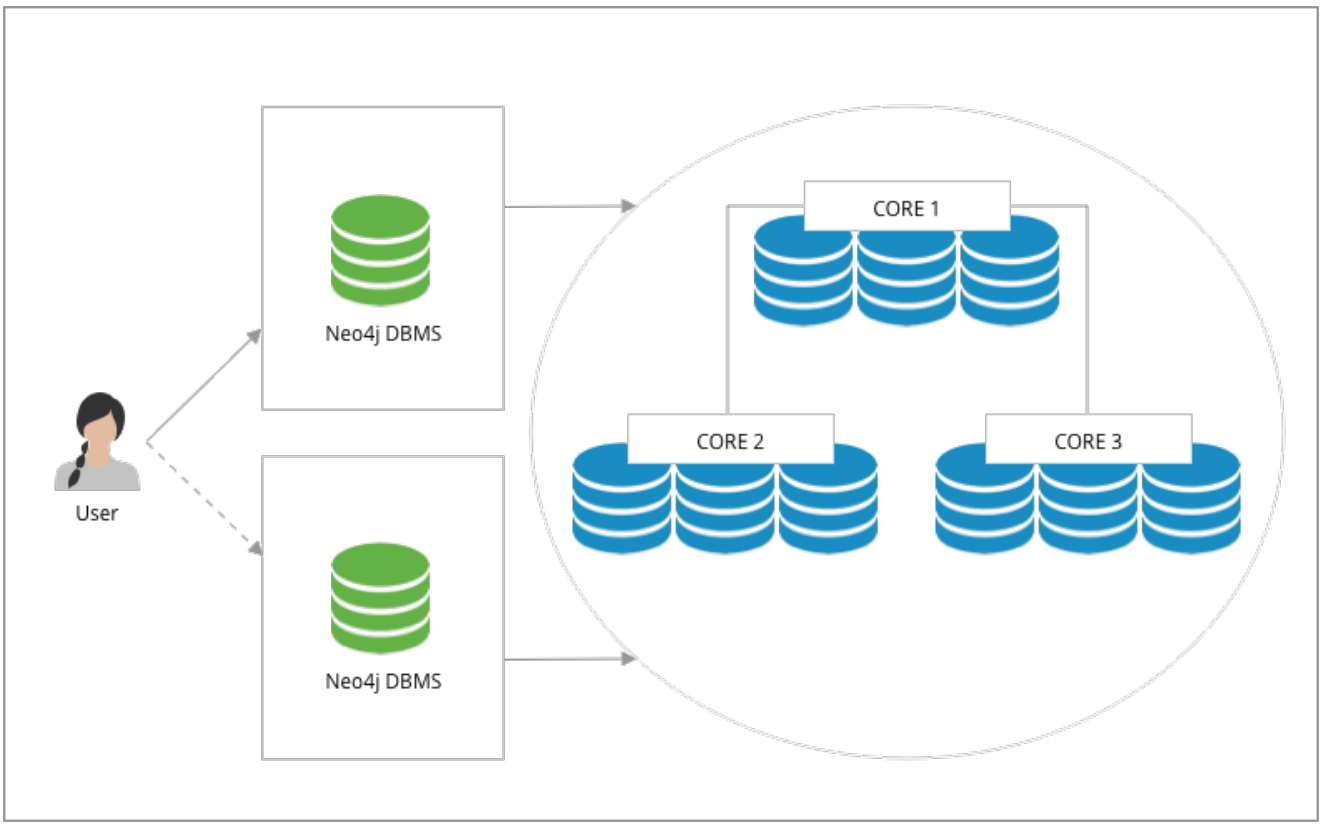



*Figure 9. Fabric deployment with no single point of failure*

## Multi-cluster deployment

In this deployment Fabric provides high scalability and availability with no single point of failure. Disjoint clusters can be sized according to the expected workload and Databases may be colocated in the same cluster or they can be hosted in their own cluster to provide higher throughput. This approach is suitable for production environments where database can be sharded, federated or a combination of the two.

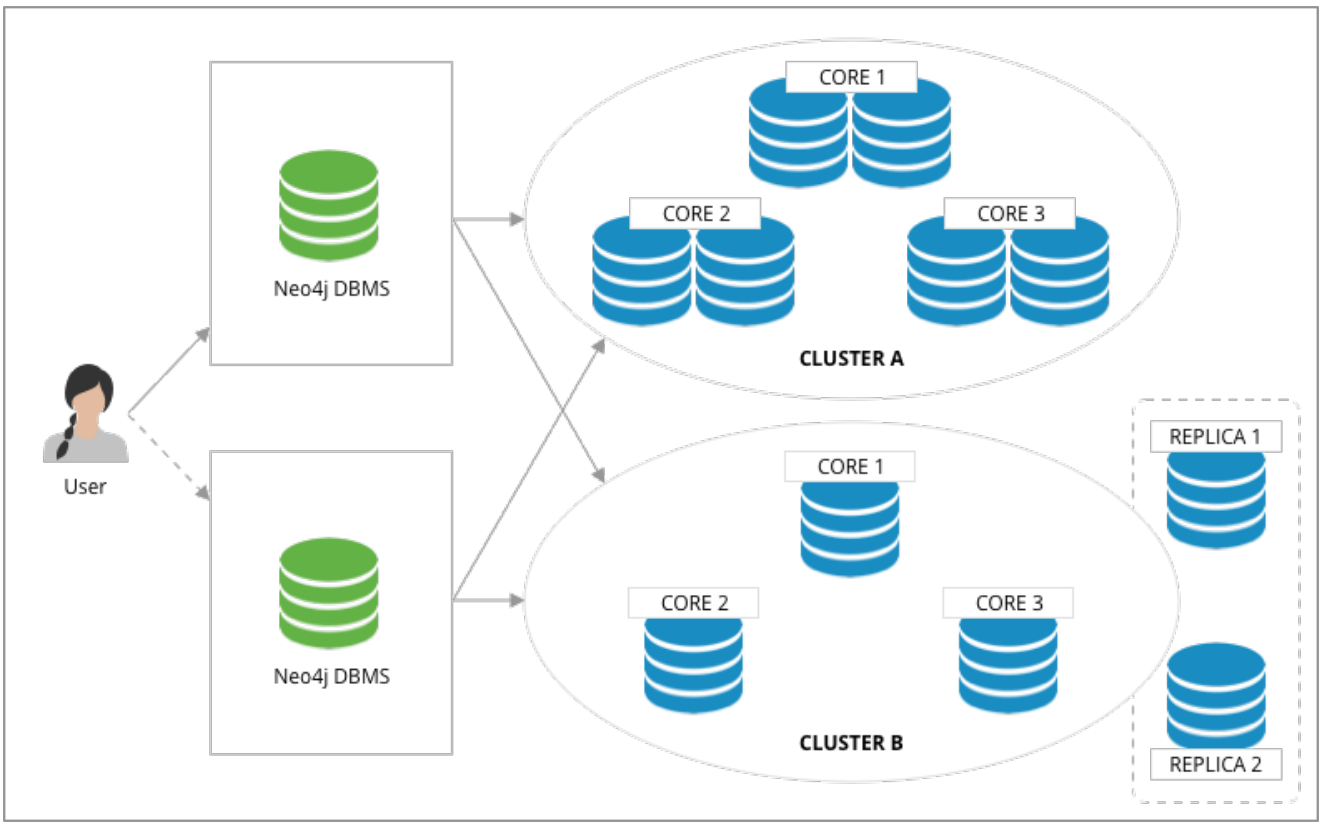



*Figure 10. Fabric deployment for scalability with no single point of failure*

# 10.2. Configuration

## 10.2.1. Fabric database setup

Fabric must be set on a standalone Neo4j DBMS: the settings in *neo4j.conf* are identified by the `fabric` namespace. The minimal requirements to setup Fabric are:

- A **virtual database name**: this is the entry point used by the client applications to access the Fabric environment.
- One or more **Fabric graph URI and database**: this a reference of a URI and a database for each graph set in the Fabric environment.

### Local development setup example

Consider a standalone Neo4j DBMS, which has two databases, `db1` and `db2`. Note that all databases except for the default and `system` must be created using the `CREATE DATABASE` command.

Fabric is enabled by configuring:

```
fabric.database.name=example
```

This configuration enables Fabric and exposes the feature under the virtual database with the name `example`, which is accessible using the default URI, *neo4j://localhost:7687*. After connecting to the DBMS with the `example` database selected, you can run queries like the following:

```
USE db1
MATCH (n) RETURN n
 UNION
USE db2
MATCH (n) RETURN n
```

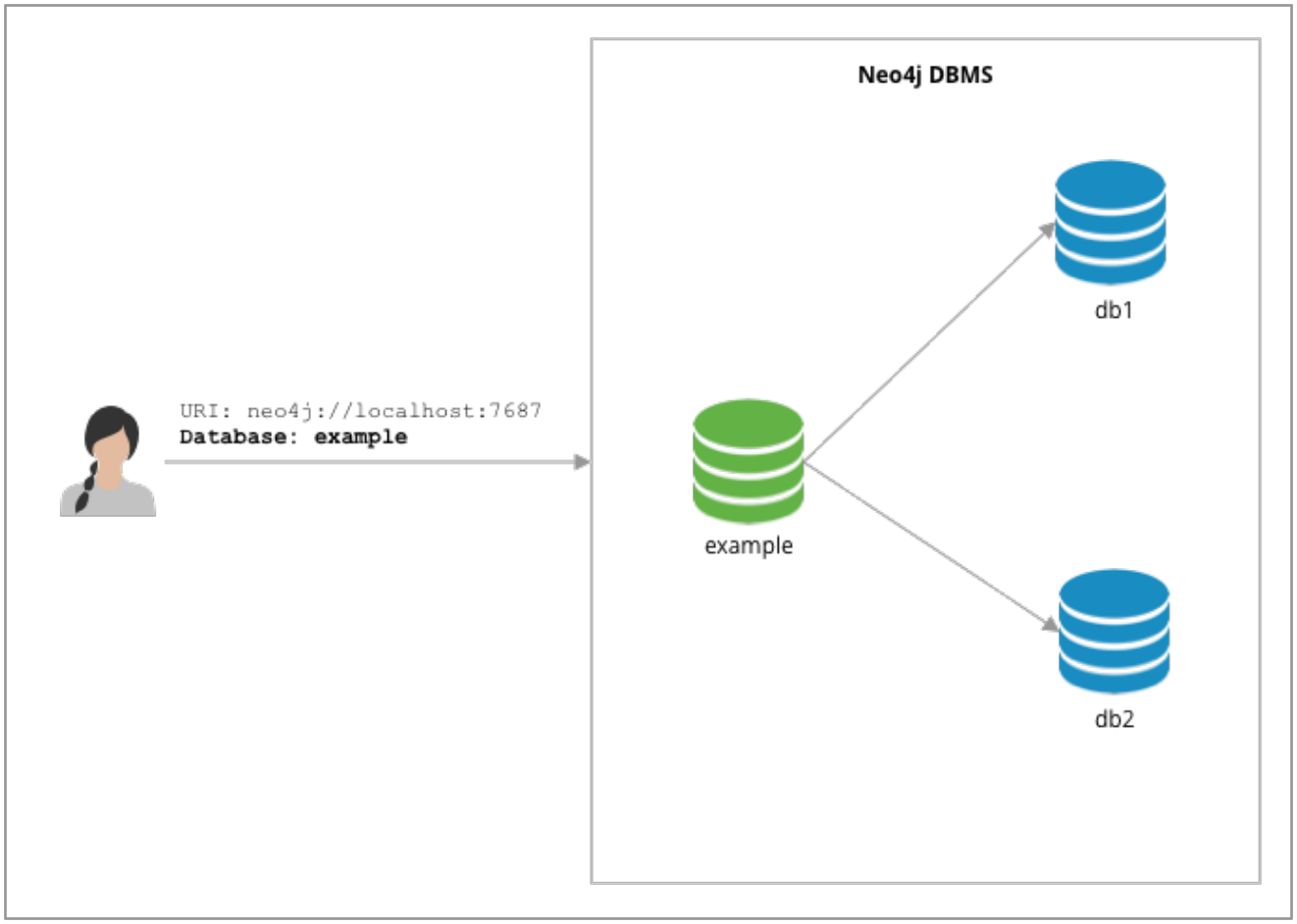



*Figure 11. Minimal local Fabric setting in a development setup*

## Remote development setup example

This example consists of a setup with three standalone Neo4j DBMSs. One instance acts as the Fabric proxy, configured to enable Fabric. The other two instances contain the databases `db1` and `db2`.

The following configuration enables Fabric on the proxy instance and allows it to access the databases in the other two instances.

```
fabric.database.name=example
fabric.graph.0.uri=neo4j://hostname-of-instance1:7687
fabric.graph.0.database=db1

fabric.graph.1.uri=neo4j://hostname-of-instance2:7687
fabric.graph.1.database=db2
```

This configuration enables Fabric and exposes the feature under the virtual database named example, which is accessible using the default URI, *neo4j://localhost:7687*. The Fabric graphs are uniquely identified by their IDs, `0` and `1`.

After connecting to the DBMS with the selected database set to "example", you can run queries like the

following:

```
USE example.graph(0)
MATCH (n) RETURN n
  UNION
USE example.graph(1)
MATCH (n) RETURN n
```

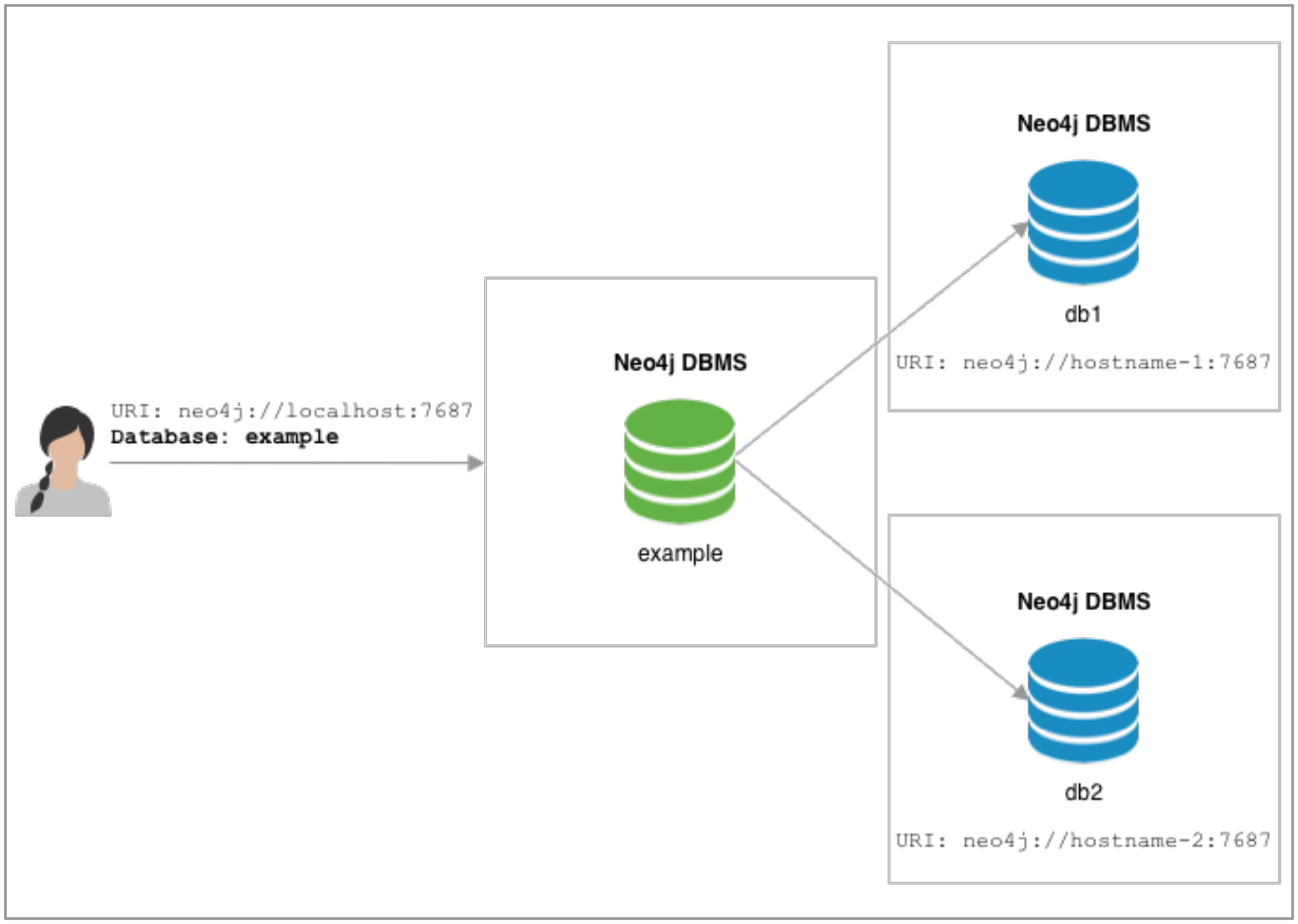



*Figure 12. Minimal remote Fabric setting in a development setup*

## Naming graphs

Graphs can be identified by their ID or by a name. A graph can be named by adding an extra configuration setting, `fabric.graph.<ID>.name`.

For example, if the given names are `graphA` (associated to `db1`) and `graphB` (associated to `db2`), the two additional settings will be:

```
fabric.graph.0.name=graphA
fabric.graph.1.name=graphB
```

Giving names to graphs means you can refer to them by name in queries:

```
USE example.graphA
MATCH (n) RETURN n
 UNION
USE example.graphB
```

```
MATCH (n) RETURN n
```

## Cluster setup with no single point of failure example

In this example, all components are redundant and data is stored in a Causal Cluster. In addition to the settings described in the previous example, a setting with no single point of failure requires the use of the *routing servers* parameter, which specifies a list of standalone Neo4j DBMSs that expose the same Fabric database and configuration. This parameter is required in order to simulate the same connectivity that client applications use with Causal Cluster, which means, in case of fault of one instance, the client application may revert to another existing instance.

Assume that in this example, the data is stored in three databases: `db1`, `db2` and `db3`. The configuration of Fabric would be:

```
dbms.mode=SINGLE

fabric.database.name=example
fabric.routing.servers=server1:7687,server2:7687

fabric.graph.0.name=graphA
fabric.graph.0.uri=neo4j://core1:7687,neo4j://core2:7687,neo4j://core3:7687
fabric.graph.0.database=db1

fabric.graph.1.name=graphB
fabric.graph.1.uri=neo4j://core1:7687,neo4j://core2:7687,neo4j://core3:7687
fabric.graph.1.database=db2

fabric.graph.2.name=graphC
fabric.graph.2.uri=neo4j://core1:7687,neo4j://core2:7687,neo4j://core3:7687
fabric.graph.2.database=db3
```

The configuration above must be added to the *neo4j.conf* file of the Neo4j DBMSs `server1` and `server2`. The parameter `fabric.routing.servers` contains the list of available standalone Neo4j DBMSs hosting the Fabric database. The parameter `fabric.graph.<ID>.uri` can contain a list of URIs, so in case the first server does not respond to the request, the connection can be established to another server that is part of the cluster. The URIs refer to the `neo4j://` schema so that Fabric can retrieve a routing table and can use one of the members of the cluster to connect.

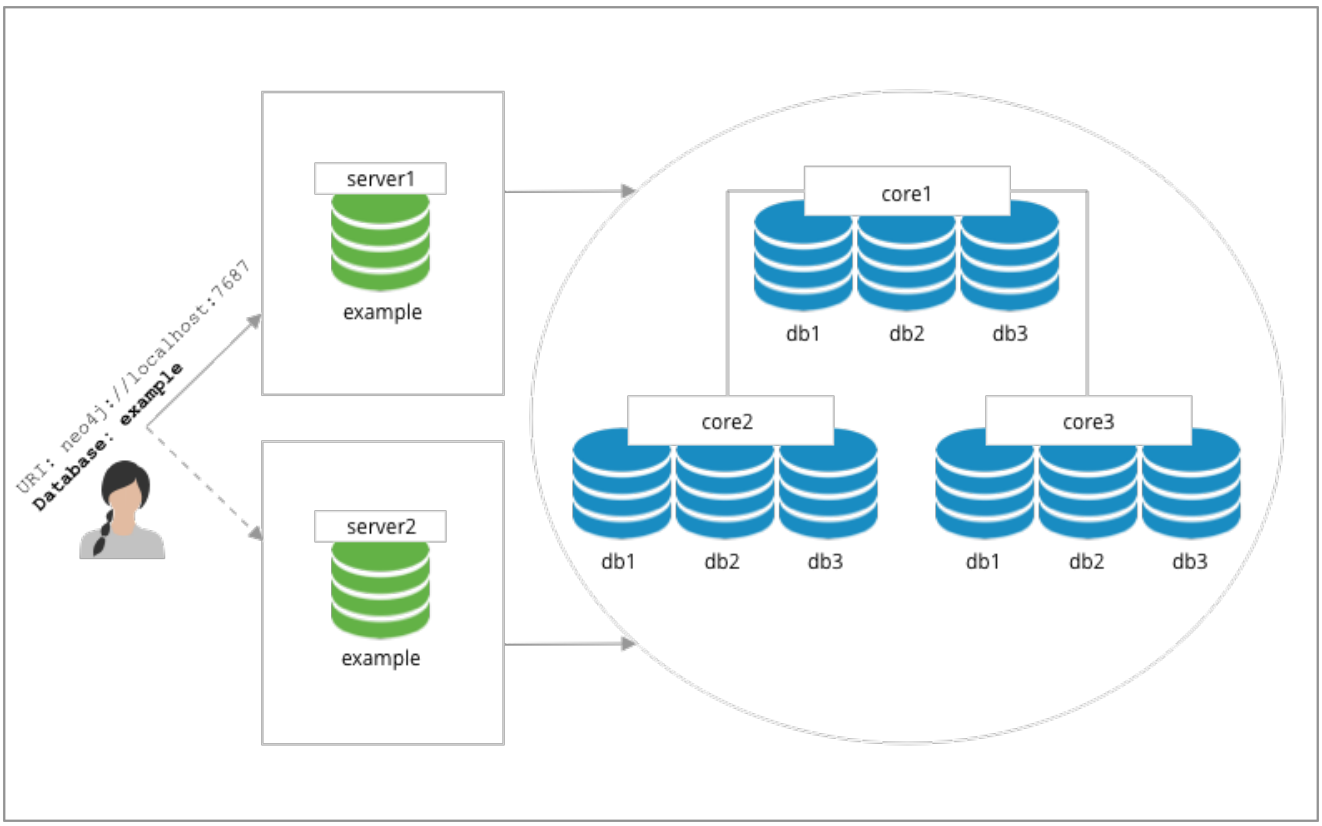



*Figure 13. Fabric setting with Causal Cluster and no single point of failure*

### Cluster routing context

The URIs in the graph settings may include routing contexts, which are described in the Neo4j Driver manuals. This can be used to associate a Fabric graph with a filtered subset of Causal Cluster members, by selecting a routing policy.

As an example, assuming you have a server policy called `read_replicas` defined in the configuration of the cluster you are targeting, you might set up a Fabric graph that accesses only the read replicas of the cluster.

```
fabric.graph.0.name=graphA
fabric.graph.0.uri=neo4j://core1:7687?policy=read_replicas
fabric.graph.0.database=db1
```

This enables scenarios where queries executed through Fabric are explicitly offloaded to specific instances in clusters.

## 10.2.2. Authentication and authorization

### Credentials

Connections between the Fabric database and the Neo4j DBMSs hosting the data are created using the same credentials that are supplied in the client connection to the Fabric database. It is recommended to maintain a set of user credentials on all the Neo4j DBMSs; if required, a subset of credentials may be set for local access on the remote DBMSs.

### User and role administration

User and role administration actions are not automatically propagated to the Fabric environment, therefore security settings must be executed on any DBMS that is part of Fabric.

### Privileges on the Fabric database

In order to use all Fabric features, users of Fabric databases need `ACCESS` and `READ` privileges.

## 10.2.3. Important settings

This section provides general information about Fabric settings and describes the ones important for creating a fabric set-up. For the full list of Fabric configuration options, see Configuration settings.

Fabric settings are divided in the following categories:

- **System Settings**: DBMS-level settings.
- **Graph Settings**: definition and configuration of Fabric graphs.
- **Drivers Settings**: configuration of drivers used to access Neo4j DBMSs and databases associated to Fabric graphs.

### System settings

*Table 437. Fabric system settings*

| Parameter | Description |
| --- | --- |
| `fabric.database.name` | Name of the Fabric database. Neo4j Fabric currently supports one Fabric database in a standalone Neo4j DBMS. |
| `fabric.routing.servers` | A comma-separated list of Neo4j DBMSs that share the same Fabric configuration. These DBMSs form a routing group. A client application will route transactions through a Neo4j driver or connector to one of the members of the routing group. A Neo4j DBMS is represented by its Bolt connector address. Example: `"fabric.routing.servers=server1:7687,server2:7687"`. |

### Graph settings



The `<ID>` in the following settings is the integer associated to each Fabric graph.

*Table 438. Fabric graph settings*

| Parameter | Description |
| --- | --- |
| `fabric.graph.<ID>.uri` | URI of the Neo4j DBMS hosting the database associated to the Fabric graph. Example: `neo4j://somewhere:7687` |

| Parameter | Description |
| --- | --- |
| `fabric.graph.<ID>.database` | Name of the database associated to the Fabric graph. |
| `fabric.graph.<ID>.name` | Name assigned to the Fabric graph. The name can be used in Fabric queries. |
| `fabric.graph.<ID>.driver.*` | Any specific driver setting, that means, any setting related to a connection to a specific Neo4j DBMS and database. This setting overrides a global driver setting. |



When configuring access to a remote DBMS, make sure that the remote is configured to advertise its address correctly, using either `dbms.default_advertised_address` or `dbms.connector.bolt.advertised_address`. Fabric reads the routing table from the remote DBMS and then connects back using an appropriate entry in that table.

## Drivers settings

Fabric uses the Neo4j Java driver to connect to and access the data stored in Neo4j databases associated to Fabric graphs. This section presents the most important parameters available to configure the driver.

Drivers settings are configured with parameters with names of the format:

`fabric.driver.<suffix>`

A setting can be global, i.e. be valid for all the drivers used in Fabric, or it can be specific for a given connection to a Neo4j database associated to a graph. The graph-specific setting overrides the global configuration for that graph.

*Example 58. Global drivers setting versus graph-specific drivers setting*

A drivers setting for Fabric as the following is valid for all the connections established with the Neo4j DBMSs set in Fabric:

```
fabric.driver.api=RX
```

A graph-specific connection for the database with `ID=6` will override the `fabric.driver.api` setting for that database:

```
fabric.graph.6.driver.api=ASYNC
```

*Table 439. Fabric drivers setting suffixes*

| Parameter suffix | Explanation |
| --- | --- |
| `ssl_enabled` | SSL for Fabric drivers is configured using the `fabric` SSL policy. This setting can be used to instruct the driver not to use SSL even though the `fabric` SSL policy is configured. The driver will use SSL if the `fabric` SSL policy is configured, and this setting is set to `true`. This parameter can only be used in `fabric.graph.<graph ID>.driver.ssl_enabled` and not `fabric.driver.ssl_enabled`. |
| `api` | Determine which driver API to be used. Supported values are `RX` and `ASYNC`. |



Most driver options described in *Configuration* in the Neo4j Driver manuals have an equivalent in Fabric configuration.

## 10.3. Queries

In this section we will look at a few example queries that show how to perform a range of different tasks.

The examples in this section make use of the two Cypher clauses: `USE` and `CALL {}`. The syntax is explained in detail in the Cypher Manual:

- See Cypher Manual → CALL {} for details about the `CALL {}` clause.
- See Cypher Manual → USE for details about the `USE` clause.

### 10.3.1. Query a single graph

*Example 59. Reading and returning data from a single graph.*

```
USE example.graphA
MATCH (movie:Movie)
RETURN movie.title AS title
```

The `USE` clause at the beginning of the query causes the rest of the query to execute against the `example.graphA` graph.

### 10.3.2. Query multiple graphs

*Example 60. Reading and returning data from two named graphs*

```
USE example.graphA
MATCH (movie:Movie)
RETURN movie.title AS title
  UNION
USE example.graphB
MATCH (movie:Movie)
RETURN movie.title AS title
```

The first part of the `UNION` query executes against the `example.graphA` graph and the second part executes against the `example.graphB` graph.

## 10.3.3. Query all graphs

*Example 61. Reading and returning data from all graphs*

```
UNWIND example.graphIds() AS graphId
CALL {
  USE example.graph(graphId)
  MATCH (movie:Movie)
  RETURN movie.title AS title
}
RETURN title
```

We call the built-in function `example.graphIds()` to get the graph IDs for all remote graphs in our Fabric setup. We `UNWIND` the result of that function to get one record per graph ID. The `CALL {}` subquery is executed once per incoming record. We use a `USE` clause in the subquery with a dynamic graph lookup, causing the subquery to execute once against each remote graph. At the end of the main query we simply `RETURN` the title variable.

## 10.3.4. Query result aggregation

*Example 62. Getting the earliest release year of all movies in all graphs*

```
UNWIND example.graphIds() AS graphId
CALL {
  USE example.graph(graphId)
  MATCH (movie:Movie)
  RETURN movie.released AS released
}
RETURN min(released) AS earliest
```

From each remote graph we return the released property of each movie. At the end of the main query we aggregate across the full result to calculate the global minimum.

## 10.3.5. Correlated subquery

*Example 63. Correlated subquery*

Assume that `graphA` contains American movies and `graphB` contains European movies. Find all European movies released in the same year as the latest released American movie:

```
CALL {
  USE example.graphA
  MATCH (movie:Movie)
  RETURN max(movie.released) AS usLatest
}
CALL {
  USE example.graphB
  WITH usLatest
  MATCH (movie:Movie)
  WHERE movie.released = usLatest
  RETURN movie
}
RETURN movie
```

We query the `example.graphA` and return the release year of the latest release. We then query the `example.graphB`. `WITH usLatest` is an import clause which lets us refer to the `usLatest` variable inside the subquery. We find all the movies in this graph that fulfill our condition and return them.

It is not possible to switch the current graph in a nested query. For example, the following query is illegal:

*Example 64. Illegal correlated subquery*

```
USE example.graphA
MATCH (movie:Movie)
WITH movie.title AS title
CALL {
  USE example.graphB // Cannot swicth from example.graphA
  WITH title
  MATCH (otherMovie:Movie)
  WHERE otherMovie.title STARTS WITH title
  RETURN otherMovie.title AS otherTitle
}
RETURN title, otherTitle
```

This limitation can be circumvented by having subqueries after one another, but without nesting them.

## 10.3.6. Updating query

*Example 65. Create a new movie node*

```
USE example.graphB
CREATE (m:Movie)
SET m.title = 'Léon: The Professional'
SET m.tagline = 'If you want the job done right, hire a professional.'
SET m.released = 1994
```

## 10.3.7. Mapping functions

Mapping functions are a common Fabric usage pattern. In the previous examples, graphs were identified by providing static graph names in the query. Fabric may be used in scenarios where graphs are identified by a mapping mechanism that can, for example, identify a key of an object contained in a graph. This can be achieved by using user-defined functions or other functions that may be already available. These functions ultimately return the ID of a graph in Fabric.

Mapping functions are commonly used in sharding scenarios. In Fabric, shards are associated to graphs, hence mapping functions are used to identify a graph, i.e. a shard.



Refer to Java Reference → User-defined functions for details on how to create user-defined functions.

Let's assume that Fabric is setup in order to store and retrieve data associated to nodes with the label `user`. User nodes are partitioned in several graphs (shards) in Fabric. Each `user` has a numerical `userId`, which is unique in all Fabric. We decide on a simple scheme where each `user` is located on a graph determined by taking the `userId` modulo the number of graphs. We create a user-defined function which

implements the following pseudo code:

```
sharding.userIdToGraphId(userId) = userId % NUM_SHARDS
```

Assuming we have supplied a query parameter `$userId` with the specific userId that we are interested in, we use our function in this way:

```
USE example.graph( sharding.userIdToGraphId($userId) )
MATCH (u:User) WHERE u.userId = $userId
RETURN u
```

### 10.3.8. Fabric built-in functions

Fabric functions are located in a namespace corresponding to a Fabric database in which they are used. The following table provides a description of Fabric built-in functions:

*Table 440. Fabric built-in functions*

| Function | Explanation |
| --- | --- |
| `<fabric database name>.graphIds()` | Provides a list of IDs of all remote graph configured for the given Fabric database. |
| `<fabric database name>.graph(graphId)` | Maps a graph ID to a Graph. It accepts a graph ID as a parameter and returns a graph representation accepted by USE clause. This function is supported only in `USE` clauses |

## 10.4. Further considerations

*DBMS mode*

The DBMS hosting the Fabric virtual database cannot be part of a Causal Cluster: it can only be a DBMS with `dbms.mode=SINGLE`.

*Sharding an existing database*

An existing database can be sharded with the help of the `neo4j-admin copy` command. See Sharding data with the copy command for an example.

*Database compatibility*

Fabric is part of Neo4j DBMS and does not require any special installation or plugin. Fabric databases can be associated to databases available on Neo4j DBMS version 4.1 or 4.2.

*Fabric configuration*

The Neo4j DBMSs that host the same Fabric virtual database must have the same configuration settings. The configuration must be kept in-sync and applied by the Database Administrator.

*Security credentials*

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.

*Administration commands*

Fabric does not support running Cypher administration commands on or through the Fabric virtual database. Any database management commands, index and constraint management commands, or user and security management commands must be issued directly to the DBMSs and databases that are part of the Fabric setup.

*Neo4j embedded*

Fabric is not available when Neo4j is used as an embedded database in Java applications. Fabric 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.

## 10.5. Sharding data with the `copy` command

The `copy` command can be used to filter out data for a Fabric installation. In the following example, a sample database is separated into 3 shards.

*Example 66. Use the `copy` command to filter out data for a Fabric installation.*

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:

```
$neo4j-home> bin/neo4j-admin copy --from-database=neo4j \
   --to-database=shard1 \
   --keep-only-nodes-with-labels=S1,SAll \
   --skip-labels=S1,S2,S3,SAll
```

   The `--keep-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*:

```
$neo4j-home> bin/neo4j-admin copy --from-database=neo4j \
   --to-database=shard2 \
   --keep-only-nodes-with-labels=S2,SAll \
   --skip-labels=S1,S2,S3,SAll \
   --keep-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 `--keep-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 `--keep-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 `--keep-only-node-properties`.

```
$neo4j-home> bin/neo4j-admin copy --from-database=neo4j \
   --to-database=shard3 \
   --keep-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 `--keep-only-node-properties`. In this example, it is easier to use `--keep-only-node-properties` because only one property should be kept. The relationship property filters works in the same way.

# Chapter 11. 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.
- Back up an online database — How to back up an online database.
- Prepare for restore - How to prepare your backup for restore by applying the latest transactions.
- Restore a database backup — How to restore a database backup in a live Neo4j deployment.
- 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.
- Copy a database store — How to copy data store from an existing database to a new database.

## 11.1. 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.

### 11.1.1. 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).
  - automatically check the consistency of a database backup (online only).
  - perform incremental backups (online only).

    > **Note:**
    >
    > The incremental backup strategy is configured by the `dbms.backup.incremental.strategy` setting.
    >
    > The default strategy is `UNBOUNDED`, meaning that the server will send transactions until all committed transactions have been sent, even those committed after the start of the backup job. This strategy is suitable for a low ingest rate environment, where the backup network throughput is high enough to keep up with the transaction rate. For the `UNBOUNDED` strategy to be effective, the backup and database files must be on the same filesystem for the backup to catch up with as few transactions as possible.
    >
    > In an environment with a high ingest rate and a low backup network throughput, it is advisable to use the `START_TIME` strategy, which sends only the transactions committed before the start of the backup job. Otherwise, the backup might hang for a long time, trying to catch up as transactions continue to checkpoint and then be committed to the database.

  - use SSL/TLS for the backup network communication (online only).
  - keep your databases as archive files (offline only).
- 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.

  > **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.

## 11.1.2. Backup and restore options

Neo4j supports backing up and restoring both online and offline databases. It uses Neo4j Admin tool commands, which can be run from a live, as well as from an offline Neo4j DBMS. All `neo4j-admin` commands must be invoked as the `neo4j` user to ensure the appropriate file permissions.

- `neo4j-admin backup/restore` (Enterprise only) -– used for performing online backup (full and

incremental) and restore operations. The database to be backed up must be in **online** mode. This command is suitable for production environments, where you cannot afford downtime. However, it is more memory intensive and is not supported in Neo4j Aura.

> **Tip:**
>
> When using `neo4j-admin backup` in Causal Cluster, it is recommended to back up from an external instance as opposed to reuse instances that form part of the cluster.

- `neo4j-admin dump/load` -- used for performing offline dump and load operations. The database to be dumped must be in **offline** mode. This dump command is suitable for environments, where downtime is not a factor. It is faster than the backup command, and produces an archive file, which occupies less space than a normal database structure.
- `neo4j-admin copy` -- used for copying an offline database or backup. This command can be used for cleaning up database inconsistencies, reclaiming unused space, and migrating Neo4j 3.5.any directly to any 4.x version of Neo4j, including the latest version, skipping the intermediate steps. For a detailed example, see Upgrade and Migration Guide → Tutorial: Back up and copy a database in a standalone instance.

> **Warning:**
>
> File system copy-and-paste of databases is not supported and may result in unwanted behavior, such as corrupt stores.

> **Note:**
>
> In Fabric deployments, the Neo4j Admin commands `backup`, `restore`, `dump`, `load`, `copy`, and `check-consistency` are not supported for use on the Fabric virtual database. They must be run directly on the databases that are part of the Fabric setup.

*Table 441. The following table describes the commands capabilities and usage.*

| Capability/ Usage | `neo4j-admin backup` | `neo4j-admin dump` | `neo4j-admin restore` | `neo4j-admin load` | `neo4j-admin copy` |
|---|---|---|---|---|---|
| Neo4j Edition | Enterprise | all | Enterprise | all | Enterprise |
| Live Neo4j DBMS | ✔ | ✔ | ✔ | ✔ | ✔ |
| Offline Neo4j DBMS | ✖ | ✔ | ✔ | ✔ | ✔ |
| Run against a user database | ✔ | ✔ | ✔ | ✔ | ✔ |
| Run against the `system` database | ✔ | ✔ | ✔ | ✔ | ✖ |

| Capability/ Usage | `neo4j-admin backup` | `neo4j-admin dump` | `neo4j-admin restore` | `neo4j-admin load` | `neo4j-admin copy` |
|---|---|---|---|---|---|
| Run against the `fabric` database | ✗ | ✗ | ✗ | ✗ | ✗ |
| Perform full backups | ✓ | ✓ | n/a | n/a | n/a |
| Perform incremental backups | ✓ | ✗ | n/a | n/a | n/a |
| Applied to an online database | ✓ | ✗ | ✗ | ✗ | ✗ |
| Applied to an offline database | ✗ | ✓ | ✓ | ✓ | ✓ |
| Can be run remotely (support SSL) | ✓ | ✗ | ✓ | ✗ | ✓ |
| Command input | database | database | database backup | archive (.dump) | database or database backup |
| Command output | database | archive (.dump) | database | database | database; no schema store |
| Run consistency check after completion | ✓ | ✗ | ✗ | ✗ | ✗ |
| Clean up database inconsistencies | ✗ | ✗ | ✗ | ✗ | ✓ |
| Compact data store | ✗ | ✗ | ✗ | ✗ | ✓ |

## 11.1.3. Databases to backup

A Neo4j DBMS can host multiple databases. Both Neo4j Community and Enterprise Editions have a default user database, called `neo4j`, and a `system` database, which contains configurations, e.g., operational states of databases, security configuration, schema definitions, login credentials, and roles. In the Enterprise Edition, you can also create additional user databases. Each of these databases are backed up independently of one another.



**Note:**

It is very important to back up each of your databases, including the `system` database, in a

safe location.

### 11.1.4. 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.

### 11.1.5. 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.

## 11.2. Backup modes

The backup client can operate in two different modes – a *full backup* and an *incremental backup*.

### 11.2.1. Full backup

A full backup is always required initially for the very first backup into a target location.



**Note:**

The full backup can be run against both an **online** (using `neo4j-admin backup`) and an **offline** (using `neo4j-admin dump`) database.

*Example 67. Full backup against an online database*

```
$neo4j-home> export HEAP_SIZE=2G
$neo4j-home> mkdir /mnt/backups
$neo4j-home> bin/neo4j-admin backup --from=192.168.1.34 --backup-dir=/mnt/backups/neo4j --database
=neo4j --pagecache=4G
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
...
...
...
```

For more information about online backup options and how to control memory usage, see Back up an online database.



**Tip:**

For more information about performing a full backup against an **offline** database, see Back up an offline database.

### 11.2.2. Incremental backup

After the initial full backup, the subsequent backups attempt to use the incremental mode, where just the delta of the transaction logs since the last backup is transferred and applied to the target location. If the required transaction logs are not available on the backup server, then the backup client falls back on performing a full backup instead, unless `--fallback-to-full` is disabled. The incremental backup can be run only against an **online** database.



**Note:**

The incremental backup strategy is configured by the `dbms.backup.incremental.strategy` setting.

The default strategy is `UNBOUNDED`, meaning that the server will send transactions until all committed transactions have been sent, even those committed after the start of the backup job. This strategy is suitable for a low ingest rate environment, where the backup network throughput is high enough to keep up with the transaction rate. For the `UNBOUNDED` strategy to be effective, the backup and database files must be on the same filesystem for the backup to catch up with as few transactions as possible.

In an environment with a high ingest rate and a low backup network throughput, it is advisable to use the `START_TIME` strategy, which sends only the transactions committed before the start of the backup job. Otherwise, the backup might hang for a long time, trying to catch up as transactions continue to checkpoint and then be committed to the database.

*Example 68. Incremental backup against an online database*

```
$neo4j-home> export HEAP_SIZE=2G
$neo4j-home> bin/neo4j-admin backup --from=192.168.1.34 --backup-dir=/mnt/backups/neo4j --database
=neo4j --pagecache=4G
Destination is not empty, doing incremental backup...
Backup complete.
```

For more information about online backup options and how to control memory usage, see Back up an online database.

## 11.3. Back up an online database



**Caution:**

Remember to plan your backup carefully and to back up each of your databases, including the `system` database.

## 11.3.1. 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.

### Usage

The `neo4j-admin backup` command can be used for performing both full and incremental backups of an **online** database. The command can be run both locally and remotely. By default, `neo4j-admin backup` also checks the database consistency at the end of every backup operation. 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 backup` command also supports SSL/TLS. For more information, see Online backup configurations.



**Note:**

`neo4j-admin backup` is not supported in Neo4j Aura.

`neo4j-admin backup` is not supported for use on the Fabric virtual database. It must be run directly on the databases that are part of the Fabric setup.

### Syntax

```
neo4j-admin backup  --backup-dir=<path>
                    [--verbose]
                    [--expand-commands]
                    [--from=<host:port>]
                    [--database=<database>]
                    [--fallback-to-full=<true/false>]
                    [--pagecache=<size>]
                    [--check-consistency=<true/false>]
                    [--report-dir=<path>]
                    [--check-graph=<true/false>]
                    [--check-indexes=<true/false>]
                    [--check-index-structure=<true/false>]
                    [--check-label-scan-store=<true/false>]
                    [--check-property-owners=<true/false>]
                    [--additional-config=<path>]
                    [--include-metadata=<all/users/roles>]
                    [--prepare-restore=<true/false>]
                    [--parallel-recovery=<true/false>]
```



**Note:**

Please note that the following options have been deprecated:

```
[--check-label-scan-store=<true/false>]
[--check-property-owners=<true/false>]
```

Values for these settings will be ignored.

## Options

| Option | Default | Description |
|---|---|---|
| `--backup-dir` | | Target directory. |
| `--verbose` | | Enable verbose output. |
| `--expand-commands` | | Allow command expansion in config value evaluation. |
| `--from` | `localhost:6362` | Host and port of Neo4j. |
| `--database` | `neo4j` | Name of the remote database to back up.<br><br>The value can contain `*` and `?` for globbing, in which cases, all matching databases will be backed up.<br><br>**Tip:**<br><br>With a single `*` as a value, you can back up all the databases of the DBMS. |
| `--fallback-to-full` | `true` | If an incremental backup fails, backup will move the old backup to `<name>.err.<N>` and fallback on a full backup instead. |
| `--pagecache` | `8m` | The size of the page cache to use for the backup process. |
| `--check-consistency` | `true` | Run a consistency check against the database backup. |
| `--report-dir` | `.` | Directory where consistency report will be written. |
| `--check-graph` | `true` | Perform consistency checks between nodes, relationships, properties, types, and tokens. |
| `--check-indexes` | `true` | Perform consistency checks on indexes. |
| `--check-index-structure` | `true` | Perform structure checks on indexes. |
| `--check-label-scan-store` | `true` | This option is deprecated, and its value is ignored. |
| `--check-property-owners` | `false` | This option is deprecated, and its value is ignored. |

| Option | Default | Description |
| --- | --- | --- |
| `--additional-config` | | Configuration file to provide additional or override the existing configuration settings in the *neo4j.conf* file. |
| `--include-metadata` | `Include metadata in the file. This cannot be used for backing up the` `system` `database. Possible values are:`<br><br>`-` `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` `. The metadata script can be found in the backup directory` `<database>/tools/metadata_script.cypher` `. [NOTE] ==== **Note:**`<br><br>`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.`<br><br>`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).`<br><br>`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 |

| Option | Default | Description |
| --- | --- | --- |
| `--prepare-restore` | `true` | Perform the recovery of the backup store by applying the latest pulled transactions. If disabled, the backup will be faster, but a recovery of the backup store will be required at a later time before restoring the data.<br><br>For more information on how to do that, see Prepare a database for restoring.<br><br>**Note:**<br><br>If `--prepare-restore` is set to `false`, `--check-consistency` is implicitly set to `false`, because the consistency of a non-recovered store cannot be checked. |
| `--parallel-recovery` | `false` | Allow multiple threads to apply transactions to a backup in parallel. For some databases and workloads, this may reduce execution times significantly.<br><br>**Note:**<br><br>`parallel-recovery` is an experimental option. Consult Neo4j support before use. |

## Exit codes

Depending on whether the backup was successful or not, `neo4j-admin backup` exits with different codes. The error codes include details of what error was encountered.

*Table 442. Neo4j Admin backup exit codes when backing up one database*

| Code | Description |
| --- | --- |
| `0` | Success. |
| `1` | Backup failed. |
| `2` | Backup succeeded but consistency check failed. |
| `3` | Backup succeeded but consistency check found inconsistencies. |

*Table 443. Neo4j Admin backup exit codes when backing multiple databases*

| Code | Description |
|---|---|
| `0` | All databases are backed up successfully. |
| 1 | One or several backup failed. |

## 11.3.2. 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 `causal_clustering.catch_up_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.

### 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.



**Warning:**

Make this change only if you need the remote backup. If your network is not adequately isolated, this change might expose your system to threats.

*Table 444. Server parameters for backups*

| Parameter name | Default value | Description |
|---|---|---|
| `dbms.backup.enabled` | `true` | Enable support for running online backups. |
| `dbms.backup.listen_address` | `127.0.0.1:6362` | Listening server for online backups. |



**Note:**

It is not recommended to use an NFS mount for backup purposes as this is likely to corrupt and slow down the backup.



**Note:**

Make sure to follow the Security Configurations in order to prevent unauthorized users from accessing the DBMS by having access to the backup server.

## Memory configuration

The following options are available for configuring the memory allocated to the backup client:

*Configure heap size for the backup*

`HEAP_SIZE` configures the maximum heap size allocated for the backup process. This is done by setting the environment 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 for the backup*

The page cache size can be configured by using the `--pagecache` option of the `neo4j-admin backup` command. If not explicitly defined, the page cache defaults to `8MB`.



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 resources configurations

*Consistency checking*

Checking the consistency of the backup is a major operation which may consume significant computational resources, such as, memory, CPU, I/O. When backing up an online database, the consistency checker is invoked at the end of the process by default. Therefore, it is highly recommended to perform the backup and consistency check on a dedicated machine, which has sufficient free resources, to avoid adversely affecting the running server.

Alternatively, you can decouple the backup operation from the consistency check (using the `neo4j-admin backup` option `--check-consistency=false`) and schedule that part of the workflow to happen at a later point in time, on a dedicated machine. Consistency checking a backup is vital for safeguarding and ensuring the quality of the data, and should not be underestimated. For more information, see Consistency checker.



Tip:

To avoid running out of resources on the running server, it is recommended to perform the backup on a separate dedicated machine.

*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 `dbms.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 `dbms.tx_log.rotation.size=1M` and `dbms.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 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.



**Note:**

For a detailed list of recommendations regarding security in Neo4j, see Security checklist.

The following table provides details on how the configured SSL policies map to the configured ports.

*Table 445. Mapping backup configurations to SSL policies*

| Topology | Backup target address on database server | SSL policy setting on database server | SSL policy setting on backup client | Default port |
|---|---|---|---|---|
| Standalone instance | `dbms.backup.listen_address` | `dbms.ssl.policy.backup` | `dbms.ssl.policy.backup` | `6362` |
| Causal cluster | `dbms.ssl.policy.cluster`<br><br>`causal_clustering.transaction_listen_address` | `dbms.ssl.policy.cluster` | `dbms.ssl.policy.backup` | `6000` |



**Note:**

It is very important to ensure that there is no external access to the port specified by the setting `dbms.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, and each server has two configurable ports capable of serving a backup. These ports are configured by `dbms.backup.listen.address` and `causal_clustering.transaction_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 Read Replicas to act as backup servers, since they are more numerous than Core members in typical cluster deployments. Furthermore, the possibility of performance issues on a Read Replica, caused by a large backup, will not affect the performance or redundancy of the Core members. If a Read Replica is not available, then a Core can be selected based on factors, such as its physical proximity, bandwidth, performance, and liveness.



**Note:**

To avoid taking a backup from a cluster 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.

### 11.3.3. Examples

The following are examples of how to back up a single database, e.g., the default database `neo4j`, and multiple databases, using the `neo4j-admin backup` command. The target directory */mnt/backups/neo4j* must exist before calling the command and the database(s) must be online.

*Example 69. Use `neo4j-admin backup` to back up a single database.*

```
bin/neo4j-admin backup --backup-dir=/mnt/backups/neo4j --database=neo4j
```

To backup several databases that match database pattern you can use name globbing. For example, to backup all databases that start with **n** you should run:

*Example 70. Use `neo4j-admin backup` to back up multiple databases.*

```
neo4j-admin backup --from=192.168.1.34 --backup-dir=/mnt/backups/neo4j --database=n* --pagecache=4G
```



**Tip:**

For a detailed example on how to back up and restore a database in a Causal cluster, see Back up and restore a database in Causal Cluster.

## 11.4. Prepare a database for restoring

### 11.4.1. Command

If the `--prepare-restore` option is disabled when you back up your database, your store may not contain the latest transactions pulled at backup time. In this case, you have to run the `--neo4j-admin prepare-restore` command to apply those transactions to the store, before you can restore your data.

### Syntax

```
neo4j-admin prepare-restore --target=<path>[,<path>...]...
                            [--verbose]
                            [--expand-commands]
                            [--parallel-recovery]
```

### Options

| Option | Default | Description |
|---|---|---|
| `--target` | | A path to the backup that is going to be prepared for restoring. A path can contain asterisks or question marks in the last subpath but must not contain commas. Multiple paths are separated by a comma. |
| `--verbose` | | Enable verbose output. |
| `--expand-commands` | | Allow command expansion in config value evaluation. |
| `--parallel-recovery` | `false` | Allow multiple threads to apply transactions to a backup in parallel. For some databases and workloads, this may reduce execution times significantly.<br><br>**Note:**<br>`parallel-recovery` is an experimental option. Consult Neo4j support before use. |

## 11.4.2. Example

The following is an example of preparing your database backup, created in the section Back up an online database, for restoring, using the `neo4j-admin prepare-restore` command.

```
bin/neo4j-admin prepare-restore --target=/mnt/backups/neo4j
```

# 11.5. Restore a database backup

## 11.5.1. Command

A database backup or an **offline** database can be restored using the `restore` command of `neo4j-admin`. 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 restore` must be invoked as the `neo4j` user to ensure the appropriate file permissions. For more information, see Administrative commands.

> **Note:**
>
> If the `--prepare-restore` option is disabled when backing up your database, you must first perform the `neo4j-admin prepare-restore` command before you can restore your database. This is to apply the latest transactions pulled at the backup time but not yet applied to the store. For more information, see Prepare a database for restoring.

## Syntax

```
neo4j-admin restore  --from=<path>[,<path>...]
                     [--verbose]
                     [--expand-commands]
                     [--database=<database>]
                     [--force]
                     [--move]
                     [--to-data-directory=<path>]
                     [--to-data-tx-directory=<path>]
```

## Options

| Option | Default | Description |
|---|---|---|
| `--from` | | A path or multiple paths to the database backup(s) for restore. A path can contain asterisks or question marks in the last subpath but must not contain commas. Commas are used to separate multiple paths. |
| `--verbose` | | Enables verbose output. |
| `--expand-commands` | | Allows command expansion in config value evaluation. |
| `--database` | `neo4j` | Name for the restored database. |
| `--force` | | Replaces an existing database. |
| `--move` | | Moves the backup files to the destination, rather than copying. This makes the restoring process faster and with no need for extra disk space. However, for this procedure to work properly, backup and database files must be on the same filesystem. In case they are not, the command will only copy the files and delete the backup, resulting in no performance benefits. |

| Option | Default | Description |
| --- | --- | --- |
| `--to-data-directory` | | Base directory for databases. Usage of this option is only allowed if the `--from` parameter points to one directory. |
| `--to-data-tx-directory` | | Base directory for transaction logs. Usage of this option is only allowed if the `--from` parameter points to one directory. |

## 11.5.2. Example

The following is an example of how to perform an online restore of the database backup created in the section Back up an online database, using the `neo4j-admin restore` command.

```
bin/neo4j-admin restore --from=/mnt/backups/neo4j --database=neo4j --force
```



Note:

Unless you are replacing an existing database, you must create the database (using `CREATE DATABASE` against the `system` database) after the restore operation finishes.

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/databases/databasename/tools/metadata_script.cypher*, which the `neo4j-admin restore` command outputs, using Cypher Shell:

### Using `cat` (UNIX)

```
cat data/databases/databasename/tools/metadata_script.cypher | bin/cypher-shell -u user -p password -a
ip_address:port -d system --param "database => 'databasename'"
```

### Using `type` (Windows)

```
type data\databases\databasename\tools\metadata_script.cypher | bin\cypher-shell.bat -u user -p password
-a ip_address:port -d system --param "database => 'databasename'"
```



Tip:

For a detailed example on how to back up and restore a database in a Causal cluster, see Back up and restore a database in Causal Cluster.



Note:

`neo4j-admin restore` cannot be applied to the Fabric virtual database. It must be run

directly on the databases that are part of the Fabric setup.

# 11.6. Back up an offline database



**Caution:**

Remember to plan your backup carefully and to back up each of your databases, including the `system` database.

## 11.6.1. Command

A Neo4j database can be backed up in offline mode using the `dump` command of `neo4j-admin`.

### Usage

The `neo4j-admin dump` command can be used for performing a full backup of an **offline** database. It dumps a database into a single-file archive, called *<database>.dump*. Alternatively, `neo4j-admin dump` can stream dump to standard output, enabling the output to be piped to another program, for example to `neo4j-admin load`.

The command can be run only locally from an online or an offline Neo4j DBMS.

It does not support SSL/TLS.

### Syntax

```
neo4j-admin dump --to=<destination-path>
                 [--verbose]
                 [--expand-commands]
                 [--database=<database>]
```

### Options

| Option | Default | Description |
|---|---|---|
| `--to` | | Destination (file or folder) of database dump, or `-` for standard output. |
| `--verbose` | | Enable verbose output. |
| `--expand-commands` | | Allow command expansion in config value evaluation. |
| `--database` | `neo4j` | Name of the database to dump. |

## 11.6.2. Example

The following is an example of how to create a dump of the default database `neo4j`, called *neo4j-<timestamp>.dump*, using the `neo4j-admin dump` command. The target directory */dumps/neo4j* must exist

before running the command and the database must be offline.

```
bin/neo4j-admin dump --database=neo4j --to=/dumps/neo4j/neo4j-<timestamp>.dump
```



Note:

`neo4j-admin dump` cannot be applied to the Fabric virtual database. It must be run directly on the databases that are part of the Fabric setup.

# 11.7. Restore a database dump

A database dump can be loaded to a Neo4j instance using the `load` command of `neo4j-admin`.

## 11.7.1. Command

The `neo4j-admin load` command loads a database from an archive created with the `neo4j-admin dump` command. Alternatively, `neo4j-admin load` can accept dump from standard input, enabling it to accept input from `neo4j-admin dump` or another source.

The command can be run from an online or an offline Neo4j DBMS.

If you are replacing an existing database, you have to shut it down before running the command. 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.

`neo4j-admin load` must be invoked as the `neo4j` user to ensure the appropriate file permissions.

### Syntax

```
neo4j-admin load --from=<archive-path>
                 [--verbose]
                 [--expand-commands]
                 [--database=<database>]
                 [--force]
                 [--info]
```

### Options

| Option | Default | Description |
|---|---|---|
| `--from` | | Path to archive created with the `neo4j-admin dump` command, or `-` to use standard input. |
| `--verbose` | | Enable verbose output. |
| `--expand-commands` | | Allow command expansion in config value evaluation. |
| `--database` | `neo4j` | Name for the loaded database. |

| Option | Default | Description |
| --- | --- | --- |
| `--force` | | Replace an existing database. |
| `--info` | | Print meta-data information about the archive file, such as, file count, byte count, and format of the load file. |

### 11.7.2. Example

The following is an example of how to load the dump of the `neo4j` database created in the section Back up an offline database, using the `neo4j-admin load` command. When replacing an existing database, you have to shut it down before running the command.

```
bin/neo4j-admin load --from=/dumps/neo4j/neo4j-<timestamp>.dump --database=neo4j --force
```



Note:

Unless you are replacing an existing database, you must create the database (using `CREATE DATABASE` against the `system` database) after the load operation finishes.



Note:

When using the `load` command to seed a Causal 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 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 from a database backup (online).



Note:

`neo4j-admin load` cannot be applied to the Fabric virtual database. It must be run directly on the databases that are part of the Fabric setup.

## 11.8. Copy a database store

A user database or backup can be copied to a Neo4j instance using the `copy` command of `neo4j-admin`.



Note:

`neo4j-admin copy` is not supported for use on the `system` database.

In Fabric deployments, `neo4j-admin copy` cannot be applied to the Fabric virtual database. It must be run directly on the databases that are part of the Fabric setup.

Warning:

It is important to note that `neo4j-admin copy` is an IOPS-intensive process. Using this process for upgrading or migration purposes can have significant performance implications, depending on your disc specification. It is therefore not appropriate for all use cases.

*Estimating the processing time*

Estimations for how long the `neo4j-admin copy` command will take can be made based upon the following:

- Neo4j, like many other databases, do 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 copy` on that 5000 IOPS disc is that it will take at least 1 hour to process a 144 GB database. [1]

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 will read/modify/write the store multiple times. Therefore, with an additional 144 GB of both read and write, the best-case scenario for running `neo4j-admin copy` on a 5000 IOPS disc is that it will take **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.

For detailed information about supported methods of upgrade and migration, see the Neo4j Upgrade and Migration Guide.

## 11.8.1. Command

`neo4j-admin copy` copies the data store of an existing **offline** database to a new database.

### Usage

The `neo4j-admin copy` command can be used to clean up database inconsistencies, compact stores, and do a direct migration from Neo4j 3.5 to any 4.x version. It can process an optional set of filters, which you can use to remove any unwanted data before copying the database. The command also reclaims the unused space of a database and creates a defragmented copy of that database or backup in the destination Neo4j instance.



Note:

`neo4j-admin copy` copies the data store without the schema (indexes and constraints). However, if the database has a schema defined, the command will output Cypher statements, which you can run to recreate the indexes and constraints.

**Tip:**

For a detailed example of how to reclaim unused space, see Reclaim unused space.
For a detailed example of how to back up a 3.5 database and use the `neo4j-admin copy` command to compact its store and migrate it to a 4.x Neo4j standalone instance, see Upgrade and Migration Guide → Tutorial: Back up and copy a database in a standalone instance.

**Warning:**

`neo4j-admin copy` preserves the node IDs; however, the relationships get new IDs.

## Syntax

```
neo4j-admin copy  [--verbose]
                  [--from-database=<database>]
                  [--from-path=<path>]
                  [--from-path-tx=<path>]
                   --to-database=<database>
                  [--neo4j-home-directory=<path>]
                  [--force]
                  [--compact-node-store]
                  [--to-format=<format>]
                  [--delete-nodes-with-labels=<label>[,<label>...]]
                  [--keep-only-node-properties=<label.property>[,<label.property>...]]
                  [--keep-only-nodes-with-labels=<label>[,<label>...]]
                  [--keep-only-relationship-
properties=<relationship.property>[,<relationship.property>...]]
                  [--skip-labels=<label>[,<label>...]]
                  [--skip-node-properties=<label.property>[,<label.property>...]]
                  [--skip-properties=<property>[,<property>...]]
                  [--skip-relationship-properties=<relationship.property>[,<relationship.property>...]]
                  [--skip-relationships=<relationship>[,<relationship>...]]
                  [--from-pagecache=<size>]
                  [--to-pagecache=<size>]
```

## Options

| Option | Description |
|---|---|
| `--verbose` | Enable verbose output. |
| `--from-database` | The database name to copy from. |
| `--from-path` | The path to the database to copy from.<br><br>It can be used to target databases outside of the installation, e.g., backups. |
| `--from-path-tx` | The path to the transaction log files. Use if the command cannot determine where they are located. |

| Option | Description |
| --- | --- |
| `--to-database` | The destination database name. |
| `--neo4j-home-directory=<path>` | Path to the home directory for the copied database.<br><br>Default: The same as the database copied from. |
| `--force` | Force the command to proceed even if the integrity of the database can not be verified. |
| `--compact-node-store` | Enforce node store compaction.<br><br>By default, the node store is not compacted on copy since it changes the node IDs. |
| `--to-format` | Set the format for the new database.<br><br>Valid values are `same`, `standard`, `high_limit`, and `aligned`. The `high_limit` format is only available in Enterprise Edition. If you go from `high_limit` to `standard`, there is no validation that the data will fit.<br><br>Default: The format of the source database. |
| `--delete-nodes-with-labels` | A comma-separated list of labels.<br><br>All nodes that have ANY of the specified labels will be deleted. Any node matching any of the labels will be ignored during copy. |
| `--keep-only-node-properties` | A list of property keys to keep for nodes with the specified label.<br><br>Any labels not explicitly mentioned will keep their properties. Cannot be combined with `--skip-properties` or `--skip-node-properties`. |
| `--keep-only-nodes-with-labels` | A list of labels.<br><br>All nodes that have any of the specified labels will be kept. Cannot be combined with `--delete-nodes-with-labels`. |
| `--keep-only-relationship-properties` | A list of property keys to keep for relationships with the specified type.<br><br>Any relationship types not explicitly mentioned will keep their properties.<br><br>Cannot be combined with `--skip-properties` or `--skip-relationship-properties`. |
| `--skip-labels` | A comma-separated list of labels to ignore during the copy. |
| `--skip-node-properties` | A list of property keys to ignore for nodes with the specified label.<br><br>Cannot be combined with `--skip-properties` or `--keep-only-node-properties`. |
| `--skip-properties` | A comma-separated list of property keys to ignore during the copy.<br><br>Cannot be combined with `--skip-node-properties`, `--keep-only-node-properties`, `--skip-relationship-properties`, and `--keep-only-relationship-properties`. |
| `--skip-relationships` | A comma-separated list of relationship types to ignore during the copy. |

| Option | Description |
| --- | --- |
| `--skip-relationship-properties` | A list of property keys to ignore for relationships with the specified type.<br><br>Cannot be combined with `--skip-properties` or `--keep-only-relationship-properties`. |
| `--from-pagecache` | The size of the page cache to use for reading. |
| `--to-pagecache` | The size of the page cache to use for writing. |



**Tip:**

You can use the `--from-pagecache` and `--to-pagecache` options to speed up the copy operation by specifying how much cache to allocate when reading the source and writing the destination. As a rule of thumb, `--to-pagecache` should be around `1-2GB` since it mostly does sequential writes. The `--from-pagecache` should then be assigned whatever memory you can spare since Neo4j does random reads from the source.

## 11.8.2. Examples

*Example 71. Use `neo4j-admin copy` to copy the data store of the database `neo4j`.*

1. Stop the database named `neo4j`:

```
STOP DATABASE neo4j
```

2. Copy the data store from `neo4j` to a new database called `my-database`:

```
bin/neo4j-admin copy --from-database=neo4j --to-database=my-database
```

3. Run the following command to verify that the database has been successfully copied.

```
ls -al ../data/databases
```



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 my-database
```

5. Verify that the copied database is online.

```
SHOW DATABASES
```

6. If your original database has a schema defined, change your active database to the copied database and recreate the schema using the `neo4j-admin copy` output.



The console output is saved to *logs/neo4j-admin-copy-<timestamp>.log*.

*Example 72. Use `neo4j-admin copy` to filter the data you want to copy.*

The command can perform some basic forms of processing. You can filter the data that you want to copy by removing nodes, labels, properties, and relationships.

```
bin/neo4j-admin copy --from-database=neo4j --to-database=my-database --delete-nodes-with-labels
="Cat,Dog"
```

The command creates a copy of the database `neo4j` but without the nodes with the labels `:Cat` and `:Dog`.



Labels are processed independently, i.e., the filter deletes any node with a label `:Cat`, `:Dog`, or both.



**Note:**

For a detailed example of how to use `neo4j-admin copy` to filter out data for a Fabric installation, see Sharding data with the `copy` command.

[1] 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`.

# Chapter 12. Authentication and authorization

This section helps you ensure that your Neo4j deployment adheres to your company's information security guidelines by setting up the appropriate authentication and authorization rules.

The following topics are covered:

- Introduction
- Built-in roles
- Fine-grained access control
- Integration with LDAP directory services
- Integration with Single Sign-On (SSO) services
- Manage procedure and user-defined function permissions
- Terminology



The functionality described in this section is applicable to Enterprise Edition. A limited set of user management functions are also available in Community Edition. Native roles overview gives a quick overview of these.

## 12.1. Introduction

This page provides an overview of authentication and authorization in Neo4j. Authorization is managed using role-based access control (*RBAC*). Permissions that define access control are assigned to roles, which are in turn assigned to users.

Neo4j has the following auth providers, that can perform user authentication and authorization:

***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, for example, to recover an `admin` user password or assign a user to the `admin` role, make sure you block all network connections during the recovery phase so users can connect to Neo4j only via `localhost`. 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*.

The Cypher commands to manage users, roles, and permissions are described in detail in Cypher Manual → Access control. Various scenarios that illustrate the use of the native auth provider are available in Fine-grained access control.

*LDAP auth provider*

Another way of controlling authentication and authorization is 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*

Some clients wish to centrally manage authentication and authorization for their systems in an identity provider which provides single sign-on for their other systems. Neo4j supports the popular OpenID Connect mechanism for integrating with identity providers and configuration for that is described in Integration with Single Sign-On Services.

*Custom-built plugin auth providers*

For clients with specific requirements not satisfied with either native or LDAP, Neo4j provides 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. The plugin is described in 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.

## 12.2. Built-in roles

Neo4j provides built-in roles with default privileges. The built-in roles and the default privileges are:

`PUBLIC`

- Access to the home database.
- Allows executing procedures with the users own privileges.
- Allows executing user-defined functions with the users own privileges.

`reader`

- Access to all databases.
- Traverse and read on the data graph (all nodes, relationships, properties).

`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.

`publisher`

- Access to all databases.
- Traverse, read, and write on the data graph.

`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.
- Create/drop/show indexes and constraints along with any other future schema constructs.
- Allows executing procedures with the users own privileges or boosted privileges.
- Allows executing admin procedures.
- Allows executing user-defined functions with the users own privileges or boosted privileges.
- View/terminate queries.
- Manage databases, users, roles, and privileges.

All users will be assigned the `PUBLIC` role, which by default does not give any rights or capabilities regarding the data, not even read privileges. A user may have more than one assigned role, and the union of these determine what action(s) on the data may be undertaken by the user. For instance, a user assigned to the `reader` role will be able to execute procedures because all users are also assigned to the `PUBLIC` role, which enables that capability.

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.
- The user will no longer be able to log back in (until re-activated by an administrator if suspended).
- There is no need to remove assigned roles from a user prior to deleting the user.



Deleting a user will not automatically terminate associated connections, sessions, transactions, or queries.

The set of actions on the data and database prescribed by each role are described below. The subset of

the functionality which is available with Community Edition is also included:

*Table 446. Native roles overview*

| Action | `reader` | `editor` | `publisher` | `architect` | `admin` | `PUBLIC` | Available in Community Edition |
|---|---|---|---|---|---|---|---|
| Change own password | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| View own details | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Read data | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ |
| Execute procedures | | | | | ✓ | ✓ | ✓ |
| Execute functions | | | | | ✓ | ✓ | ✓ |
| Execute admin procedures | | | | | ✓ | | ✓ |
| View own queries | ✓ | ✓ | ✓ | ✓ | ✓ | | |
| Terminate own queries | ✓ | ✓ | ✓ | ✓ | ✓ | | |
| Write/update/delete existing data | | ✓ | ✓ | ✓ | ✓ | | ✓ |
| Create new types of properties key | | | ✓ | ✓ | ✓ | | ✓ |
| Create new types of nodes labels | | | ✓ | ✓ | ✓ | | ✓ |
| Create new types of relationship types | | | ✓ | ✓ | ✓ | | ✓ |
| Create/drop/show index/constraint | | | | ✓ | ✓ | | ✓ |
| Create/delete user | | | | | ✓ | | ✓ |
| Change another user's name | | | | | ✓ | | ✓ |
| Change another user's password | | | | | ✓ | | ✓ |

| Action | `reader` | `editor` | `publisher` | `architect` | `admin` | `PUBLIC` | Available in Community Edition |
|---|---|---|---|---|---|---|---|
| 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 | | | | | ✔ | | |
| Access home database | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| Access all databases | ✔ | ✔ | ✔ | ✔ | ✔ | | ✔ |
| View all users | | | | | ✔ | | ✔ |
| View all roles | | | | | ✔ | | |
| View all roles for a user | | | | | ✔ | | |
| View all users for a role | | | | | ✔ | | |
| View all queries | | | | | ✔ | | |
| View all databases | | | | | ✔ | | |
| View own privileges | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | |

| Action | `reader` | `editor` | `publisher` | `architect` | `admin` | `PUBLIC` | Available in Community Edition |
|---|---|---|---|---|---|---|---|
| View another user's privileges | | | | | ✔ | | |
| Grant/deny/revoke privileges | | | | | ✔ | | |
| Terminate all queries | | | | | ✔ | | |
| Dynamically change configuration (see Dynamic settings) | | | | | ✔ | | |

More information about the built-in roles and their privileges can be found in Neo4j Cypher Manual.

## 12.3. 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.

### 12.3.1. 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
```

> **Note:**
>
> It is recommended to block network connections during the recovery phase, so users can connect to Neo4j only via `localhost`. This can be achieved by either:
>
> - Temporarily commenting out the `dbms.default_listen_address` parameter:

```
#dbms.default_listen_address=<your_configuration>
```

or

- Providing the specific localhost value:

```
dbms.default_listen_address=127.0.0.1
```

3. Start Neo4j:

```
$ bin/neo4j start
```

1. Stop the cluster (all Core servers and Read Replicas).

```
$ bin/neo4j stop
```

2. On each Core server, 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`. This ensures that no one from outside can access the cluster during the recovery period.

```
#dbms.connector.http.enabled=true
#dbms.connector.https.enabled=true
dbms.connector.bolt.listen_address:127.0.0.1
```

3. Start all Core servers:

```
$ bin/neo4j start
```

## 12.3.2. 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 Core servers.

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:**
>
> Cluster If you have specified a non-default port for your `bolt` connector, add `-a neo4j://<your-core>:<non-default-bolt-port>` to the `cypher-shell` command to be able to connect to your Core server.

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.

### 12.3.3. 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 Core servers.

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:**
>
> Cluster If you have specified a non-default port for your `bolt` connector, add `-a neo4j://<your-core>:<non-default-bolt-port>` to the `cypher-shell` command to be able to connect to your Core server.

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.

## 12.3.4. 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 Core servers.

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:**
>
> Cluster If you have specified a non-default port for your `bolt` connector, add `-a neo4j://<your-core>:<non-default-bolt-port>` to the `cypher-shell` command to be able to connect to your Core server.

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, we suggest granting 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.

### 12.3.5. 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 Core servers.

```
$ bin/neo4j stop
```

2. Enable the authentication and restore each Core server to its original configuration (See Disable authentication).
3. Start the cluster (all Core servers and Read Replicas):

```
$ bin/neo4j start
```

## 12.4. Fine-grained access control

When creating a database, administrators may want to establish which users have the ability to access certain information.

As described in Built-in roles, 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 page contains an example that illustrates various aspects of security and fine-grained access control.

### 12.4.1. 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

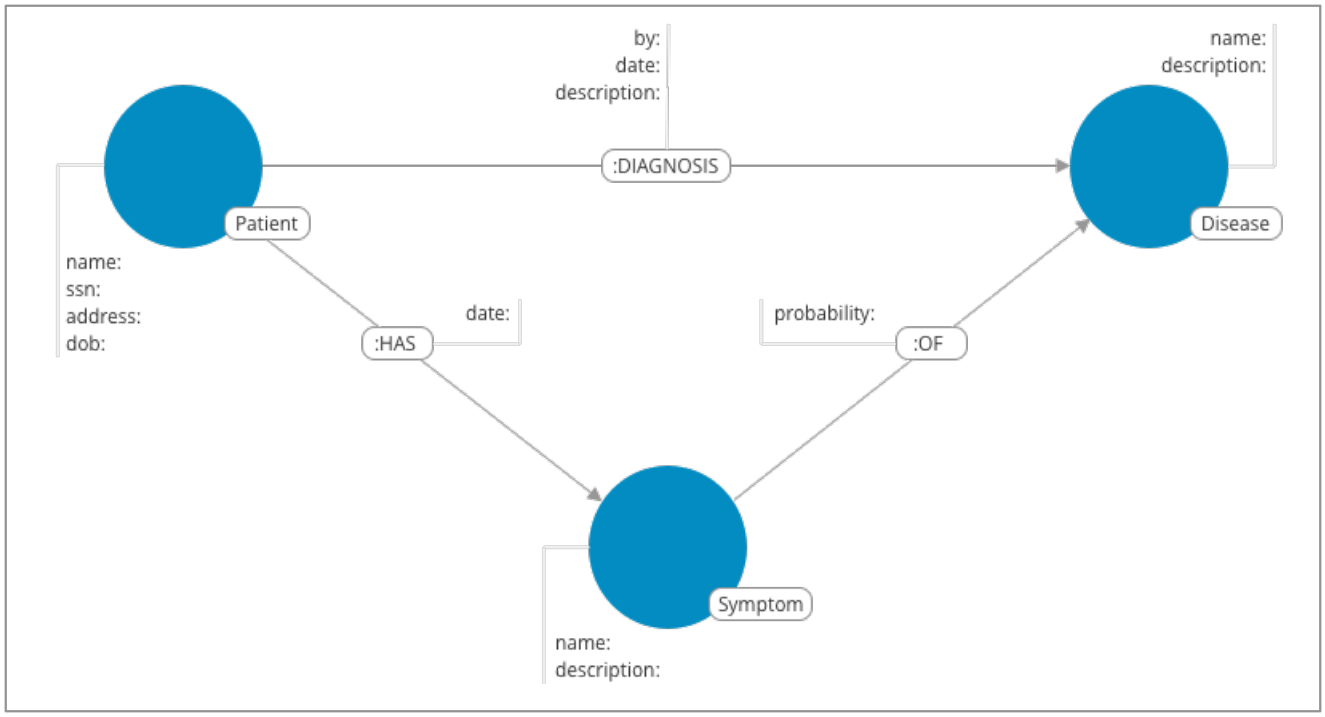



*Figure 14. Healthcare use case*

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.

## 12.4.2. Security

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 Cypher Manual → Access control.

The following examples show two different approaches to using Neo4j security features to support the *healthcare* database application. The first approach uses Built-in roles, whereas the second uses more advanced resources with fine-grained privileges for sub-graph access control.

In this example, consider five users of the *healthcare* database:

- Alice, the doctor.
- Daniel, the nurse.

- Bob, the receptionist.
- Charlie, the researcher.
- Tina, the IT administrator.

These users can be created 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 have no ability to 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 the built-in roles, or through more fine-grained access control using privileges and custom roles.

## 12.4.3. 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, can by default access the home database, 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 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:

```
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;
```

## 12.4.4. 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:

```
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:

```
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;
```

```
+-------------------------------------------------------------------------+
| command                                                                 |
+-------------------------------------------------------------------------+
| "GRANT ACCESS ON DATABASE * TO `itadmin`"                               |
| "GRANT MATCH {*} ON GRAPH * NODE * TO `itadmin`"                        |
| "GRANT MATCH {*} ON GRAPH * RELATIONSHIP * TO `itadmin`"                |
| "GRANT WRITE ON GRAPH * TO `itadmin`"                                   |
| "GRANT INDEX MANAGEMENT ON DATABASE * TO `itadmin`"                     |
| "GRANT CONSTRAINT MANAGEMENT ON DATABASE * TO `itadmin`"                |
| "GRANT NAME MANAGEMENT ON DATABASE * TO `itadmin`"                      |
| "GRANT START ON DATABASE * TO `itadmin`"                                |
| "GRANT STOP ON DATABASE * TO `itadmin`"                                 |
| "GRANT TRANSACTION MANAGEMENT (*) ON DATABASE * TO `itadmin`"           |
| "GRANT ALL DBMS PRIVILEGES ON DBMS TO `itadmin`"                        |
| "DENY READ {ssn} ON GRAPH `healthcare` NODE Patient TO `itadmin`"       |
| "DENY CREATE ON GRAPH `healthcare` RELATIONSHIP DIAGNOSIS TO `itadmin`" |
| "DENY ROLE MANAGEMENT ON DBMS TO `itadmin`"                             |
| "DENY PRIVILEGE MANAGEMENT ON DBMS TO `itadmin`"                        |
+-------------------------------------------------------------------------+
```

> **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`:

```
neo4j@system> GRANT ROLE itadmin TO tina;
```

To demonstrate that Tina is not able to see the patients' `SSN`, you can login to `healthcare` as `tina` and run the following query:

```
MATCH (n:Patient)
 WHERE n.dateOfBirth < date('1972-06-12')
RETURN n.name, n.ssn, n.address, n.dateOfBirth;
```

```
+-----------------------------------------------------------------------+
| n.name          | n.ssn | n.address                | n.dateOfBirth |
+-----------------------------------------------------------------------+
| "Mary Stone"    | NULL  | "1 secret way, downtown" | 1970-01-15    |
| "Ally Anderson" | NULL  | "1 secret way, downtown" | 1970-08-20    |
| "Sally Stone"   | NULL  | "1 secret way, downtown" | 1970-03-12    |
| "Jane Stone"    | NULL  | "1 secret way, downtown" | 1970-07-21    |
| "Ally Svensson" | NULL  | "1 secret way, downtown" | 1971-08-15    |
| "Jane Svensson" | NULL  | "1 secret way, downtown" | 1972-05-12    |
| "Ally Svensson" | NULL  | "1 secret way, downtown" | 1971-07-30    |
+-----------------------------------------------------------------------+
```

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);
```

```
Create relationship with type 'DIAGNOSIS' 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:

```
// 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:

```
// 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:

```
neo4j@system> SHOW USER charlie PRIVILEGES AS COMMANDS;
```

```
+------------------------------------------------------------------------+
| command                                                                |
+------------------------------------------------------------------------+
| "GRANT ACCESS ON HOME DATABASE TO $role"                               |
| "GRANT ACCESS ON DATABASE `healthcare` TO $role"                       |
| "GRANT EXECUTE PROCEDURE * ON DBMS TO $role"                           |
| "GRANT EXECUTE FUNCTION * ON DBMS TO $role"                            |
| "GRANT MATCH {*} ON GRAPH `healthcare` NODE * TO $role"                |
| "GRANT MATCH {*} ON GRAPH `healthcare` RELATIONSHIP * TO $role"        |
| "DENY TRAVERSE ON GRAPH `healthcare` RELATIONSHIP DIAGNOSIS TO $role" |
| "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"          |
+------------------------------------------------------------------------+
```

Now when Charlie logs into the `healthcare` database and tries to run a command 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;
```

```
+-------------------------------------------+
| n.name | n.ssn | n.address | n.dateOfBirth |
+-------------------------------------------+
| NULL   | NULL  | NULL      | 1971-05-31    |
| NULL   | NULL  | NULL      | 1971-04-17    |
| NULL   | NULL  | NULL      | 1971-12-27    |
| NULL   | NULL  | NULL      | 1970-02-13    |
| NULL   | NULL  | NULL      | 1971-02-04    |
| NULL   | NULL  | NULL      | 1971-05-10    |
| NULL   | NULL  | NULL      | 1971-02-21    |
+-------------------------------------------+
```

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;
```

```
+-----------------------------------------+
| disease               | score           |
+-----------------------------------------+
| "Acute Argitis"       | 95.05395287286318 |
| "Chronic Someitis"    | 88.7220337139605  |
| "Chronic Placeboitis" | 88.43609533058974 |
| "Acute Whatitis"      | 83.23493746472457 |
| "Acute Otheritis"     | 82.46129768949129 |
| "Chronic Otheritis"   | 82.03650063794025 |
| "Acute Placeboitis"   | 77.34207326583929 |
| "Acute Yellowitis"    | 76.34519967465832 |
| "Chronic Whatitis"    | 73.73968070128234 |
| "Chronic Yellowitis"  | 71.58791287376775 |
+-----------------------------------------+
```

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:

```
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 them this new role:

```
neo4j@system> GRANT ROLE doctor TO alice;
```

To demonstrate that Alice is not able to see patient addresses, log in as `alice` to `healthcare` and run the following query:

```
MATCH (n:Patient)
 WHERE n.dateOfBirth < date('1972-06-12')
RETURN n.name, n.ssn, n.address, n.dateOfBirth;
```

```
+-----------------------------------------------------+
| n.name          | n.ssn   | n.address | n.dateOfBirth |
+-----------------------------------------------------+
| "Jack Anderson" | 1234647 | NULL      | 1970-07-23    |
| "Joe Svensson"  | 1234659 | NULL      | 1972-06-07    |
| "Mary Jackson"  | 1234568 | NULL      | 1971-10-19    |
| "Jack Jackson"  | 1234583 | NULL      | 1971-05-04    |
| "Ally Smith"    | 1234590 | NULL      | 1971-12-07    |
| "Ally Stone"    | 1234606 | NULL      | 1970-03-29    |
| "Mark Smith"    | 1234610 | NULL      | 1971-03-30    |
```

```
+-----------------------------------------------------+
```

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(*);
```

```
+------------------------+
| labels      | count(*) |
+------------------------+
| ["Patient"] | 101      |
| ["Symptom"] | 10       |
| ["Disease"] | 12       |
+------------------------+
```

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.

```
+-------------------------------------------------+
| n.name           | d.name                | score |
+-------------------------------------------------+
| "Sally Anderson" | "Chronic Otheritis"   | 4     |
| "Sally Anderson" | "Chronic Yellowitis"  | 3     |
| "Sally Anderson" | "Chronic Placeboitis" | 3     |
| "Sally Anderson" | "Acute Whatitis"      | 2     |
| "Sally Anderson" | "Acute Yellowitis"    | 2     |
| "Sally Anderson" | "Chronic Someitis"    | 2     |
| "Sally Anderson" | "Chronic Argitis"     | 2     |
| "Sally Anderson" | "Chronic Whatitis"    | 2     |
| "Sally Anderson" | "Acute Someitis"      | 1     |
| "Sally Anderson" | "Acute Argitis"       | 1     |
| "Sally Anderson" | "Acute Otheritis"     | 1     |
+-------------------------------------------------+
```

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:

```
+-------------------------------------------------------------------------------------+
| p.name          | d.name                | i.by    | i.date     | updated              |
+-------------------------------------------------------------------------------------+
| "Sally Anderson" | "Chronic Placeboitis" | "Alice" | 2020-05-29 | P0M0DT213.076000000S |
+-------------------------------------------------------------------------------------+
```

> **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:

```
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:

```
neo4j@system> GRANT ROLE receptionist TO bob;
```

With these privileges, if Bob tries to read the entire database, they will still only see the patients:

```
MATCH (n) WITH labels(n) AS labels
RETURN labels, count(*);
```

```
+------------------------+
| labels      | count(*) |
+------------------------+
| ["Patient"] | 101      |
+------------------------+
```

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;
```

```
+-------------------------------------------------------------------+
| n.name          | n.ssn   | n.address               | n.dateOfBirth |
+-------------------------------------------------------------------+
| "Mark Stone"    | 1234666 | "1 secret way, downtown" | 1970-08-04    |
| "Sally Jackson" | 1234633 | "1 secret way, downtown" | 1970-10-21    |
| "Bob Stone"     | 1234581 | "1 secret way, downtown" | 1972-02-16    |
| "Ally Anderson" | 1234582 | "1 secret way, downtown" | 1970-05-13    |
| "Mark Svensson" | 1234594 | "1 secret way, downtown" | 1970-01-16    |
| "Bob Anderson"  | 1234597 | "1 secret way, downtown" | 1970-09-23    |
| "Jack Svensson" | 1234599 | "1 secret way, downtown" | 1971-02-13    |
| "Mark Jackson"  | 1234618 | "1 secret way, downtown" | 1970-03-28    |
| "Jack Jackson"  | 1234623 | "1 secret way, downtown" | 1971-04-02    |
+-------------------------------------------------------------------+
```

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;
```

```
+-----------------------------------+
| n.name            | n.dateOfBirth |
+-----------------------------------+
| "Another Patient" | 2001-01-20    |
+-----------------------------------+
```

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;
```

```
+-------------------------------------------------------------+
| n.name            | n.dateOfBirth | n.address               |
+-------------------------------------------------------------+
| "Another Patient" | 2001-01-20    | "2 streets down, uptown" |
+-------------------------------------------------------------+
```

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;
```

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;
```

```
org.neo4j.graphdb.ConstraintViolationException: Cannot delete node<42>, 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, they do not have sufficient traverse rights to find nor 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:

```
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:

```
neo4j@system> GRANT ROLE doctor, receptionist TO daniel;
```

Daniel should now have a combined set of privileges:

```
SHOW USER daniel PRIVILEGES AS COMMANDS;
```

```
+-------------------------------------------------------------------------+
| command                                                                 |
+-------------------------------------------------------------------------+
| "GRANT ACCESS ON HOME DATABASE TO $role"                                |
| "GRANT ACCESS ON DATABASE `healthcare` TO $role"                        |
| "GRANT EXECUTE PROCEDURE * ON DBMS TO $role"                            |
| "GRANT EXECUTE FUNCTION * ON DBMS TO $role"                             |
| "GRANT TRAVERSE ON GRAPH `healthcare` NODE * TO $role"                  |
| "GRANT TRAVERSE ON GRAPH `healthcare` RELATIONSHIP * TO $role"          |
| "GRANT READ {*} ON GRAPH `healthcare` NODE * TO $role"                  |
| "GRANT READ {*} ON GRAPH `healthcare` RELATIONSHIP * TO $role"          |
| "GRANT MATCH {*} ON GRAPH `healthcare` NODE Patient TO $role"           |
| "GRANT WRITE ON GRAPH `healthcare` TO $role"                            |
| "GRANT SET PROPERTY {*} ON GRAPH `healthcare` NODE Patient TO $role"    |
| "GRANT CREATE ON GRAPH `healthcare` NODE Patient TO $role"              |
| "GRANT DELETE ON GRAPH `healthcare` NODE Patient TO $role"              |
| "DENY READ {address} ON GRAPH `healthcare` NODE Patient TO $role"       |
| "DENY SET PROPERTY {address} ON GRAPH `healthcare` NODE Patient TO $role" |
```

```
+--------------------------------------------------------------------+
```

> **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. To do so, the nurse can run the following query:

```
MATCH (n:Patient)
 WHERE n.dateOfBirth < date('1972-06-12')
RETURN n.name, n.ssn, n.address, n.dateOfBirth;
```

Which returns these results:

```
+-----------------------------------------------------+
| n.name          | n.ssn   | n.address | n.dateOfBirth |
+-----------------------------------------------------+
| "Jane Anderson" | 1234572 | NULL      | 1971-05-26    |
| "Mark Stone"    | 1234586 | NULL      | 1972-06-07    |
| "Joe Smith"     | 1234595 | NULL      | 1970-12-28    |
| "Joe Jackson"   | 1234603 | NULL      | 1970-08-31    |
| "Jane Jackson"  | 1234628 | NULL      | 1972-01-31    |
| "Mary Anderson" | 1234632 | NULL      | 1971-01-07    |
| "Jack Svensson" | 1234639 | NULL      | 1970-01-06    |
+-----------------------------------------------------+
```

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:

```
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 takes another look at the patient records, 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;
```

```
+-----------------------------------------------------------------------+
| n.name          | n.ssn   | n.address                | n.dateOfBirth |
+-----------------------------------------------------------------------+
| "Jane Anderson" | 1234572 | "1 secret way, downtown" | 1971-05-26    |
| "Mark Stone"    | 1234586 | "1 secret way, downtown" | 1972-06-07    |
| "Joe Smith"     | 1234595 | "1 secret way, downtown" | 1970-12-28    |
| "Joe Jackson"   | 1234603 | "1 secret way, downtown" | 1970-08-31    |
| "Jane Jackson"  | 1234628 | "1 secret way, downtown" | 1972-01-31    |
| "Mary Anderson" | 1234632 | "1 secret way, downtown" | 1971-01-07    |
| "Jack Svensson" | 1234639 | "1 secret way, downtown" | 1970-01-06    |
+-----------------------------------------------------------------------+
```

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;
```

```
+-------------------------------------------------------------------+
| p.name           | d.name                | i.by     | i.date     |
+-------------------------------------------------------------------+
| "Sally Anderson" | "Chronic Placeboitis" | "Daniel" | 2020-05-29 |
+-------------------------------------------------------------------+
```

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:

```
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`:

```
neo4j@system> SHOW USER daniel PRIVILEGES AS COMMANDS;
```

```
+----------------------------------------------------------------------+
| command                                                              |
+----------------------------------------------------------------------+
| "GRANT ACCESS ON HOME DATABASE TO $role"                             |
| "GRANT ACCESS ON DATABASE `healthcare` TO $role"                     |
| "GRANT EXECUTE PROCEDURE * ON DBMS TO $role"                         |
| "GRANT EXECUTE FUNCTION * ON DBMS TO $role"                          |
| "GRANT TRAVERSE ON GRAPH `healthcare` NODE * TO $role"               |
| "GRANT TRAVERSE ON GRAPH `healthcare` RELATIONSHIP * TO $role"       |
| "GRANT READ {*} ON GRAPH `healthcare` NODE * TO $role"               |
| "GRANT READ {*} ON GRAPH `healthcare` RELATIONSHIP * TO $role"       |
| "GRANT WRITE ON GRAPH `healthcare` TO $role"                         |
| "DENY CREATE ON GRAPH `healthcare` RELATIONSHIP DIAGNOSIS TO $role" |
+----------------------------------------------------------------------+
```

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;
```

```
+-----------------------------------------------------+
| n.ssn   | n.name       | d.name                | score |
+-----------------------------------------------------+
| 1234650 | "Mark Smith" | "Chronic Whatitis"    | 3     |
| 1234650 | "Mark Smith" | "Chronic Someitis"    | 3     |
| 1234650 | "Mark Smith" | "Acute Someitis"      | 2     |
| 1234650 | "Mark Smith" | "Chronic Otheritis"   | 2     |
| 1234650 | "Mark Smith" | "Chronic Yellowitis"  | 2     |
| 1234650 | "Mark Smith" | "Chronic Placeboitis" | 2     |
| 1234650 | "Mark Smith" | "Acute Otheritis"     | 2     |
| 1234650 | "Mark Smith" | "Chronic Argitis"     | 2     |
| 1234650 | "Mark Smith" | "Acute Placeboitis"   | 2     |
| 1234650 | "Mark Smith" | "Acute Yellowitis"    | 1     |
| 1234650 | "Mark Smith" | "Acute Argitis"       | 1     |
| 1234650 | "Mark Smith" | "Acute Whatitis"      | 1     |
+-----------------------------------------------------+
```

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;
```

```
Create relationship with type 'DIAGNOSIS' 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:

```
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:

```
neo4j@system> SHOW USER tina PRIVILEGES AS COMMANDS;
```

```
+--------------------------------------------------+
| command                                          |
+--------------------------------------------------+
| "GRANT ACCESS ON HOME DATABASE TO $role"         |
| "GRANT ACCESS ON DATABASE `healthcare` TO $role" |
| "GRANT EXECUTE PROCEDURE * ON DBMS TO $role"     |
| "GRANT EXECUTE FUNCTION * ON DBMS TO $role"      |
| "GRANT ROLE MANAGEMENT ON DBMS TO $role"         |
| "GRANT USER MANAGEMENT ON DBMS TO $role"         |
| "GRANT SHOW PRIVILEGE ON DBMS TO $role"          |
+--------------------------------------------------+
```



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 Cypher

Manual → 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:

```
CREATE USER sally SET PASSWORD 'secretpassword' CHANGE REQUIRED;
GRANT ROLE receptionist TO sally;
SHOW USER sally PRIVILEGES AS COMMANDS;
```

```
+-----------------------------------------------------------------------+
| command                                                               |
+-----------------------------------------------------------------------+
| "GRANT ACCESS ON HOME DATABASE TO $role"                              |
| "GRANT ACCESS ON DATABASE `healthcare` TO $role"                      |
| "GRANT EXECUTE PROCEDURE * ON DBMS TO $role"                          |
| "GRANT EXECUTE FUNCTION * ON DBMS TO $role"                           |
| "GRANT MATCH {*} ON GRAPH `healthcare` NODE Patient TO $role"         |
| "GRANT SET PROPERTY {*} ON GRAPH `healthcare` NODE Patient TO $role" |
| "GRANT CREATE ON GRAPH `healthcare` NODE Patient TO $role"            |
| "GRANT DELETE ON GRAPH `healthcare` NODE Patient TO $role"            |
+-----------------------------------------------------------------------+
```

# 12.5. Integration with LDAP directory services

This page describes Neo4j support for integrating with LDAP systems. The following topics are covered:

- Introduction
- LDAP configuration parameters
- Set Neo4j to use LDAP
- Map the LDAP groups to the Neo4j roles
- Configure Neo4j to use Active Directory
  - Configure Neo4j to support LDAP user ID authentication
  - Configure Neo4j to support attribute authentication
  - Configure Neo4j to support `sAMAccountName` authentication by setting `user_dn_template`
- Configure Neo4j to use OpenLDAP
- Verify the LDAP configuration
- The auth cache
- Available methods of encryption
  - Use LDAP with encryption via StartTLS
  - Use LDAP with encrypted LDAPS
- Use a self-signed certificate (SSL) in a test environment

## 12.5.1. Introduction

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.

## 12.5.2. LDAP dynamic configuration settings

The following configuration settings can be updated while the database is running, see 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 on a user object that contains groups to be used for mapping to roles. Common values: `memberOf` and `gidNumber`. |
| 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. |

All settings are defined at server startup time in the default configuration file *neo4j.conf* or can be modified at runtime using `dbms.setConfigValue()`.

## 12.5.3. 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 73. Configure Neo4j to use LDAP and the native authentication and authorization provider*

   ```
   dbms.security.authentication_providers=ldap,native
   dbms.security.authorization_providers=ldap,native
   ```

## 12.5.4. Map the LDAP groups to the Neo4j roles

To access the user and role management procedures, 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 74. 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 the Cypher Manual → Creating roles.

## 12.5.5. 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=mypassword
```

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
```



Earlier Neo4j versions only supported `samaccountname` as a search attribute. This could be configured with `dbms.security.ldap.authentication.use_samaccountname`. That setting has been deprecated and replaced by `dbms.security.ldap.authentication.search_for_attribute`.

## 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.

## 12.5.6. 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.

## 12.5.7. 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 mypassword -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
```

## 12.5.8. 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 447. 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.<br><br>Whether authentication caching should be enabled or not must be considered in view of your company's security guidelines. |

| Parameter name | Default value | Description |
| --- | --- | --- |
| dbms.security.auth_cache_ttl | `600 seconds` | Is the time to live (TTL) for cached authentication and authorization info.<br><br>Setting the TTL to 0 disables all auth caching.<br><br>A short TTL requires more frequent re-authentication and re-authorization, which can impact performance.<br><br>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.<br><br>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()
```

## 12.5.9. 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
```

### 12.5.10. 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 `dbms.jvm.additional` in *neo4j.conf*. The path to the certificate file `MyCert.jks` is an absolute path to the Neo4j server.

```
dbms.jvm.additional=-Djavax.net.ssl.keyStore=/path/to/MyCert.jks
dbms.jvm.additional=-Djavax.net.ssl.keyStorePassword=mypasword
dbms.jvm.additional=-Djavax.net.ssl.trustStore=/path/to/MyCert.jks
dbms.jvm.additional=-Djavax.net.ssl.trustStorePassword=mypasword
```

## 12.6. Integration with Single Sign-On Services

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.



Note:

Currently it is not possible to login to Cypher Shell using SSO authentication and authorization.

### 12.6.1. 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. |
| dbms.security.oidc.<provider>.client_id | | true | The `client_id` of this client as issued by the provider. |
| 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. |

| Parameter name | Default value | Dynamic | Description |
| --- | --- | --- | --- |
| 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. |
| dbms.security.oidc.<provider>.claims.groups | | true | The claim to use for the database roles. |
| 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`). |

## 12.6.2. 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 75. 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
   ```

## 12.6.3. 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 76. 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 the Cypher Manual → Creating 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.

## 12.6.4. 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). In the configuration examples below, we are using `mysso` as our provider 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 users can identify the provider they 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 a 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
   ```

   In some situations there may be multiple values for the `aud` claim. In this situation, the id_token should contain an authorized party(`azp`) claim containing the client id, which is configured as follows:

   ```
   dbms.security.oidc.mysso.claims.client_id=myclientid
   ```

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 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.

## 12.6.5. 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 `dbms.jvm.additional` in *neo4j.conf*. The path to the certificate file `MyCert.jks` is an absolute path to the Neo4j server.

```
dbms.jvm.additional=-Djavax.net.ssl.keyStore=/path/to/MyCert.jks
dbms.jvm.additional=-Djavax.net.ssl.keyStorePassword=mypasword
dbms.jvm.additional=-Djavax.net.ssl.trustStore=/path/to/MyCert.jks
dbms.jvm.additional=-Djavax.net.ssl.trustStorePassword=mypasword
```

## 12.6.6. 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`.



**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.

# 12.7. Manage procedure and user-defined function permissions

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 on creating and using user-defined procedures and functions.

## 12.7.1. Manage procedure permissions

Procedure permissions can be managed using the native execute privileges. These control whether the user is allowed to both 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 77. Grant privilege to execute 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 have 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 that 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.

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 78. Grant privilege to execute procedure with elevated privileges*

```
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 that user will see all nodes and relationships that exist in the graph, even though they have no traversal privileges.

## 12.7.2. 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 79. Grant privilege to execute 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 have 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.

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 80. Grant privilege to execute user-defined function with elevated privileges*

```
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 have 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.

## 12.7.3. Manage procedure and user-defined function permissions from config setting

It is possible to grant boosting for procedures and user-defined functions through config settings. These settings will be translated to temporary `execute boosted procedure` and `execute boosted function` privileges that cannot be revoked.

### `dbms.security.procedures.default_allowed`

The setting `dbms.security.procedures.default_allowed` defines a single role that is allowed to execute any procedure or user-defined function that is not matched by the `dbms.security.procedures.roles` configuration.

*Example 81. Configure a default role that can execute procedures and user-defined functions*

Assume that we have the following configuration:

```
dbms.security.procedures.default_allowed=superAdmin
```

This will create the following temporary privileges:

- `GRANT EXECUTE BOOSTED PROCEDURE * ON DBMS TO superAdmin`
- `GRANT EXECUTE BOOSTED USER DEFINED FUNCTION * ON DBMS TO superAdmin`
- If the setting dbms.security.procedures.roles has some roles to name defined, then for any procedure/function not also granted to the superAdmin role, create temporary privileges:
  - `DENY EXECUTE BOOSTED PROCEDURE name ON DBMS TO superAdmin`
  - `DENY EXECUTE BOOSTED USER DEFINED FUNCTION name ON DBMS TO superAdmin`

### `dbms.security.procedures.roles`

The `dbms.security.procedures.roles` setting provides fine-grained control over procedures and user-defined functions.

*Example 82. Configure roles for the execution of specific procedures and user-defined functions*

Assume that we have the following configuration:

```
dbms.security.procedures.default_allowed=superAdmin
dbms.security.procedures.roles=apoc.coll.*:Collector;apoc.trigger.add:TriggerHappy,superAdmin
```

This will have create the following temporary privileges:

- `GRANT EXECUTE BOOSTED PROCEDURE apoc.coll.* ON DBMS TO Collector`
- `GRANT EXECUTE BOOSTED USER DEFINED FUNCTION apoc.coll.* ON DBMS TO Collector`
- `GRANT EXECUTE BOOSTED PROCEDURE apoc.trigger.add ON DBMS TO TriggerHappy, superAdmin`
- `GRANT EXECUTE BOOSTED USER DEFINED FUNCTION apoc.trigger.add ON DBMS TO TriggerHappy, superAdmin`

- `GRANT EXECUTE BOOSTED PROCEDURE * ON DBMS TO superAdmin`
- `GRANT EXECUTE BOOSTED USER DEFINED FUNCTION * ON DBMS TO superAdmin`
- `DENY EXECUTE BOOSTED PROCEDURE apoc.coll.* ON DBMS TO superAdmin`
- `DENY EXECUTE BOOSTED USER DEFINED FUNCTION apoc.coll.* ON DBMS TO superAdmin`

## 12.8. 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.

*current user*

This is the currently logged-in user invoking the commands described in this chapter.

*password policy*

The password policy is a set of rules of what makes up a valid password. For Neo4j, the following rules apply:

- The password cannot be the empty string.
- When changing passwords, the new password cannot be the same as the previous password.

*role*

This is a collection of actions — such as read and write — permitted on the data.

*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.

# Chapter 13. 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.

## 13.1. 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.

### 13.1.1. 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 83. Allow listing*

In this example, we need to allow the use of the method `apoc.load.json` as well as all the methods under `apoc.coll`. We do not want to make available any additional extensions from the `apoc` library, 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.

### 13.1.2. 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 84. Unrestricting*

In this example, we 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
```

## 13.2. 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.

### 13.2.1. 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.

Follow these steps to use OpenSSL:

- Install a suitable dependency into the `plugins/` folder of Neo4j.

> **Note:**
>
> Dependencies can be downloaded from https://netty.io/wiki/forked-tomcat-native.html. Which dependencies you need depends upon the Neo4j version. Each version of Neo4j ships with a version of Netty and Netty requires specific tcnative versions. Make sure to install the version that matches your OS processor. For more details, see the Netty support per Neo4j version.

- Using non static versions of tcnative will require installation of platform-specific OpenSSL dependencies as described in https://netty.io/wiki/forked-tomcat-native.html.
- Set `dbms.netty.ssl.provider =OPENSSL`.
- Restart Neo4j.

Most supported versions of Neo4j use Netty 4.1.111.Final, which requires tcnative 2.0.65.Final. Only Neo4j 3.5 still uses older versions of Netty. See the table below for detailed information:

*Table 448. Netty support per Neo4j version*

| Neo4j version | Netty version | tcnative version | Direct link |
|---|---|---|---|
| 4.4.36 | 4.1.111.Final | 2.0.65.Final. Both netty-tcnative-boringssl-static and netty-tcnative-classes are required. | https://search.maven.org/artifact/io.netty/netty-tcnative-boringssl-static/2.0.65.Final/jar https://search.maven.org/artifact/io.netty/netty-tcnative-classes/2.0.65.Final/jar |
| 4.4.27 | 4.1.100.Final | 2.0.61.Final. Both netty-tcnative-boringssl-static and netty-tcnative-classes are required. | https://search.maven.org/artifact/io.netty/netty-tcnative-boringssl-static/2.0.61.Final/jar https://search.maven.org/artifact/io.netty/netty-tcnative-classes/2.0.61.Final/jar |
| 4.4.24 | 4.1.94.Final | 2.0.61.Final. Both netty-tcnative-boringssl-static and netty-tcnative-classes are required. | https://search.maven.org/artifact/io.netty/netty-tcnative-boringssl-static/2.0.61.Final/jar https://search.maven.org/artifact/io.netty/netty-tcnative-classes/2.0.61.Final/jar |
| 4.4.16 | 4.1.86.Final | 2.0.54.Final. Both netty-tcnative-boringssl-static and netty-tcnative-classes are required. | https://search.maven.org/artifact/io.netty/netty-tcnative-boringssl-static/2.0.54.Final/jar https://search.maven.org/artifact/io.netty/netty-tcnative-classes/2.0.54.Final/jar |
| 4.4.10 | 4.1.77.Final | 2.0.52.Final. Both netty-tcnative-boringssl-static and netty-tcnative-classes are required. | https://search.maven.org/artifact/io.netty/netty-tcnative-boringssl-static/2.0.52.Final/jar https://search.maven.org/artifact/io.netty/netty-tcnative-classes/2.0.52.Final/jar |

| Neo4j version | Netty version | tcnative version | Direct link |
| --- | --- | --- | --- |
| 4.4.6 | 4.1.75.Final | 2.0.50.Final. Both netty-tcnative-boringssl-static and netty-tcnative-classes are required. | https://search.maven.org/artifact/io.netty/netty-tcnative-boringssl-static/2.0.50.Final/jar https://search.maven.org/artifact/io.netty/netty-tcnative-classes/2.0.50.Final/jar |
| 4.4.4 | 4.1.73.Final | 2.0.46.Final. Both netty-tcnative-boringssl-static and netty-tcnative-classes are required. | https://search.maven.org/artifact/io.netty/netty-tcnative-boringssl-static/2.0.46.Final/jar https://search.maven.org/artifact/io.netty/netty-tcnative-classes/2.0.46.Final/jar |
| 4.4.0 | 4.1.69.Final | 2.0.44.Final. Both netty-tcnative-boringssl-static and netty-tcnative-classes are required. | https://search.maven.org/artifact/io.netty/netty-tcnative-boringssl-static/2.0.44.Final/jar https://search.maven.org/artifact/io.netty/netty-tcnative-classes/2.0.44.Final/jar |



Note:

Using OpenSSL can significantly improve performance, especially for AES-GCM-cryptos, e.g. TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256.

## 13.2.2. Certificates

The SSL configuration requires SSL certificates to be issued by Certificate Authority (CA). All Certificates and the private key must be in `PEM` format.



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:

Valid trusted certificates can be generated for free using non-profit CAs such as Let's Encrypt.

The instructions on this page assume that you have already obtained the required certificates from the CA.



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
```

### 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*

```
PEM - $openssl x509 -in public.crt -text -noout
DER - $openssl x509 -in certificate.der -inform der -text -noout
```

### Transform the certificates

Neo4j requires all SSL certificates to be in the `PEM` format. If your certificate is in the `DER` format, you must transform it into `PEM` format.

*Transform `DER` format certificate to `PEM` format*

```
openssl x509 -in cert.crt -inform der -outform pem -out cert.pem
```

*Transform `PEM` format certificate to `DER` format*

```
openssl x509 -in cert.crt -outform der -out cert.der
```

## 13.2.3. Connectors

Before enabling SSL support, you must ensure the following connector configurations to avoid errors:

- Set `dbms.connector.https.enabled` to `true` when using HTTPS.
- Set `dbms.connector.bolt.tls_level` to `REQUIRED` or `OPTIONAL` when using Bolt.

For more information on configuring connectors, see Configure connectors.

## 13.2.4. 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`, `backup`, and `fabric`.
- `setting-suffix` can be any of the following:

| Setting suffix | Description | Default value |
|---|---|---|
| Basic | | |
| `enabled` | Setting this to `true` enables this policy. | `false` |
| `base_directory` | The base directory under which <term-ssl-cryptographic-objects, 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/` |
| `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 suits allowed during cipher negotiation. Valid values depend on the current JRE and SSL provider. For Ciphers supported by the Oracle JRE, see the Oracle official documentation. | Java platform default allowed cipher suites. |

| Setting suffix | Description | Default value |
| --- | --- | --- |
| `tls_versions` | A comma-separated list of allowed TLS versions. | `TLSv1.2` |
| `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:

```
dbms.connector.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 a different path 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:

```
dbms.connector.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 |

| URI Scheme | Routing | Description |
|---|---|---|
| `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:

   ```
   dbms.connector.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 a different path 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:

```
dbms.connector.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 set up intra-cluster encryption, on each server create the folder structure and place the key file and the certificates under those. Then, you need to configure the SSL cluster policies in the *neo4j.conf* file and test that the intra-cluster communication is encrypted.

1. Set up the *cluster* folder under *certificates*.

a. Create a directory *cluster* under_<neo4j-home>/certificates_ folder:

```
mkdir certificates/cluster
```

b. Create a directory *trusted* and *revoked* under *<neo4j-home>/certificates/cluster* folder:

```
mkdir certificates/cluster/trusted
mkdir certificates/cluster/revoked
```

2. 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
```

3. 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.

4. (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
```

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.

5. Set the cluster SSL configuration in *neo4j.conf*.

a. Set the cluster SSL policy to `true`:

```
dbms.ssl.policy.cluster.enabled=true
```

b. 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 a different path outside of NEO4J_HOME, then set the absolute path for the certificates directory.

c. 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.

d. Verify that the intra-cluster communication is encrypted. You may use an external tooling, such as Nmap (link: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, by default the backup communication happens on port `6362`. 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. These ports are configured by `dbms.backup.listen.address` (port `6362`) and `causal_clustering.transaction_listen_address` (port `6000`) respectively. If the intra-cluster encryption is enabled and the backup communication is using port `6000`, then your communication channels are already encrypted. The following steps assumes that your backup is set up on a different port.

To set up SSL for backup communication, create the folder structure and place the key file and the certificates under those. Then, you need to configure the SSL backup policies in the *neo4j.conf* file.

1. Set up the *backup* folder under *certificates*.
   a. Create a directory *backup* under *<neo4j-home>/certificates* folder:

   ```
   mkdir certificates/backup
   ```

   b. Create a directory *trusted* and *revoked* under *<neo4j-home>/certificates/backup* folder:

   ```
   mkdir certificates/backup/trusted
   mkdir certificates/backup/revoked
   ```

2. 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
   ```

3. Place the *public.crt* file under the *<neo4j-home>/certificates/backup/trusted* folder.

   ```
   cp /path/to/certs/public.crt certificates/backup/trusted
   ```

4. (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
   ```

   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/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-------- |

| Path | Directory/File | Owner | Group | Permission | Unix/Linux View |
|---|---|---|---|---|---|
| /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.

5. Set the backup SSL configuration in *neo4j.conf*.
   a. Set the backup SSL policy to `true`:

```
dbms.ssl.policy.backup.enabled=true
```

   b. 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 a different path outside of NEO4J_HOME, then set the absolute path for the certificates directory.

   c. 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
```

## 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 suits 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

- JSSE Cipher Suite Names.

*Bolt*

```
dbms.ssl.policy.bolt.ciphers=TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
,TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
```

*HTTPS*

```
dbms.ssl.policy.https.ciphers=TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA38
4,TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
```

*Intra-cluster encryption*

```
dbms.ssl.policy.cluster.ciphers=TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA
384,TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
```

*Backup*

```
dbms.ssl.policy.backup.ciphers=TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA3
84,TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
```

### Using OCSP stapling

From version 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:

   ```
   dbms.connector.bolt.ocsp_stapling_enabled=true (default = false)
   ```

## 13.2.5. 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.

```
dbms.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.

## 13.2.6. 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 encryption of Neo4j communications, we make 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, and this 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*.

*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.

## 13.2.7. Configure SSL for FIPS 140-2 compatibility

The National Institute of Standards and Technology (NIST) developed the Federal Information Processing Standard (FIPS) Publication 140-2 as a security standard that sets forth requirements for cryptographic modules, including hardware, software, and/or firmware, for U.S. federal agencies. 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.
- Use Neo4j Enterprise 4.4.36 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:

- 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[1].

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 `netty-tcnative` 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. Download netty-tcnative dynamic library.

   The dynamic version of `netty-tcnative` library provided by netty.io is not compatible with OpenSSL 3.0.0 or later.

   To get around this, we have re-compiled `netty-tcnative` to make OpenSSL 3 compatible netty-tcnative JAR files for use with Neo4j. These can be downloaded from https://assets.neo4j.com/netty-tcnative/2.0.65.Final/simplified/netty-tcnative-2.0.65.Final-all.tar

   There are Linux and Fedora Linux variants available in the tar, compiled for both x86_64 and ARM 64 architectures. Select the one matching the local machine's operating system and architecture.

4. Copy the downloaded `netty-tcnative` JAR file into *<NEO4J_LIB>*.
5. Verify the `netty-tcnative` dependencies are correctly installed using `ldd`:

   *Verify netty-tcnative dependencies are installed*

   ```
   unzip -d /tmp <NEO4J_LIB>/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-*-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.



**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.

### Bolt

1. Set `dbms.netty.ssl.provider =OPENSSL`
2. Set `dbms.connector.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,TLS_AES_128_CCM_8_SHA256,TLS_AES_128_CCM_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,TLS_AES_128_CCM_8_SHA256,TLS_AES_128_CCM_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,TLS_AES_128_CCM_8_SHA256,TLS_AES_128_CCM_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,TLS_AES_128_CCM_8_SHA256,TLS_AES_128_CCM_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.

### Example SSL configurations

*Example Bolt SSL configuration*

```
dbms.ssl.policy.bolt.enabled=true
dbms.ssl.policy.bolt.base_directory=certificates/https
dbms.ssl.policy.bolt.private_key=private.key
dbms.ssl.policy.bolt.public_certificate=public.crt
dbms.ssl.policy.bolt.client_auth=NONE
dbms.ssl.policy.bolt.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,TLS_AES_128_CCM_8_SHA256,TLS_AES_128_CCM_SHA256
```

*Example HTTPS SSL configuration*

```
dbms.netty.ssl.provider=OPENSSL
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
dbms.ssl.policy.https.client_auth=NONE
dbms.ssl.policy.https.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,TLS_AES_128_CCM_8_SHA256,TLS_AES_128_CCM_SHA256
```

## 13.3. 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.

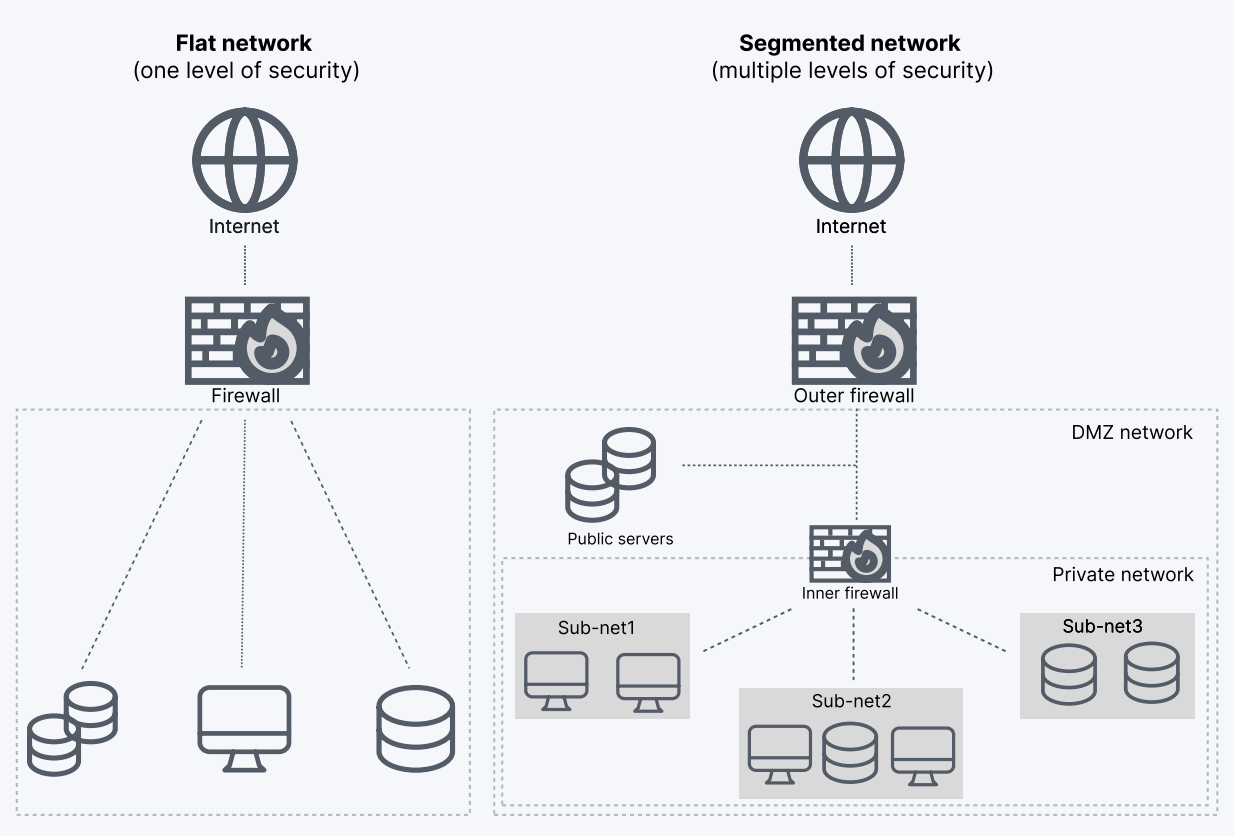
Note:

For more information on how to administer and use Neo4j Browser, see the Neo4j Browser manual → Browser operations.

## 13.4. 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 `dbms.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 File permissions for instructions on permission levels.

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 Causal Clustering, configure and use encryption for intra-cluster communication. For details, see Intra-cluster encryption.
    e. If using Causal 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 correct file permissions 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 File permissions 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 `dbms.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://boringssl.googlesource.com/boringssl/+/master/crypto/fipsmodule/FIPS.md

# Chapter 14. 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.

## 14.1. 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:

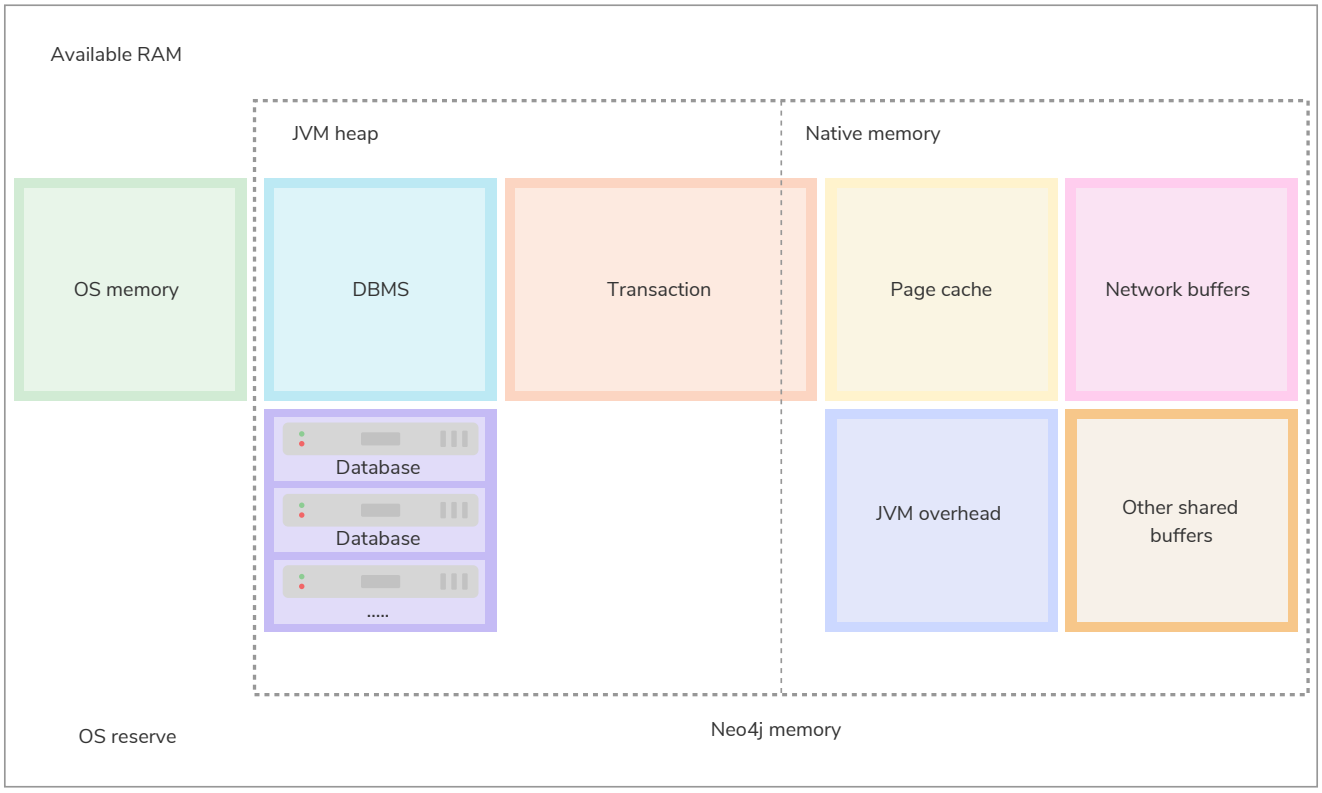



*Figure 15. Neo4j memory management*

*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. If you do not leave enough space for the OS, it will start to swap memory to disk, which will heavily affect performance.

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.

*JVM Heap*

The JVM heap is a separate dynamic memory allocation that Neoj4 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 `dbms.memory.heap.initial_size` and `dbms.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.global_max_size`.

*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: `dbms.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.

## 14.1.1. 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 memrec` 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 memrec` command is useful for inspecting the current distribution of data and indexes.

*Example 85. Use `neo4j-admin memrec` to inspect the memory settings of all your databases*

Estimate the total size of the database files.

```
$neo4j-home> bin/neo4j-admin memrec
...
...
...
# 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 `dbms.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 `dbms.memory.pagecache.size` and `dbms.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.

## 14.1.2. 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 86. 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.

```
$neo4j-home> bin/neo4j-admin memrec
...
...
...
# 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.

`dbms.memory.pagecache.size` = 1.2 * (35GB) = 42GB

You configure the page cache by adding the following to *neo4j.conf*:

```
dbms.memory.pagecache.size=42GB
```

*Example 87. 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 `memrec` command to see the total size of the data and indexes in all current databases.

   ```
   $neo4j-home> bin/neo4j-admin memrec
   ...
   ...
   ...
   # 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.

   ```
   $neo4j-home> bin/neo4j-admin memrec
   ...
   ...
   ...
   # 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.

   `dbms.memory.pagecache.size` = 1.2 * (40GB) = 48GB

5. Configure the page cache by adding the following to *neo4j.conf*:

   ```
   dbms.memory.pagecache.size=48G
   ```

## 14.1.3. Limit transaction memory usage

By using the `dbms.memory.transaction.global_max_size` 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 `dbms.memory.transaction.database_max_size` limits the transaction memory usage per database.
- The setting `dbms.memory.transaction.max_size` 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()
CALL dbms.listTransactions()
CALL dbms.listQueries()
```

Or alternatively, you can enable `dbms.logs.query.allocation_logging_enabled` and monitor the memory usage of each query in the query.log.



**Note:**

By default, transaction sizes are unconstrained. However, in a Cluster deployment, where an instance is configured as either a Single, Core or a Read Replica, the maximum amount of memory each transaction is permitted to use is 2GB. Consequently, if your configuration contains `dbms.mode=SINGLE`, `dbms.mode=CORE`, or `dbms.mode=READ_REPLICA`, the largest value permitted for the `dbms.memory.transaction.max_size` setting is also 2GB.

# 14.2. Index configuration

This page describes how to configure Neo4j indexes to enhance search performance and enable full-text search. The supported index types are:

- B-tree
- Text
- Full-text
- Token lookup

All four 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.

B-tree, 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.

Users are not required to know the difference between the various indexes to use them since Cypher's query planner decides which index to use in which situation.

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 following table:

| Index type | Cypher command | Core API |
| --- | --- | --- |
| B-tree index | `SHOW BTREE INDEXES` | `org.neo4j.graphdb.schema.IndexType#BTREE` |
| Text index | `SHOW EXT 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` |

## 14.2.1. B-tree indexes

B-tree indexes are deprecated, partially replaced for now, and will be fully replaced in 5.0 by the future indexes. In 4.4, B-tree indexes are still the correct alternative, except when the `lucene+native-3.0` provider is used. A new text index type has been introduced to handle the case `lucene+native-3.0` covered for single property strings.

B-tree indexes are suitable for exact lookups on all types of values, range scans, full scans, and prefix searches. They can be backed by two different index providers, `native-btree-1.0` and `lucene+native-3.0`. If not explicitly set, `native-btree-1.0` is used.

### Limitations

B-tree indexes have some limitations that can be worked around by using text indexes or the deprecated index provider `lucene+native-3.0` instead.

#### Limitations for queries using `CONTAINS` and `ENDS WITH`

The index provider `native-btree-1.0` has limited support for `ENDS WITH` and `CONTAINS` queries. These queries are not able to do an optimized search as per queries that use `STARTS WITH`, `=`, and `<>`. Instead, the index result is a stream of an index scan with filtering.

`ENDS WITH` and `CONTAINS` queries are natively supported by text indexes, and these types of queries are handled by a text index in a more performant way than a B-tree index. Even though the deprecated index provider `lucene+native-3.0` can also be used for these queries, it provides no extra value over a text index and it will be removed in 5.0.



Note:

`lucene+native-3.0` and text indexes only support `ENDS WITH` and `CONTAINS` for single property strings.

- For details about execution plans, refer to Cypher Manual → Execution plans.
- For details about string operators, refer to Cypher Manual → Operators.

### Limitations on key size

The index provider `native-btree-1.0` has a key size limit of approximately 8kB.

If a transaction exceeds this limit for any of its changes, it will fail before committing any changes. If the limit is reached during index population, the resulting index will be in a failed state and cannot be used for any queries.

In such a situation, you can use the index provider `lucene+native-3.0`. This provider has a key size limit for single property strings of approximately 32kB.

### Workarounds to address limitations

To work around problems with key size or performance issues related to `ENDS WITH` or `CONTAINS`, you can use the text index type or the deprecated index provider `lucene+native-3.0`. However, it only works for single-property string indexes.

**Creating a text index** is recommended to work around key size or performance issues. For details on the syntax for creating a text index, see Cypher Manual → Indexes for search performance. For more information about text indexes see Text indexes.

Alternatively, you can use one of the following methods:

- *Use* `lucene+native-3.0` *in the* `OPTIONS` *clause in the* `CREATE` *command.*

  Note that this option uses the `lucene+native-3.0` index provider, which is deprecated and will be removed in a future release.

  The Cypher commands for index creation, unique property constraint creation, and node key creation contain an optional `OPTIONS` clause, which can be used to specify the index provider.

  For details on indexes, see Cypher Manual → Indexes for search performance. For further information on constraints, see Cypher manual → Constraints.

- *Use a built-in procedure.*

  Note that this option uses deprecated built-in procedures that will be removed in a future release. These have been replaced with the Cypher commands in the first option above.

  The built-in procedures `db.createIndex`, `db.createUniquePropertyConstraint`, and `db.createNodeKey` can be used to specify the index provider on index creation, unique property constraint creation, and node key creation.

  For details on constraints, see Procedures.

- *Change the config.*

  Note that this option uses the index setting `dbms.index.default_schema_provider`, which has been

deprecated and will be removed in a future release. It will be a fully internal concern which index provider an index uses.

1. Configure the setting `dbms.index.default_schema_provider` to the one required.
2. Restart Neo4j.
3. Drop and recreate the relevant index.
4. Change `dbms.index.default_schema_provider` back to the original value.
5. Restart Neo4j.

The recommended way to set an index provider for an index is to use the `OPTIONS` clause for index creation, unique property constraint creation, and node key creation. For more information, see Cypher manual → Constraints.

## Index migration

When upgrading a 3.5 store to 4.4.48, all indexes are upgraded to the latest index version and rebuilt automatically, except for those previously using Lucene for single-property strings. They have been upgraded to a fallback version that uses Lucene for those properties. Note that they still need to be rebuilt. For more information, see Upgrade and Migration Guide → Neo4j indexes.

## Procedures to create an index and index-backed constraint

Indexes and constraints are best created using the optional `OPTIONS` clause of the `CREATE` Cypher command. Alternatively, you can create them via the deprecated procedures described in the following example:

The following procedures provide the option to specify both index provider and index settings (optional). Note that settings keys need to be escaped with back-ticks if they contain dots.

*Example 88. Use `db.createIndex` procedure to create an index and specify the index provider and settings*

```
CALL db.createIndex("MyIndex", ["Person"], ["name"], "native-btree-1.0", {`spatial.cartesian.max`:
[100.0,100.0], `spatial.cartesian.min`: [-100.0,-100.0]})
```

*Example 89. Use procedures to create an index and an index-backed constraint without specifying the setting map*

If a settings map is not provided, the settings are picked up from the Neo4j config file, the same way as when creating an index or constraint through Cypher.

*Use `db.createIndex` to create an index without specifying the setting map*

```
CALL db.createIndex("MyIndex", ["Person"], ["name"], "native-btree-1.0")
```

*Use `db.createUniquePropertyConstraint` to create a node property uniqueness constraint without specifying the setting map*

```
CALL db.createUniquePropertyConstraint("MyIndex", ["Person"], ["name"], "native-btree-1.0")
```

*Use `db.createNodeKey` to create node key constraint without specifying the setting map*

```
CALL db.createNodeKey("MyIndex", ["Person"], ["name"], "native-btree-1.0")
```

## 14.2.2. Text indexes

Text indexes are a type of single-property index and only index properties with string values, unlike B-tree indexes. They are specifically designed to deal with `ENDS WITH` or `CONTAINS` queries efficiently. They are used through Cypher and support a smaller set of string queries. Even though text indexes support other text queries, `ENDS WITH` or `CONTAINS` queries are the only ones for which this index type provides an advantage over a B-tree index.

For more information, refer to Cypher Manual → Indexes for search performance.

### Limitations

Text indexes only index single property strings. If the property to index can contain several value types, but string-specific queries are also performed, it is possible to have both a B-tree and a text index on the same schema.

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.

## 14.2.3. Full-text indexes

Full-text indexes are optimized for indexing and searching text. They make it possible to write queries that match the *contents* of indexed string properties. In other words, they are used for queries that demand an understanding of language and only index string data. They must also be queried explicitly via procedures, as Cypher does not make plans that rely on them.

An example of a use case for full-text indexes is parsing a book for a specific 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 words that are not relevant for the search (for example, "*if*" and "*and*") and includes conjugations of words that are.

Another use case example is indexing the various address fields and text data in a corpus of emails. Indexing this data using the `email` analyzer makes it possible to find all emails sent from, to, or about an email account.

In contrast to B-tree 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

they 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.

### Configuration

The following options are available for configuring full-text indexes. For a complete list of Neo4j procedures, see Operations Manual → Procedures.

`dbms.index.fulltext.default_analyzer`

The default analyzer for full-text indexes. This setting takes effect when creating an index and is remembered as an index-specific setting.

The list of possible analyzers is available through the `db.index.fulltext.listAvailableAnalyzers()` Cypher procedure.

Unless otherwise specified, the default analyzer is standard-no-stop-words, the same as the StandardAnalyzer from Lucene, except no stop-words are filtered out.

`dbms.index.fulltext.eventually_consistent`

Used to declare whether full-text indexes should be eventually consistent or not. This setting only has effect when a full-text index is created and is remembered as an index-specific setting from then on.

Indexes usually are fully consistent, and the committing of a transaction returns once both the store and the indexes are updated. On the other hand, eventually consistent full-text indexes are not updated as part of the commit; instead, their updates are queued up and applied in a background thread. This means there can be a short delay between committing a change and the change becoming visible via any eventually consistent full-text indexes. This delay is just an artefact of the queueing and is usually relatively small since. Eventually, consistent indexes are updated "as soon as possible".

This is turned off by default, and full-text indexes are fully consistent.

`dbms.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, the commit transactions block and wait until there is more room in the queue before adding more updates.

This setting applies to all eventually consistent full-text indexes, and they all use the same queue. Due to considerations regarding heap space usage, the maximum queue length must be at least 1 index update and no more than 50 million.

The default maximum queue length is 10.000 index updates.

## 14.2.4. Token lookup indexes

Token lookup indexes, as the name suggests, are used to look up nodes with a specific label or relationships of a specific type. A token lookup index is always created over all labels or relationship types,

respectively, and hence, a database can only have a maximum of two token lookup indexes — one for nodes and one for relationships.

Token lookup indexes are introduced in 4.3, and whereas the relationship type lookup index is a new concept, the node label lookup index is not. The latter evolved from the label scan store, which has existed in various forms for a long time. The node label lookup index provides the same functionality as the former label scan store. Still, it has additional features common to all indexes, such as the ability to be created and dropped using a non-blocking population.

## Use and significance

Token lookup indexes are the most important indexes that can be present in a database. They are essential for both Cypher queries and Core API operations. More importantly, their presence significantly speeds up the population of other indexes: the node label lookup index for node B-tree and full-text indexes and the relationship type lookup index for the corresponding relationship indexes.

The node label lookup index is essential for queries that match a node by one or more labels. It can be used even when matching labels and properties of a node if no suitable B-tree indexes are available. This is essential, considering that no B-tree indexes are defined by default. In other words, a node label lookup index is often the best approach to a query that matches labels unless the user has defined a more appropriate B-tree index. Accordingly, the relationship type lookup index does the same for relationships and their types.

Most queries are executed by matching nodes and expanding their relationships, and hence, the node label lookup index is slightly more significant than the relationship type lookup index.

Since these indexes are important for both query execution and index population, they should be carefully considered before being dropped.

Node and relationship type lookup indexes are present by default in all databases created in 4.3 and onwards.

## Databases created before 4.3

By default, databases created before 4.3 get only a node label lookup index when used in a DBMS of version 4.3 or later. This is to preserve the backward compatibility and performance characteristics of such databases.

Such databases can get a relationship-type lookup index by explicitly creating it through Cypher if needed.



**Note:**

Creating a relationship-type lookup index on a large database can take significant time, as all relationships must be scanned when populating such an index.

When used in a Neo4j 4.3 or later, databases created before 4.3 automatically get a node label lookup index. This index is created by converting the former label scan store and naming it `__org_neo4j_schema_index_label_scan_store_converted_to_token_index`. This index name is reserved from 4.3 onwards, and an error is returned if you attempt to create a user-defined index with this name.

Similarly, in the unlikely situation when an index with such a name exists in a previous version, it must be dropped and recreated with a different name before upgrading to 4.3.

The following table summarizes which token lookup indexes and label scan stores are present by default in various versions. Note that the table represents only the default indexes and that the relationship type lookup index can be created explicitly through Cypher if needed.

| Database created | before 4.3 | | from 4.3 |
|---|---|---|---|
| Neo4j version | < 4.3 | >= 4.3 | |
| Label scan store | yes | no | no |
| Node label lookup index | no | yes | yes |
| Relationship type lookup index | no | no | yes |

### 14.2.5. Future indexes

Two new index types, range and point index, will be introduced in 5.0. They will, together with the text index, replace the deprecated B-tree indexes.

Like the B-tree index, the range index will index all types of values and be good for exact lookups on all types of values, range scans, full scans, and prefix searches. The difference is that the range index will not support spatial queries and therefore will not have the same config options. It will still index the point values to support full scans, but if spatial queries are needed, a point index should be created.

The point index is a highly specialized single-property index that is optimized for spatial queries. It only indexes point values and exact lookups are the only non-spatial query it supports.

These indexes can be created on the same combination of property and label/relationship type if the functionality of both is needed.

It is possible to create and drop these index types, but they cannot be used in queries yet. They are introduced now to allow a smoother migration to 5.0 later. See Cypher Manual → Indexes for search performance → Future indexes for the new syntax.

## 14.3. 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 `dbms.memory.heap.max_size` (in MBs) setting in the *neo4j.conf* file. The initial size of the heap is specified by the `dbms.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 449. neo4j.conf JVM tuning properties*

| Property Name | Meaning |
| --- | --- |
| `dbms.memory.heap.initial_size` | initial heap size (in MB) |
| `dbms.memory.heap.max_size` | maximum heap size (in MB) |
| `dbms.jvm.additional` | additional literal JVM parameter |

# 14.4. 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.

## 14.4.1. 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.

## 14.4.2. Configuration options

The following configuration options are available for configuring the Bolt connector:

*Table 450. Thread pool options*

| Option name | Default | Description |
| --- | --- | --- |
| `dbms.connector.bolt.thread_pool_min_size` | `5` | The minimum number of threads that will always be up even if they are idle. |
| `dbms.connector.bolt.thread_pool_max_size` | `400` | The maximum number of threads that will be created by the thread pool. |

| Option name | Default | Description |
| --- | --- | --- |
| `dbms.connector.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 `dbms.connector.bolt.thread_pool_min_size`. |

### 14.4.3. 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 `dbms.connector.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 `dbms.connector.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 90. Configure the thread pool for a Bolt connector*

In this example we configure the Bolt thread pool to be of minimum size `5`, maximum size `100`, and have a keep-alive time of `10 minutes`.

```
dbms.connector.bolt.thread_pool_min_size=10
dbms.connector.bolt.thread_pool_max_size=100
dbms.connector.bolt.thread_pool_keep_alive=10m
```

## 14.5. 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.

## 14.6. 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.

### 14.6.1. 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.

### 14.6.2. 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.

## 14.6.3. Active page cache warmup

Neo4j Enterprise Edition has a feature called *active page cache warmup*, which is enabled by default via the `dbms.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 `dbms.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 `dbms.memory.pagecache.warmup.preload.allowlist` setting. It takes a regular expression as a value to match the files.

*Example 91. 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 `dbms.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.

### 14.6.4. 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 `dbms.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 transaction logs 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.

## 14.7. 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.

### 14.7.1. 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 |
|---|---|---|
| `dbms.index_sampling.background_enabled` | `true` | Enable the automatic (background) index sampling. |
| `dbms.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()
```

# 14.7.2. 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 |
| --- | --- | --- |
| `cypher.statistics_divergence_threshold` | `0.75` | The threshold for statistics above which a plan is considered stale.<br>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)`.<br>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. |
| `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 `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 → Query tuning
- Cypher manual → Execution plans
- Procedures

## 14.8. 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, 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, 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.

### 14.8.1. 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, the `neostore.nodestore.db.id` contains the IDs of all deleted nodes.

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.

## 14.8.2. Reclaim unused space

You can use the `neo4j-admin 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 92. Example of database compaction using* `neo4j-admin 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.

```
$neo4j-home/bin$> ./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 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 copy` command to create a copy of your `neo4j` database.

```
$neo4j-home/bin$> ./neo4j-admin copy --to-database=neo4jcopy1 --from-database=neo4j --force
--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         | aliases | access       | address          | role         | requestedStatus |
currentStatus | error | default | home  |
+-----------------------------------------------------------------------------------------------------------
-------------------------------------+
| "neo4j"      | []      | "read-write" | "localhost:7687" | "standalone" | "offline"       |
"offline"     | ""    | TRUE    | TRUE  |
| "neo4jcopy1" | []      | "read-write" | "localhost:7687" | "standalone" | "online"        |
"online"      | ""    | FALSE   | FALSE |
| "system"     | []      | "read-write" | "localhost:7687" | "standalone" | "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.

# Chapter 15. 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:


- Monitor the logs
- Metrics
  - Essential metrics
  - Enable metrics logging
  - Connect monitoring tools
  - Metrics reference
- Manage queries
  - List all running queries
  - List all active locks for a query
  - Terminate queries
- Manage transactions
  - Configure transaction timeout
  - Configure lock acquisition timeout
  - List all running transactions
- Manage connections
  - List all network connections
  - Terminate multiple network connections
  - Terminate a single network connection
- Manage background jobs
  - Listing active background jobs
  - Listing failed job executions
- Monitor a Causal Cluster
  - Procedures for monitoring a Causal Cluster
  - Monitor cluster endpoints for status information
- Monitor the state of individual databases

# 15.1. Logging

## 15.1.1. Log files

Neo4j provides logs for monitoring purposes. The root directory where the general log files are located is configured by `dbms.directories.logs`. The default format of the log files is configured by `dbms.logs.default_format`. For more information on where files are located, see File locations.

The following table describes the Neo4j general log files and the information they contain.

*Table 451. Neo4j logs for monitoring*

| Filename | Name | Description |
|---|---|---|
| *neo4j.log* | The *user log* | Logs general information about Neo4j. For Debian and RPM packages run `journalctl --unit=neo4j`. |
| *debug.log* | The *debug log* | Logs information that is useful when debugging problems with Neo4j. |
| *http.log* | The *HTTP log* | Logs information about the HTTP API. |
| *gc.log* | The *garbage collection log* | Logs information provided by the JVM. |
| *query.log* | The *query log* | Logs information about queries that run longer than a specified threshold. Enterprise |
| *security.log* | The *security log* | Logs information about security events. Enterprise |
| *service-error.log* | The *windows service log* | Logs information about errors encountered when installing or running the Windows service. Windows |

*Table 452. Log paths*

| Configuration setting | Default value | Description |
|---|---|---|
| `dbms.directories.logs` | `logs` | Path to the logs directory. |
| `dbms.logs.user.path` | `neo4j.log` | Path to the *user log* file. |
| `dbms.logs.debug.path` | `debug.log` | Path to the *debug log* file. |
| `dbms.logs.http.path` | `http.log` | Path to the *HTTP log* file. |
| `dbms.logs.query.path` | `query.log` | Path to the *query log* file. |
| `dbms.logs.security.path` | `security.log` | Path to the *security log* file. |

## 15.1.2. Log format

*Table 453. Log formats*

| Configuration setting | Default value | Description |
|---|---|---|
| `dbms.logs.default_format` | `PLAIN` | The default log format for all logs. Valid options are `PLAIN` and `JSON`. |

| Configuration setting | Default value | Description |
| --- | --- | --- |
| `dbms.logs.user.format` | Inherits from `dbms.logs.default_format` | The log format for the *user log*. Valid options are `PLAIN` and `JSON`. |
| `dbms.logs.query.format` | Inherits from `dbms.logs.default_format` | The log format for the *query log*. Valid options are `PLAIN` and `JSON`. |
| `dbms.logs.debug.format` | Inherits from `dbms.logs.default_format` | The log format for the *debug log*. Valid options are `PLAIN` and `JSON`. |
| `dbms.logs.security.format` | Inherits from `dbms.logs.default_format` | The log format for the *security log*. Valid options are `PLAIN` and `JSON`. |

## 15.1.3. Log configurations

There are a number of configuration options to enable/disable logging, decide how to rotate the logs, and how many of the logs to keep around.

### User log (*neo4j.log*)

*Table 454. User log configurations*

| The *user log* configuration | Default value | Description |
| --- | --- | --- |
| `dbms.logs.user.format` | Inherits from `dbms.logs.default_format` | The log format for the *user log*. |
| `dbms.logs.user.rotation.delay` | `5m` | The minimum time interval after the last rotation of the *user log* before it may be rotated again. |
| `dbms.logs.user.rotation.keep_number` | `7` | The maximum number of history files for the *user log*. |
| `dbms.logs.user.rotation.size` | `0B` | The threshold size for rotation of the *user log*. If set to `0`, the log rotation is disabled. |
| `dbms.logs.user.stdout_enabled` | `true` | Send user logs to the process stdout. If disabled, the logs are sent to the *user log*. |

The following information is available in the JSON format:

*Table 455. JSON format log entries*

| Name | Description |
| --- | --- |
| `time` | The timestamp of the log message. |
| `level` | The log level. |
| `message` | The log message. |
| `stacktrace` | Included when there is a stacktrace associated with the log message. |

## Debug log

The *debug log* logs problems, errors, stacktraces, etc.

*Table 456. Debug log configurations*

| The *debug log* configuration | Default value | Description |
|---|---|---|
| `dbms.logs.debug.level` | `INFO` | Log level threshold for the *debug log*. |
| `dbms.logs.debug.format` | Inherits from `dbms.logs.default_format` | The log format for the *debug log*. |
| `dbms.logs.debug.rotation.delay` | `5m` | The minimum time interval after the last rotation of the *debug log* before it may be rotated again. |
| `dbms.logs.debug.rotation.keep_number` | `7` | The maximum number of history files for the *debug log*. |
| `dbms.logs.debug.rotation.size` | `20M` | The threshold size for rotation of the *debug log*. |

The following table lists all message types raised by Neo4j and their severity level:

*Table 457. Message types*

| Message type | Severity level | Description |
|---|---|---|
| `INFO` | Low severity | Report status information and errors that are not severe. |
| `DEBUG` | Low severity | Report details on the raised errors and possible solutions. |
| `WARN` | Low severity | Report errors that need attention but are not severe. |
| `ERROR` | High severity | Report errors that prevent the Neo4j server from running and must be addressed immediately. |

To set the log level threshold for the *debug log*, use the configuration setting `dbms.logs.debug.level`.

The following information is available in the JSON format:

*Table 458. JSON format log entries*

| Name | Description |
|---|---|
| `time` | The timestamp of the log message. |
| `level` | The log level. |
| `category` | The class the message was logged from. |
| `message` | The log message. |
| `stacktrace` | Included when there is a stacktrace associated with the log message. |

## Garbage collection log

*Table 459. Garbage collection log configurations*

| The *garbage collection log* configuration | Default value | Description |
|---|---|---|
| `dbms.logs.gc.enabled` | `false` | Enable garbage collection logging. |
| `dbms.logs.gc.options` | | Garbage collection logging options. |
| `dbms.logs.gc.rotation.keep_number` | `0` | The maximum number of history files for the garbage collection log. |
| `dbms.logs.gc.rotation.size` | | The threshold size for rotation of the garbage collection log. |

## HTTP log

*Table 460. HTTP log configurations*

| The *HTTP log* configuration | Default value | Description |
|---|---|---|
| `dbms.logs.http.enabled` | `false` | Enable HTTP logging. |
| `dbms.logs.http.rotation.keep_number` | `5` | The maximum number of history files for the *HTTP log*. |
| `dbms.logs.http.rotation.size` | `20M` | The threshold size for rotation of the *HTTP log*. |

## Security log

Neo4j provides security event logging that records all security events.

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.
- All security procedures run against the `system` database.
- Authorization failures from role-based access control.

Rotation of the security events log can be configured in the *neo4j.conf* configuration file.

The following configuration settings are available for the *security log*:

*Table 461. Security log configurations*

| The *security log* configuration | Default value | Description |
|---|---|---|
| `dbms.logs.security.level` | `INFO` | Security log level threshold. |
| `dbms.logs.security.format` | Inherits from `dbms.logs.default_format` | The log format for the *security log*. |

| The *security log* configuration | Default value | Description |
| --- | --- | --- |
| `dbms.logs.security.path` | `security.log` | The name of the *security log* file. |
| `dbms.logs.security.rotation.size` | `20M` | The file size at which the security event log auto-rotates. |
| `dbms.logs.security.rotation.delay` | `300s` | The minimum time interval after the last *security log* rotation occurred before the *security log* may be rotated again. |
| `dbms.logs.security.rotation.keep_number` | `7` | The number of historical log files kept. |

If using LDAP as the authentication method, some cases of LDAP misconfiguration are also logged, as well as 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 following information is available in the JSON format:

*Table 462. 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. |
| `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. |

Example of the security log in 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
```

## Query log

Query logging is enabled by default and is controlled by the setting `dbms.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.

The following configuration settings are available for the *query log*:

*Table 463. Query log enabled setting*

| Option | Description |
|---|---|
| `OFF` | Completely disable logging. |
| `INFO` | Log at the end of queries that have either succeeded or failed. The `dbms.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` | Log all queries at both start and finish, regardless of `dbms.logs.query.threshold`. Default |

The name of the query log file is *query.log* by default. For more information, see `dbms.logs.query.path`.

You can configure the rotation of the query log in the *neo4j.conf* file.

The following configuration settings are available for the query log file:

*Table 464. Query log configurations*

| The *query log* configuration | Default value | Description |
|---|---|---|
| `dbms.logs.query.allocation_logging_enabled` | `true` | Log allocated bytes for the executed queries being logged. The logged number is cumulative over the duration of the query. That means the value may be larger for memory intense or long-running queries than the current memory allocation. Requires `dbms.track_query_allocation=true`. |
| `dbms.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. |
| `dbms.logs.query.enabled` | `VERBOSE` | Log executed queries. |
| `dbms.logs.query.format` | Inherits from `dbms.logs.default_format` | The log format for the *query log*. For logging detailed time information requires `dbms.track_query_cpu_time=true`. |

| The *query log* configuration | Default value | Description |
| --- | --- | --- |
| `dbms.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. |
| `dbms.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.<br><br>It is recommended to set this setting to `true` in production.<br><br>**Note:**<br><br>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()`.<br><br>**Warning:**<br><br>The setting does not obfuscate literals in parameters.<br>If parameter values are not required in the log, set `dbms.logs.query.parameter_logging_enabled=false`.<br><br>The setting does not obfuscate any sensitive information when logging errors.<br>To get that capability, set `db.logs.query.obfuscate_errors=true`. |

| The *query log* configuration | Default value | Description |
| --- | --- | --- |
| Introduced in 4.4.48<br>`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 the `failureReason` field.<br><br>It is recommended to set this setting to `true` in production.<br><br>**Warning:**<br><br>The setting does not obfuscate literals when logging Cypher queries. To get that capability, set `dbms.logs.query.obfuscate_literals=true`. |
| `dbms.logs.query.page_logging_enabled` | `false` | Log page hits and page faults for the executed queries being logged. |
| `dbms.logs.query.parameter_full_entities` | `false` | Log complete parameter entities including ID, labels or relationship type, and properties. If `false`, only the entity ID is logged. This only takes effect if `dbms.logs.query.parameter_logging_enabled=true`. |
| `dbms.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. |

| The *query log* configuration | Default value | Description |
| --- | --- | --- |
| `dbms.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.<br><br>**Note:**<br><br>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. |
| `dbms.logs.query.rotation.keep_number` | `7` | The maximum number of history files for the *query log*. |
| `dbms.logs.query.rotation.size` | `20M` | The file size in bytes at which the *query log* auto-rotates. |
| `dbms.logs.query.runtime_logging_enabled` | `true` | Logs which runtime that was used to run the query. |
| `dbms.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. |
| `dbms.logs.query.time_logging_enabled` | `false` | Log detailed time information for the executed queries being logged, such as `(planning: 92, waiting: 0)`. Eanbling `dbms.track_query_cpu_time=true` as well, adds the CPU time used by current transaction in milliseconds, e.g., `(planning: 95, cpu: 96, waiting: 0)`. |

| The *query log* configuration | Default value | Description |
| --- | --- | --- |
| `dbms.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 (`dbms.logs.query.transaction.threshold`). |
| `dbms.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. |
| `dbms.logs.query.transaction_id.enabled` | `false` | This configuration option allows the administrator to request the transaction ID is included with the query ID in all query log entries. Queries are run as part of a transaction. For simple queries, there is usually a 1:1 correlation. In application usage, however, a transaction can include many queries, especially if retries are required in the event of connection instability. |

*Example 93. 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.

```
dbms.logs.query.enabled=INFO
```

Below 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  ...
```

*Example 94. Configure for query logging with more details*

In this example, the query log is enabled, as well as some additional logging:

```
dbms.logs.query.enabled=INFO
dbms.logs.query.allocation_logging_enabled=true
dbms.logs.query.page_logging_enabled=true
dbms.logs.query.parameter_logging_enabled=true
dbms.logs.query.time_logging_enabled=true
dbms.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 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 allocated bytes for the executed queries being logged. 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 page cache as opposed to disk. In this case, 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 MovieGraph dataset from the Neo4j Browser guide.

*Example 95. Using `cypher-shell`, attach metadata to a transaction*

```
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 96. 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 97. 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.

### JSON format

The following information is available in the JSON format:

*Table 465. 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 466. JSON format log entries*

| Name | Description |
| --- | --- |
| `event` | Valid options are `start`, `fail`, and `success`. |
| `id` | The query ID. Included when `dbms.logs.query.enabled` is `VERBOSE`. |
| `elapsedTimeMs` | The elapsed time in milliseconds. |
| `planning` | Milliseconds spent on planning. Included when `dbms.logs.query.time_logging_enabled` is `true`. |
| `cpu` | Milliseconds spent actively executing on the CPU. Included when `dbms.logs.query.time_logging_enabled` and `dbms.track_query_cpu_time=true` are `true`. |
| `waiting` | Milliseconds spent waiting on locks or other queries, as opposed to actively running this query. Included when `dbms.logs.query.time_logging_enabled` is `true`. |
| `allocatedBytes` | Number of bytes allocated by the query. Included when `dbms.logs.query.allocation_logging_enabled` is `true`. |
| `pageHits` | Number of page hits. Included when `dbms.logs.query.page_logging_enabled` is `true`. |
| `pageFaults` | Number of page faults. Included when `dbms.logs.query.page_logging_enabled` is `true`. |
| `source` | Connection details. |
| `database` | The database name on which the query is run. |
| `username` | The user running the query. This field is deprecated by `executingUser`. |
| `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 `dbms.logs.query.parameter_logging_enabled` is `true`. |
| `runtime` | The runtime used to run the query. Included when `dbms.logs.query.runtime_logging_enabled` is `true`. |
| `annotationData` | Metadata attached to the transaction. |

| Name | Description |
| --- | --- |
| `failureReason` | Reason for failure. Included when applicable. |
| `transactionId` | The transaction ID of the running query. Included when `dbms.logs.query.transaction_id.enabled` is `true`. |
| `queryPlan` | The query plan. Included when `dbms.logs.query.plan_description_enabled` is `true`. |

If the type of the log entry is `transaction`, the following additional fields are available:

*Table 467. JSON format log entries*

| Name | Description |
| --- | --- |
| `event` | Valid options are `start`, `rollback`, and `commit`. |
| `database` | The database name on which the transaction is run. |
| `username` | The user connected to the transaction. This field is deprecated by `executingUser`. |
| `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. |

# 15.2. Metrics

You can configure Neo4j to log many different metrics to help you keep your applications running smoothly. By default, this data can be retrieved from CSV files or exposed over JMX MBeans, but you can also export them 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 JMX, CSV files, or other third-party monitoring tools.
- Metrics reference — Neo4j available metrics.

## 15.2.1. Essential metrics

To ensure your applications are running smoothly, it is good to 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. |
| 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.<br><br>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. |

## 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 | `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 | `page_cache.hit_ratio` and `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. |

| Metric name | Metric | Description |
| --- | --- | --- |
| JVM garbage collection | `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. |
| Checkpoint time | `neo4j.<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:<br>• Raise the `dbms.checkpoint.iops.limit` to make checkpoints faster, but only if there is enough IOPS budget available to avoid slowing the commit process.<br>• `dbms.memory.pagecache.flush.buffer.enabled` / `dbms.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).<br>• Change the checkpointing policy (`dbms.checkpoint.*`, `dbms.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 Log pruning. |

## Neo4j cluster health metrics

The cluster health metrics indicate the health of a cluster member at a glance. It is important to know which instance is the leader. The leader has a different load pattern from the followers, which should exhibit similar load patterns.

| Metric name | Metric | Description |
| --- | --- | --- |
| Leader | `neo4j.causal_clustering.core.is_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. |

| Metric name | Metric | Description |
| --- | --- | --- |
| Transaction workload | `neo4j.transaction.last_committed_tx_id` | The ID of the last committed transaction. Track this for each Neo4j instance. In a cluster setup, track this for each Core cluster member and Read Replica. 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 member is no longer replicating data, and action is needed to rectify the situation. |

### Workload metrics

These metrics are useful for monitoring 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>.bolt.connections_running` | The number of connections that are currently executing cypher and returning results. |
| Total nodes/relationships | `neo4j.count.node` and `neo4j.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 | `<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 full list of all available metrics in Neo4j, see Metrics reference.

## 15.2.2. Enable metrics logging

> **Note:**
>
> A subset of all available metrics is enabled by default. See `metrics.filter`. The list was last updated in version 4.2.

You can enable/disable metrics using the configuration setting `metrics.enabled`. You can also use the setting `metrics.filter` to enable only the metrics that 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.
metrics.enabled=true

# Setting for exposing metrics.
```

```
metrics.filter=*check_point*,neo4j.page_cache.evictions
```

When specifying a complete metric name, you should take into account whether `metrics.namespaces.enabled` is set.

All metrics are logged to CSV files in the */metrics* folder.

## 15.2.3. Expose metrics

Neo4j supports the following ways of exposing data for monitoring purposes:

- 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.

### 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'.
metrics.csv.enabled=true
# Directory path for output files.
# Default is a "metrics" directory under NEO4J_HOME.
#dbms.directories.metrics='/local/file/system/path'
# How often to store data. Default is 30 seconds.
metrics.csv.interval=30s
# The maximum number of CSV files that will be saved. Default is 7.
metrics.csv.rotation.keep_number=7
# The file size at which the csv files will auto-rotate. Default is 10M.
metrics.csv.rotation.size=10M
# Compresses the metric archive files.
metrics.csv.rotation.compression=zip
```

`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 version 4.2.2 onwards, the JMX metrics are exposed by default over JMX MBeans.

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* in order to enable integration with Graphite:

```
# Enable the Graphite integration. Default is 'false'.
metrics.graphite.enabled=true
# The IP and port of the Graphite server on the format <hostname or IP address>:<port number>.
# The default port number for Graphite is 2003.
metrics.graphite.server=localhost:2003
# How often to send data. Default is 30 seconds.
metrics.graphite.interval=30s
# Prefix for Neo4j metrics on Graphite server.
metrics.prefix=Neo4j_1
```

Start Neo4j and connect to Graphite via a web browser in order 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 link: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* in order to enable the Prometheus endpoint.

```
# Enable the Prometheus endpoint. Default is 'false'.
metrics.prometheus.enabled=true
# The IP and port the endpoint will bind to in the format <hostname or IP address>:<port number>.
# The default is localhost:2004.
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 `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 `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 permissible, such as `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 other any insecure networks.

## 15.2.4. 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.

### Global metrics

Global metrics cover the whole database management system, and represents the status of the system as a whole.

Global metrics have the following name format: `<user-configured-prefix>.<metric-name>` if `metrics.namespaces.enabled` is `false`, or `<user-configured-prefix>.dbms.<metric-name>` if the setting is `true`. The <user-configured-prefix> can be configured with the `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.vm.thread.count` metric has a default user-configured-prefix `neo4j` and the 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-name>.<metric-name>` if `metrics.namespaces.enabled` is `false`, or `<user-configured-prefix>.database.<database-name>.<metric-name>` if the setting is `true`. The <user-configured-prefix> can be configured with the `metrics.prefix` configuration setting.

For example, any transaction metric is a database metric. In particular, the `neo4j.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
- Causal clustering metrics

## General-purpose metrics

*Table 468. Bolt metrics*

| Name | Description |
| --- | --- |
| `<prefix>.bolt.sessions_started` | The total number of Bolt sessions created by users since startup. This includes both succeeded and failed sessions (deprecated, use connections_opened instead). Useful for monitoring load via the Bolt drivers in combination with other metrics. (counter) |
| `<prefix>.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) |
| `<prefix>.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>.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>.bolt.connections_idle` | The total number of Bolt connections that are not currently executing Cypher or returning results. (gauge) |
| `<prefix>.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) |

| Name | Description |
| --- | --- |
| `<prefix>.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 being observed between between `bolt.messages_received` and `bolt.messages_started` could indicate the server is running at capacity. (counter) |
| `<prefix>.bolt.messages_done` | The total number of Bolt messages that have completed processing whether successfully or unsuccessfully. Useful for tracking overall load. (counter) |
| `<prefix>.bolt.messages_failed` | The total number of messages that have failed while processing. A high number of failures may indicate an issue with server and further investigation of the logs is recommended. (counter) |
| `<prefix>.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 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>.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) |

*Table 469. Database checkpointing metrics*

| Name | Description |
| --- | --- |
| `<prefix>.check_point.events` | The total number of check point events executed so far. (counter) |
| `<prefix>.check_point.total_time` | The total time, in milliseconds, spent in check pointing so far. (counter) |
| `<prefix>.check_point.duration` | The duration, in milliseconds, of the last check point 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) |

*Table 470. 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) |

*Table 471. 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) |

*Table 472. Database neo4j pools metrics*

| Name | Description |
| --- | --- |
| <prefix>.pool.<pool>.<database>.used_heap | Used or reserved heap memory in bytes. (gauge) |
| <prefix>.pool.<pool>.<database>.used_native | Used or reserved native memory in bytes. (gauge) |
| <prefix>.pool.<pool>.<database>.total_used | Sum total used heap and native memory in bytes. (gauge) |
| <prefix>.pool.<pool>.<database>.total_size | Sum total size of capacity of the heap and/or native memory pool. (gauge) |
| <prefix>.pool.<pool>.<database>.free | Available unused memory in the pool, in bytes. (gauge) |

*Table 473. 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 which failed previously but have recovered. (counter) |

*Table 474. Database data metrics*

| Name | Description |
| --- | --- |
| <prefix>.ids_in_use.relationship_type | The total number of different relationship types stored in the database. Informational, not an indication of any issue. Spikes or large increases indicate large data loads, which could correspond with some behavior you are investigating. (gauge) |

| Name | Description |
| --- | --- |
| <prefix>.ids_in_use.property | The total number of different property names used in the database. Informational, not an indication of any issue. Spikes or large increases indicate large data loads, which could correspond with some behavior you are investigating. (gauge) |
| <prefix>.ids_in_use.relationship | The total number of relationships stored in the database. Informational, not an indication of any issue. Spikes or large increases indicate large data loads, which could correspond with some behavior you are investigating. (gauge) |
| <prefix>.ids_in_use.node | The total number of nodes stored in the database. Informational, not an indication of any issue. Spikes or large increases indicate large data loads, which could correspond with some behavior you are investigating. (gauge) |

*Table 475. 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) |
| <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 capacity of the heap and/or native memory pool. (gauge) |
| <prefix>.dbms.pool.<pool>.free | Available unused memory in the pool, in bytes. (gauge) |

*Table 476. 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) |

| Name | Description |
| --- | --- |
| <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.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.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. |
| <prefix>.page_cache.throttled.times | The total number of times page cache flush IO limiter was throttled during ongoing IO operations. |
| <prefix>.page_cache.throttled.millis | The total number of millis page cache flush IO limiter was throttled during ongoing IO operations. |

*Table 477. Query execution metrics*

| Name | Description |
| --- | --- |
| <prefix>.db.query.execution.success | Count of successful queries executed. (counter) |
| <prefix>.db.query.execution.failure | Count of failed queries executed. (counter) |
| <prefix>.db.query.execution.latency.millis | Execution time in milliseconds of queries executed successfully. (histogram) |

*Table 478. 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) |

*Table 479. 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 transaction log. (counter) |
| <prefix>.log.flushes | The total number of transaction log flushes. (counter) |
| <prefix>.log.append_batch_size | The size of the last transaction append batch. (gauge) |

*Table 480. 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) |

| Name | Description |
| --- | --- |
| <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) |
| <prefix>.transaction.last_committed_tx_id | The ID of the last committed transaction. Track this for each instance. (Cluster) Track this for each Core cluster member, and each Read Replica. 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) |

*Table 481. 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) |

# Chapter 16. Metrics specific to Causal Clustering

*Table 482. CatchUp Metrics*

| Name | Description |
| --- | --- |
| `<prefix>.causal_clustering.catchup.tx_pull_requests_received` | TX pull requests received from read replicas. (counter) |

*Table 483. Discovery core metrics*

| Name | Description |
| --- | --- |
| `<prefix>.causal_clustering.core.discovery.replicated_data` | Size of replicated data structures. (gauge) |
| `<prefix>.causal_clustering.core.discovery.cluster.members` | Discovery cluster member size. (gauge) |
| `<prefix>.causal_clustering.core.discovery.cluster.unreachable` | Discovery cluster unreachable size. (gauge) |
| `<prefix>.causal_clustering.core.discovery.cluster.converged` | Discovery cluster convergence. (gauge) |

*Table 484. 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 less 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 bigger than or equal to the append 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 bigger 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) |

| Name | Description |
| --- | --- |
| `<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) |
| `<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) |

| Name | Description |
| --- | --- |
| <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) |

*Table 485. 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) |
| <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) |

# Chapter 17. 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.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.

*Table 486. 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) |

*Table 487. 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 potentially 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) |

*Table 488. 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 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 initial and max heap size, or if you are underutilizing, you should decrease the initial and max heap sizes. (gauge) |

*Table 489. 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) |

*Table 490. JVM memory pools metrics.*

| Name | Description |
|---|---|
| `<prefix>.vm.memory.pool.<pool>` | Estimated amount of memory in bytes used by the pool. (gauge) |

*Table 491. JVM pause time metrics.*

| Name | Description |
|---|---|
| `<prefix>.vm.pause_time` | Accumulated detected VM pause time. (gauge) |

*Table 492. JVM threads metrics.*

| Name | Description |
|---|---|
| `<prefix>.vm.thread.count` | Estimated number of active threads in the current thread group. (gauge) |
| `<prefix>.vm.thread.total` | The total number of live threads including daemon and non-daemon threads. (gauge) |

# 17.1. Manage queries

## 17.1.1. 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.

## 17.1.2. List all active locks for a query

An administrator is able to view all active locks held by the transaction executing the query with the `queryId`.

Syntax:

```
CALL dbms.listActiveLocks(queryId)
```

Returns:

| 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 98. Viewing active locks for a query*

The following example shows the active locks held by transaction executing query with id `query-614`

```
CALL dbms.listActiveLocks( "query-614" )
```

| "mode" | "resourceType" | "resourceId" |
| --- | --- | --- |
| "SHARED" | "SCHEMA" | 0 |

1 row

The following example shows the active locks for all currently executing queries by yielding the `queryId` from `dbms.listQueries` procedure

```
CALL dbms.listQueries() YIELD queryId, query, database
CALL dbms.listActiveLocks( queryId ) YIELD resourceType, resourceId, mode
RETURN queryId, query, resourceType, resourceId, mode, database
```

| "queryId" | "query" | "resourceType" | "resourceId" | "mode" | "database" |
| --- | --- | --- | --- | --- | --- |
| "query-614" | "match (n), (m), (o), (p), (q) return count(*)" | "SCHEMA" | 0 | "SHARED" | "myDb" |
| "query-684" | "CALL dbms.listQueries() YIELD ..." | "SCHEMA" | 0 | "SHARED" | "myOtherDb" |

2 rows

## 17.1.3. Terminate queries

The procedures for terminating queries, `dbms.killQueries(queryIds)` and `dbms.killQuery(queryId)`, are replaced by the command for terminating transactions, `TERMINATE TRANSACTIONS transactionIds`. These commands use the transactionId instead of the queryId, which 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. |
| `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.

# 17.2. Manage transactions

## 17.2.1. Configure transaction timeout

It is recommended to configure Neo4j to terminate transactions whose execution time has exceeded the configured timeout.

- Set `dbms.transaction.timeout` to some positive time interval value (e.g., `10s`) denoting the default transaction timeout. Setting `dbms.transaction.timeout` to `0` — which is the default value — disables the feature.
- You can also set this dynamically on each instance (Read Replicas only if required) using the procedure `dbms.setConfigValue('dbms.transaction.timeout','10s')`.

*Example 99. Configure transaction timeout*

Set the timeout to ten seconds.

```
dbms.transaction.timeout=10s
```

Configuring transaction timeout has no effect on transactions executed with custom timeouts (e.g., via the Java API or Neo4j Drivers), as the custom timeout overrides the value set for `dbms.transaction.timeout`. Note that the timeout value can only be overridden to a value that is smaller than that configured by `dbms.transaction.timeout`.

The *transaction timeout* feature is also known as the *transaction guard*.

### 17.2.2. 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 set `dbms.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 `dbms.lock.acquisition.timeout` to `0` — which is the default value — disables the lock acquisition timeout.

This feature cannot be set dynamically.

*Example 100. Configure lock acquisition timeout*

Set the timeout to ten seconds.

```
dbms.lock.acquisition.timeout=10s
```

### 17.2.3. List all running transactions

The procedure for viewing transactions, `dbms.listTransactions()`, is replaced by the command for showing transactions, `SHOW TRANSACTIONS`. Both the command and procedure returns the transactions that are currently executing within the instance.

The `SHOW TRANSACTION` privilege determines what transactions are returned by the command. However, the current user can always view all of their own currently executing transactions.

**Syntax:**

```
SHOW TRANSACTIONS
```

For more information on this command, see the Cypher manual → `SHOW TRANSACTIONS` command.

## 17.3. Manage connections

### 17.3.1. 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 493. 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. |
| `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 494. 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 101. 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
```

## 17.3.2. 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. |

| Name | Type | Description |
| --- | --- | --- |
| `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 102. 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

## 17.3.3. 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 103. 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

# 17.4. 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.

## 17.4.1. Listing active background jobs

An administrator can list background jobs active on an instance:

**Syntax:**

```
CALL dbms.scheduler.jobs()
```

**Returns:**

| Name | Type |
|---|---|
| `jobId`<br><br>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`<br><br>A job is a member of a job group. For example, `INDEX_POPULATION`, `LOG_ROTATION` or `RAFT_SERVER`. | String |
| `submitted`<br><br>A timestamp for when the job was submitted, in ISO-8601 format. | String |
| `database`<br><br>Jobs can have either a database or a DBMS scope:<br>• For database, this column will display the name of the database.<br>• For DBMS, this column will be blank. | String |
| `submitter`<br><br>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 |

| Name | Type |
| --- | --- |
| `description`<br><br>A short description of a job that, unlike `currentStateDescription`, does not change during the running of the job. | String |
| `type`<br><br>Type of the job. The values can be `IMMEDIATE`, `DELAYED` or `PERIODIC`. | String |
| `scheduledAt`<br><br>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`<br><br>A period of a `PERIODIC` job, in format `hh:mm:ss.sss`. | String |
| `state`<br><br>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`<br><br>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 |

## 17.4.2. Listing failed job executions

An administrator can list job executions failed on an instance:

**Syntax:**

```
CALL dbms.scheduler.failedJobs()
```

**Returns:**

| Name | Type |
| --- | --- |
| `jobId`<br><br>ID of the failed job. | String |
| `group`<br><br>A job is a member of a job group. For example, `INDEX_POPULATION`, `LOG_ROTATION` or `RAFT_SERVER`. | String |
| `database`<br><br>Jobs can have either a database or a DBMS scope:<br><br>• For database, this column will display the name of the database.<br>• For DBMS, this column will be blank. | String |

| Name | Type |
| --- | --- |
| `submitter`<br><br>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`<br><br>A short description of a job that, unlike `currentStateDescription`, does not change during the running of the job. | String |
| `type`<br><br>Type of the job. The values can be `IMMEDIATE`, `DELAYED` or `PERIODIC`. | String |
| `submitted`<br><br>A timestamp for when the job was submitted, in ISO-8601 format. | String |
| `executionStart`<br><br>A timestamp for when the failed execution started, in ISO-8601 format. | String |
| `failureTime`<br><br>A timestamp for when the execution failed, in ISO-8601 format. | String |
| `failureDescription`<br><br>A short description of the failure. If the failure description is insufficient, more information can be found in logs. | String |

# 17.5. Monitor a Causal Cluster

In addition to specific metrics as described in previous sections, Neo4j Causal Clusters provide an infrastructure that operators will wish to monitor. The procedures can be used to inspect the cluster state and to understand its current condition and topology. Additionally, there are HTTP endpoints for checking health and status.

This section describes the following:

- Procedures for monitoring a Causal Cluster
  - Find out the role of a cluster member
  - Gain an overview over the instances in the cluster
  - Get routing recommendations
- Endpoints for status information
  - Adjusting security settings for Causal Clustering endpoints
  - Unified endpoints

## 17.5.1. Procedures for monitoring a Causal Cluster

A number of procedures are available that provide information about a cluster.

## Find out the role of a cluster member

The procedure `dbms.cluster.role(databaseName)` can be called on every instance in a Causal Cluster to return the role of the instance. Each instance holds multiple databases and participates in multiple independent Raft groups. The role returned by the procedure is for the database denoted by the `databaseName` parameter.

**Syntax:**

```
CALL dbms.cluster.role(databaseName)
```

**Arguments:**

| Name | Type | Description |
|---|---|---|
| `databaseName` | String | The name of the database to get the cluster role for. |

**Returns:**

| Name | Type | Description |
|---|---|---|
| `role` | String | This is the role of the current instance, which can be `LEADER`, `FOLLOWER`, or `READ_REPLICA`. |

**Considerations:**

- While this procedure is useful in and of itself, it serves as basis for more powerful monitoring procedures.

*Example 104. Check the role of this instance*

The following example shows how to find out the role of the current instance for database `neo4j`, which in this case is `FOLLOWER`.

```
CALL dbms.cluster.role("neo4j")
```

| role |
|---|
| FOLLOWER |

## Gain an overview over the instances in the cluster

The procedure `dbms.cluster.overview()` provides an overview of cluster topology by returning details on all the instances in the cluster.

Syntax:

```
CALL dbms.cluster.overview()
```

Returns:

| Name | Type | Description |
| --- | --- | --- |
| `id` | String | This is id of the instance. |
| `addresses` | List | This is a list of all the addresses for the instance. |
| `groups` | List | This is a list of all the server groups which an instance is part of. |
| `databases` | Map | This is a map of all databases with corresponding roles which the instance is hosting. The keys in the map are database names. The values are roles of this instance in the corresponding Raft groups, which can be `LEADER`, `FOLLOWER`, or `READ_REPLICA`. |

*Example 105. Get an overview of the cluster*

The following example shows how to explore the cluster topology.

```
CALL dbms.cluster.overview()
```

| id | addresses | groups | databases |
| --- | --- | --- | --- |
| 08eb9305-53b9-4394-9237-0f0d63bb05d5 | [bolt://neo20:7687, http://neo20:7474, https://neo20:7473] | [] | {system: LEADER, neo4j: FOLLOWER} |
| cb0c729d-233c-452f-8f06-f2553e08f149 | [bolt://neo21:7687, http://neo21:7474, https://neo21:7473] | [] | {system: FOLLOWER, neo4j: FOLLOWER} |
| ded9eed2-dd3a-4574-bc08-6a569f91ec5c | [bolt://neo22:7687, http://neo22:7474, https://neo22:7473] | [] | {system: FOLLOWER, neo4j: LEADER} |
| 00000000-0000-0000-0000-000000000000 | [bolt://neo34:7687, http://neo34:7474, https://neo34:7473] | [] | {system: READ_REPLICA, neo4j: READ_REPLICA} |

| id | addresses | groups | databases |
| --- | --- | --- | --- |
| 00000000-0000-0000-0000-000000000000 | [bolt://neo28:7687, http://neo28:7474, https://neo28:7473] | [] | {system: READ_REPLICA, neo4j: READ_REPLICA} |
| 00000000-0000-0000-0000-000000000000 | [bolt://neo31:7687, http://neo31:7474, https://neo31:7473] | [] | {system: READ_REPLICA, neo4j: READ_REPLICA} |

## Get routing recommendations

From the application point of view it is not interesting to know about the role a member plays in the cluster. Instead, the application needs to know which instance can provide the wanted service. The procedure `dbms.routing.getRoutingTable(routingContext, databaseName)` provides this information.

**Syntax:**

```
CALL dbms.routing.getRoutingTable(routingContext, databaseName)
```

**Arguments:**

| Name | Type | Description |
| --- | --- | --- |
| `routingContext` | Map | The routing context used for multi-data center deployments. It should be used in combination with multi-data center load balancing. |
| `databaseName` | String | The name of the database to get the routing table for. |

*Example 106. Get routing recommendations*

The following example shows how discover which instances in the cluster can provide which services for database `neo4j`.

```
CALL dbms.routing.getRoutingTable({}, "neo4j")
```

The procedure returns a map between a particular service, `READ`, `WRITE` and `ROUTE`, and the addresses of instances that provide this service. It also returns a Time To Live (TTL) in seconds as a suggestion on how long the client could cache the response.

The result is not primarily intended for human consumption. Expanding it this is what it looks like.

```
{
    "ttl": 300,
    "servers": [
        {
            "addresses": ["neo20:7687"],
```

```
            "role": "WRITE"
        },
        {
            "addresses": ["neo21:7687", "neo22:7687", "neo34:7687", "neo28:7687", "neo31:7687"],
            "role": "READ"
        },
        {
            "addresses": ["neo20:7687", "neo21:7687", "neo22:7687"],
            "role": "ROUTE"
        }
    ]
}
```

## 17.5.2. Monitor cluster endpoints for status information

A Causal Cluster exposes some HTTP endpoints which can be used to monitor the health of the cluster. In this section we will describe these endpoints and explain their semantics.

### Adjusting security settings for Causal Clustering endpoints

If authentication and authorization is enabled in Neo4j, the Causal Clustering status endpoints will 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 Causal Clustering status endpoints by setting `dbms.security.causal_clustering_status_auth_enabled=false` in *neo4j.conf*.

### Unified endpoints

A unified set of endpoints exist, both on Core Servers and on Read Replicas, 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` will have 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
```

*Table 495. Unified HTTP endpoint responses*

| Endpoint | Instance state | Returned code | Body text |
|---|---|---|---|
| `/db/<databasename>/cluster/writable` | Leader | `200 OK` | `true` |
| | Follower | `404 Not Found` | `false` |
| | Read Replica | `404 Not Found` | `false` |
| `/db/<databasename>/cluster/read-only` | Leader | `404 Not Found` | `false` |
| | Follower | `200 OK` | `true` |
| | Read Replica | `200 OK` | `true` |
| `/db/<databasename>/cluster/available` | Leader | `200 OK` | `true` |
| | Follower | `200 OK` | `true` |
| | Read Replica | `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. |
| | Read Replica | `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. |
| | Read Replica | `200 OK` | JSON - See Status endpoint for details. |

*Example 107. Use a Causal Clustering monitoring endpoint*

From the command line, a common way to ask those endpoints is to use `curl`. With no arguments, `curl` will do an HTTP `GET` on the URI provided and will output the body text, if any. If you also want to get the response code, just add the `-v` flag for verbose output. Here are some examples:

- Requesting `writable` endpoint on a Core Server 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 → Upgrade a Causal Cluster.

Typically, you will want to have some guarantee that a Core is safe to shutdown for each database before removing it from a cluster. Counter intuitively, a core being safe to shutdown means that a majority of the **other** cores 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 Causal Clusters to provide highly available transactions. When using multiple databases, each database implements Raft independently. Therefore, details such as `leader` and `raftCommandsPerSecond` are database specific.

*Example 108. 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 496. Status endpoint descriptions*

| Field | Type | Optional | Example | Description |
|---|---|---|---|---|
| `core` | boolean | no | `true` | Used to distinguish between Core Servers and Read Replicas. |
| `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. |

| Field | Type | Optional | Example | Description |
| --- | --- | --- | --- | --- |
| `participatingInRaftGroup` | boolean | no | `false` | A participating member is able to vote. A Core 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 Core. |
| `isHealthy` | boolean | no | `true` | Indicates that the local database on this cluster member has not encountered a critical error that can block database operation.<br><br>**Caution:**<br><br>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 Core and Read Replica. |
| `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 Read Replicas, and hence is not included. |
| `raftCommandsPerSecond` | number | yes | `124` | An estimate of the average Raft state machine throughput over a sampling windown configurable via `causal_clustering.status_throughput_window` setting. |

After an instance has been switched on, you can access the status endpoint 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 core members and compare the result. The following table explains how results can be compared.

*Table 497. Measured values, accessed via the status endpoint*

| Name of check | Method of calculation | Description |
| --- | --- | --- |
| `allServersAreHealthy` | Every Core's status endpoint indicates `isHealthy == true`. | We want to make sure the data across the entire cluster is healthy. Whenever any Cores are false that indicates a larger problem. |
| `allVotingSetsAreEqual` | For any 2 Cores (A and B), status endpoint A's `votingMembers` == status endpoint B's `votingMembers`. | When the voting begins, all the Cores are equal to each other, and you know all members agree on membership. |
| `allVotingSetsContainAtLeastTargetCluster` | For all Cores (S), excluding Core 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 will not be perfect and it may make sense to switch off a different Core to the one we originally wanted to switch off. If you run this check for all Cores, the ones that match this condition can be switched off (providing other conditions are also met). |
| `hasOneLeader` | For any 2 Cores (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 Core A with `lastAppliedRaftIndex` = `min`, and Core B with `lastAppliedRaftIndex` = `max`, `B.lastAppliedRaftIndex-A.lastAppliedRaftIndex<raftIndexLagThreshold`. | If there is a large difference in the applied indexes between Cores, then it could be dangerous to switch off a Core. |



**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, you need to assess whether a Core is safe to shutdown for **all** databases. To avoid having to issue a separate request to each

`/db/<databasename>/cluster/status` endpoint, you can 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 109. 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
    }
  }
]
```


# 17.6. Monitor individual database states

In addition to the system-wide metrics and logs described in previous sections, operators may wish to monitor the state of individual databases being hosted within a Neo4j instance. The `SHOW DATABASES` command may be used for this purpose.

## 17.6.1. Listing Databases

First ensure that you are executing queries against the `system` database, either by running the command `:use system` (if using the Cypher shell or Neo4j Browser) or by creating a session against the `system` database using a Neo4j driver. Subsequently, run the `SHOW DATABASES` command.

**Syntax:**

```
SHOW DATABASES
```

Returns:

| Name | Type | Description |
|---|---|---|
| `name` | String | The human-readable name of the database. |
| `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 Neo4j instance hosting the database. |
| `role` | String | The cluster role which the Neo4j instance fulfils for this database. |
| `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 Neo4j instance. |
| `error` | String | Error encountered by the Neo4j instance 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. |

*Example 110. Listing databases in standalone Neo4j*

When executing `SHOW DATABASES` against a standalone instance of Neo4j, you should see output like the following:

| `name` | `aliases` | `access` | `address` | `role` | `requestedStatus` | `currentStatus` | `error` | `default` | `home` |
|---|---|---|---|---|---|---|---|---|---|
| "neo4j" | [] | "read-write" | "localhost:7687" | "standalone" | "online" | "online" | "" | true | true |
| "system" | [] | "read-write" | "localhost:7687" | "standalone" | "online" | "online" | "" | false | false |

Note that the `role` and `address` columns are primarily intended to distinguish between the states of a given database, across multiple Neo4j instances deployed in a Causal Cluster. In a standalone deployment where you have a single Neo4j instance, your `address` field should be the same for every database, and your `role` field should always be "standalone".

If an error occurred whilst creating (or stopping, dropping etc.) a database, you should see output like the following:

| name | aliases | access | address | role | requestedStatus | currentStatus | error | default | home |
|---|---|---|---|---|---|---|---|---|---|
| "neo4j" | [] | "read-write" | "localhost:7687" | "standalone" | "online" | "online" | "" | true | true |
| "system" | [] | "read-write" | "localhost:7687" | "standalone" | "online" | "online" | "" | false | false |
| "foo" | [] | "read-write" | "localhost:7687" | "standalone" | "online" | "offline" | "An error occurred! Unable to start database ..." | false | false |

Note that for failed databases, the `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 Neo4j instance copying store files from another.

The possible statuses are "initial", "online", "offline", "store copying", and "unknown".

*Example 111. Listing databases in a Neo4j Causal Cluster*

When running `SHOW DATABASES` against a Neo4j Causal Cluster you might see output like the following:

| name | aliases | access | address | role | requestedStatus | currentStatus | error | default | home |
|---|---|---|---|---|---|---|---|---|---|
| "neo4j" | [] | "read-write" | "localhost:20031" | "follower" | "online" | "online" | "" | true | true |
| "neo4j" | [] | "read-write" | "localhost:20010" | "follower" | "online" | "online" | "" | true | true |
| "neo4j" | [] | "read-write" | "localhost:20005" | "leader" | "online" | "online" | "" | true | true |
| "neo4j" | [] | "read-write" | "localhost:20034" | "read_replica" | "online" | "online" | "" | true | true |

| name | aliases | access | address | role | requestedStatus | currentStatus | error | default | home |
|---|---|---|---|---|---|---|---|---|---|
| "system" | [] | "read-write" | "localhost:20031" | "follower" | "online" | "online" | "" | false | false |
| "system" | [] | "read-write" | "localhost:20010" | "follower" | "online" | "online" | "" | false | false |
| "system" | [] | "read-write" | "localhost:20005" | "leader" | "online" | "online" | "" | false | false |
| "system" | [] | "read-write" | "localhost:20034" | "read_replica" | "online" | "online" | "" | false | false |
| "foo" | [] | "read-write" | "localhost:20031" | "leader" | "online" | "online" | "" | false | false |
| "foo" | [] | "read-write" | "localhost:20010" | "follower" | "online" | "online" | "" | false | false |
| "foo" | [] | "read-write" | "localhost:20005" | "follower" | "online" | "online" | "" | false | false |
| "foo" | [] | "read-write" | "localhost:20034" | "read_replica" | "online" | "online" | "" | false | false |

Note that `SHOW DATABASES` does **not** return 1 row per database. Instead, it returns 1 row per database, per Neo4j instance in the cluster. Therefore, if you have a 4-instance cluster, hosting 3 databases, you will have 12 rows.

If an error occurred whilst creating (or stopping, dropping etc.) a database, you should see output like the following:

| name | aliases | access | address | role | requestedStatus | currentStatus | error | default | home |
|---|---|---|---|---|---|---|---|---|---|
| "neo4j" | [] | "read-write" | "localhost:20031" | "follower" | "online" | "online" | "" | true | true |
| "neo4j" | [] | "read-write" | "localhost:20010" | "follower" | "online" | "online" | "" | true | true |
| "neo4j" | [] | "read-write" | "localhost:20005" | "leader" | "online" | "online" | "" | true | true |
| "neo4j" | [] | "read-write" | "localhost:20034" | "read_replica" | "online" | "online" | "" | true | true |
| "system" | [] | "read-write" | "localhost:20031" | "follower" | "online" | "online" | "" | false | false |

| name | aliases | access | address | role | requestedStatus | currentStatus | error | default | home |
|---|---|---|---|---|---|---|---|---|---|
| "system" | [] | "read-write" | "localhost:20010" | "follower" | "online" | "online" | "" | false | false |
| "system" | [] | "read-write" | "localhost:20005" | "leader" | "online" | "online" | "" | false | false |
| "system" | [] | "read-write" | "localhost:20034" | "read_replica" | "online" | "online" | "" | false | false |
| "foo" | [] | "read-write" | "localhost:20031" | "unknown" | "online" | "initial" | "An error occurred! Unable to start database ..." | false | false |
| "foo" | [] | "read-write" | "localhost:20010" | "leader" | "online" | "online" | "" | false | false |
| "foo" | [] | "read-write" | "localhost:20005" | "follower" | "online" | "online" | "" | false | false |
| "foo" | [] | "read-write" | "localhost:20034" | "unknown" | "online" | "initial" | "An error occurred! Unable to start database ..." | false | false |

Note that different instances may have different roles for each database.

If a database is offline on a particular Neo4j instance, either because it was stopped by an operator or an error has occurred, its cluster `role` is "unknown". This is because the cluster role of a given instance/database combination cannot be assumed in advance. This differs from standalone Neo4j instances, where the role of that instance for each database can always be assumed to be "standalone".

The possible roles are "standalone", "leader", "follower", "read_replica", and "unknown".

## 17.6.2. 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. |
| `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 Neo4j instance hosting the database. |
| `role` | String | The cluster role which the Neo4j instance fulfils for this database. |
| `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 Neo4j instance. |
| `error` | String | Error encountered by Neo4j instance 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. |

*Example 112. Listing statuses for database foo*

When running `SHOW DATABASE foo` in a Neo4j Causal Cluster, you should see output like the following:

| name | aliases | access | address | role | requestedStatus | currentStatus | error | default | home |
|---|---|---|---|---|---|---|---|---|---|
| "foo" | [] | "read-write" | "localhost:20031" | "unknown" | "online" | "initial" | "An error occurred! Unable to start database ..." | false | false |
| "foo" | [] | "read-write" | "localhost:20010" | "leader" | "online" | "online" | "" | false | false |
| "foo" | [] | "read-write" | "localhost:20005" | "follower" | "online" | "online" | "" | false | false |
| "foo" | [] | "read-write" | "localhost:20034" | "unknown" | "online" | "initial" | "An error occurred! Unable to start database ..." | false | false |

# Chapter 18. Tools

This chapter comprises the following topics:

- Neo4j CLI tool — A description of the Neo4j database server CLI tool.
- Neo4j Admin tool — A description of the Neo4j Admin tool.
  - 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 Core Server — How to remove cluster state data from a Neo4j server.
  - Push to Neo4j AuraDB — How to push an existing Neo4j graph to Neo4j AuraDB.
- 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.

## 18.1. Neo4j CLI tool

The Neo4j CLI tool is a tool for managing a Neo4j database server. It is a command-line tool that is installed as part of the product and can be executed with a number of commands. Neo4j CLI tool is located in the *bin* directory.

### 18.1.1. Syntax and commands

#### General synopsis

The `neo4j` tool has the following general synopsis:

```
neo4j <command>
```

#### Available commands

| Command | Description |
|---|---|
| `console` | Start server in console. |
| `start` | Start server as a daemon. |
| `stop` | Stop the server daemon. |
| `restart` | Restart the server daemon. |
| `status` | Get the status of the server. |

| Command | Description |
| --- | --- |
| `version, --version` | Print version information and exit. |
| `help, --help` | Display help information about the specified command.<br><br>For example, running `./bin/neo4j --help start` outputs the usage, description, and all available options of the `start` command:<br><br>`Usage: Neo4j start [--expand-commands] [--verbose]`<br>`Start server as a daemon.`<br>`--expand-commands Allow command expansion in config value evaluation.`<br>`--verbose Prints additional information.` |

> **Note:**
>
> Command expansion can be enabled by adding a customised script to the neo4j.conf file, and then including the `--expand-commands` argument to the Neo4j startup script.
>
> For more information, see Command expansion.

### 18.1.2. Environment variables

Neo4j CLI tool 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. Refer to JVM documentation about the exact format. This variable is incompatible with `HEAP_SIZE` and takes precedence over `HEAP_SIZE`. |

> **Note:**
>
> By default, `dbms.jvm.additional` settings specified in the configuration file are used when invoking `neo4j` CLI commands. If set, `JAVA_OPTS` overrides all relevant settings specified in the configuration file.

## 18.2. Neo4j Admin

### 18.2.1. Introduction

Neo4j Admin is the primary tool for managing a Neo4j DBMS. It is a command-line tool that is installed as

part of the product and can be executed with a number of commands. Some of the commands are described in more detail in separate sections.



Note:

The Neo4j 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.

## 18.2.2. Syntax and commands

### Syntax

Neo4j Admin is located in the *bin* directory and is invoked as:

```
neo4j-admin [-hV] [COMMAND]
```

- `-h`, `--help` — Show the `neo4j-admin` help message and exit.
- `-V`, `--version` — Print the `neo4j-admin` version information and exit.

### Commands

| Functionality area | Command | Description |
| --- | --- | --- |
| General | `help <command>` | Display help information about the specified command. |
| | `check-consistency` | Check the consistency of a database.<br><br>For details, see Consistency checker. |
| | `report` | Produce a zip/tar of the most common information needed for remote assessments.<br><br>For details, see Neo4j Admin report. |
| | `store-info` | Print information about a Neo4j database store.<br><br>For details, see Display store information. |
| | `memrec` | Print Neo4j heap and pagecache memory settings recommendations.<br><br>For details, see Memory recommendations. |
| | `import` | Import from a collection of CSV files or a pre-3.0 database.<br><br>For details, see Import. |
| | `copy` | Copy data from an existing database to a new database.<br><br>For details, see Copy a database store. |
| | `push-to-cloud` | Dump a local offline database, and imports it into a specified Neo4j AuraDB instance. The target location is your Neo4j Aura Bolt URI.<br><br>For details, see Push to cloud. |
| Authentication | `set-default-admin` | Set the default admin user when no roles are present. |
| | `set-initial-password` | Set the initial password of the initial admin user (`neo4j`).<br><br>For details, see Set an initial password. |
| Offline backup and restore<br><br>For details, see Back up an offline database and Restore a database dump. | `dump` | Dump a database into a single-file archive. |
| | `load` | Load a database from an archive created with the `dump` command. |
| Online backup and restore<br><br>For details, see Back up an online database, Prepare a database for restoring, and Restore a database backup. | `backup` | Perform an online backup from a running Neo4j server. |
| | `prepare-restore` | Prepare a backup for restoring by applying the latest transactions pulled at the time of backup but not yet applied to the store. |
| | `restore` | Restore a backed-up database. |

| Functionality area | Command | Description |
| --- | --- | --- |
| Clustering | `unbind` | Remove cluster state data from a stopped Neo4j server.<br><br>For details, see Unbind a Core Server. |
| | `get-server-id` | Display the Server ID of a Neo4j instance.<br><br>If no Server ID is returned, either Neo4j has never started or it has been unbound. Start Neo4j to create a new Server ID and then re-run this command.<br><br>The Server ID can be used in Cypher commands to identify an instance. |

## 18.2.3. Environment variables

## 18.2.4. Environment variables

The Neo4j Admin tool 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. |



**Note:**

By default, `dbms.jvm.additional` settings specified in the configuration file are used when invoking Admin commands. Even though this is a default behavior, it might lead to problems when an Admin command is invoked on the same machine running DBMS. Since the command reuses JVM memory settings configured for the DBMS, the machine might not have enough memory to run both tasks simultaneously. Even if it does, Admin commands require nowhere near as much memory as the DBMS, so this default behavior is wasteful in most cases.

Still, memory consumption might not be the only issue. For instance, if the DBMS JVM is configured to open an administration port for JMX or debugging, Admin commands will try to open the same port and thus conflict with a running DBMS. If the default behavior described above is problematic, Admin commands can be instructed to ignore `dbms.jvm.additional` from the configuration file by simply using `JAVA_OPTS` with any non-empty value, for example: `JAVA_OPTS=-Xmx512m neo4j-admin …`. Setting `JAVA_OPTS` to any non-empty value will make the invoked Admin command completely ignore all

`dbms.jvm.additional` settings in the configuration file.

## 18.2.5. Exit codes

When `neo4j-admin` finishes as expected, it returns an exit code of `0`. A non-zero exit code means something undesired happened during command execution. The non-zero exit code can contain further information about the error, such as the `backup` command's exit codes.

## 18.2.6. Consistency checker

The consistency of a database or a backup can be checked using the `check-consistency` argument to the `neo4j-admin` tool. The `neo4j-admin` tool is located in the `bin` directory. If checking the consistency of a database, note that it has to be stopped first or else the consistency check will result in an error.



It is not recommended to use an NFS to check the consistency of a database or a backup as this slows the process down significantly.

### Syntax

```
neo4j-admin check-consistency ([--database=<database>] | [--backup=<path>])
                              [--verbose] [--additional-config=<path>]
                              [--check-graph=<true/false>]
                              [--check-indexes=<true/false>]
                              [--check-index-structure=<true/false>]
                              [--check-label-scan-store=<true/false>]
                              [--check-property-owners=<true/false>]
                              [--report-dir=<path>]`
```



**Note:**

Please note that the following options have been deprecated:

```
[--check-label-scan-store=<true/false>]
[--check-property-owners=<true/false>]
```

Values for these settings will be ignored.

### Options

| Option | Default | Description |
|---|---|---|
| --database | neo4j | Name of database. |
| --backup | | Path to backup to check consistency of. Cannot be used together with --database. |
| --additional-config | | Configuration file to supply additional configuration in. |

| Option | Default | Description |
| --- | --- | --- |
| --verbose | `false` | Enable verbose output. |
| --report-dir | `.` | Directory to write report file in. |
| --check-graph | `true` | Perform checks between nodes, relationships, properties, types and tokens. |
| --check-indexes | `true` | Perform checks on indexes by comparing content with the store. |
| --check-index-structure | `true` | Perform physical structure check on indexes. No comparison with the store takes place. |
| `--check-label-scan-store` | `true` | This option is deprecated and its value will be ignored. |
| `--check-property-owners` | `false` | This option is deprecated and its value will be ignored. |

### Output

If the consistency checker does not find errors, it will exit cleanly and not produce a report. If the consistency checker finds errors, it will exit with an exit code of `1` and write a report file with a name on 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-dir`.

*Example 113. Run the consistency checker*

Run with the `--database` option to check the consistency of a database. Note that the database must be stopped first.

```
$neo4j-home> bin/neo4j-admin check-consistency --database=neo4j

2019-11-13 12:42:14.479+0000 INFO [o.n.k.i.s.f.RecordFormatSelector] Selected
RecordFormat:StandardV4_0[SF4.0.b] record format from store /data/databases/neo4j
2019-11-13 12:42:14.481+0000 INFO [o.n.k.i.s.f.RecordFormatSelector] Format not configured for store
/data/databases/neo4j. Selected format from the store files: RecordFormat:StandardV4_0[SF4.0.b]
Index structure consistency check
.....................  10%
.....................  20%
.....................  30%
.....................  40%
.....................  50%
.....................  60%
.....................  70%
.....................  80%
.....................  90%
..................... 100%
Full Consistency Check
.....................  10%
.....................  20%
.....................  30%
.....................  40%
.....................  50%
.....................  60%
.....................  70%
.....................  80%
.....................  90%
.Checking node and relationship counts
```

```
....................  10%
....................  20%
....................  30%
....................  40%
....................  50%
....................  60%
....................  70%
....................  80%
....................  90%
.................... 100%
```

Run with the `--backup` option to check the consistency of a backup.

```
bin/neo4j-admin check-consistency --backup backup/neo4j-backup
```



**Note:**

`neo4j-admin check-consistency` cannot be applied to the Fabric virtual database. It must be run directly on the databases that are part of the Fabric setup.

## 18.2.7. Neo4j Admin report

Use the `report` command of `neo4j-admin` to gather information about a Neo4j installation and save it to an archive.

```
neo4j-admin report [--force] [--list] [--verbose] [--database=<database>] [--pid=<pid>] [--to=<path>] [<classifier>…]
```

The intended usage of the report tool is to simplify the support process by collecting the relevant information in a standard way. This tool does not send any information automatically. To share this information with the Neo4j Support organization, you have to send it manually.

Starting from Neo4j version 4.4.31, 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 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.

*Table 498. Options*

| Option | Default | Description |
|---|---|---|
| `--to` | reports/ | Specify to target directory where the report should be written to. |
| `--list` | | List available classifiers. |

| Option | Default | Description |
| --- | --- | --- |
| `--verbose` | | Instruct the tool to print more verbose output. |
| `--force` | | Disable the available disk space check. |
| `--pid` | | Specify process id of a running Neo4j instance. Only applicable when used together with the *Online* classifiers. See the Classifiers table. |
| `--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. |

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 the compression. Therefore, if you are confident that you do have enough disk space, you can disable this check with the option `--force`.

*Table 499. Classifiers*

| Classifier | Online | Description |
| --- | --- | --- |
| `all` | | Include all of the available classifiers. |
| `ccstate` | | Include the current cluster state. |
| `config` | | Include the *neo4j.conf* file. |
| `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`, and `ps`.

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`.



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 114. Invoke `neo4j-admin report` against a running Neo4j instance using the default classifiers*

The following command gathers information about a Neo4j instance using the default classifiers and saves it to the default location:

```
$neo4j-home> bin/neo4j-admin report --pid=47369
```

*Example 115. Invoke `neo4j-admin report` against a running Neo4j instance using all classifiers*

The following command gathers information about a Neo4j instance using all classifiers and saves it to a specified location:

```
$neo4j-home> bin/neo4j-admin report --pid=47369 --to=./report all
```

*Example 116. Invoke `neo4j-admin report` against running Neo4j to gather only logs and thread dumps*

The following command gathers only logs and thread dumps from a running Neo4j instance and saves it to a specified location:

```
$neo4j-home> bin/neo4j-admin report --pid=47369 --to=./report threads logs
```

## 18.2.8. Display store information

The `neo4j-admin store-info` command outputs information about the *store format* for a given *database store*.

1. The store format version.
2. When the store format version was introduced.
3. Whether the store format needs to be migrated to a newer version.

The store format can be set with the `dbms.record_format` configuration setting.

The store formats are:

- `aligned`
- `standard`
- `high_limit` Enterprise Edition

## Syntax

The `neo4j-admin store-info` command is located in the *bin* directory. It is invoked against an **offline** database store or a backup as follows:

```
neo4j-admin store-info [--all] [--structured] [--verbose] <path>
```

`<path>` — Path to database store, or databases directory if `--all` option is used.

## Options

*Table 500. Options*

| Name | Description |
| --- | --- |
| `--verbose` | Enable verbose output. |
| `--structured` | Return result structured as JSON. |
| `--all` | Return store format info for all databases at provided path. |

## Examples

*Example 117. Invoke `neo4j-admin store-info` against a database store*

```
bin/neo4j-admin store-info data/databases/mygraph.db
```

Output:

```
Store format version:         SF4.0.0
Store format introduced in:   4.0.0
```

*Example 118. Invoke `neo4j-admin store-info` against a database backup*

You can run the `store-info` command to see if the store format of the backup that you want to restore, is compatible with your running Neo4j instance. For example, if you want to restore the database backup */tmp/3518/mygraph.db* into a 4.x Neo4j instance:

```
bin/neo4j-admin store-info /tmp/3518/mygraph.db
```

Output:

```
Store format version:         vE.H.4
Store format introduced in:   3.4.0
Store format superseded in:   4.0.0
```

The store format version reveals that the database is configured to use the `high_limit`, see `dbms.record_format`.

The store format version was introduced in `Neo4j 3.4.0`.

The store format of the current instance is 4.0.0, which means that a format migration must be performed if you want to restore this backup into the current instance.



Note:

For more information on how to migrate a single database, see Upgrade and Migration Guide → Tutorial: Back up and copy a database in a standalone instance.

*Example 119. Invoke `neo4j-admin store-info` against a root containing several databases*

The command can also be invoked against a root directory containing several databases, as follows:

`neo4j-admin store-info <path> --all`

```
bin/neo4j-admin store-info data/databases --all
```

Output:

```
Database name:                foo
Database in use:              false
Store format version:         SF4.0.0
Store format introduced in:   4.0.0
Last committed transaction id:2
Store needs recovery:         true

Database name:                bar
Database in use:              true
```



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 120. Invoke `neo4j-admin store-info` against a database and output JSON*

If you are parsing the results of this command you may use the `--structured` option to receive the output as JSON. All the same fields are included and all values are strings.

```
bin/neo4j-admin store-info data/databases/foo --structured
```

Output:

```
{"databaseName":"foo",
"inUse":"false",
"storeFormat”:"SF4.0.0",
"storeFormatIntroduced”:"4.0.0",
"lastCommittedTransaction":"2",
"recoveryRequired":"true"}
```

## Store formats and entity limits

The following tables show the format and Neo4j version compatibility and the limits of the different store formats:

### Aligned format

*Table 501. Aligned format and Neo4j version compatibility*

| Name | Store format version | Neo4j version |
|---|---|---|
| `ALIGNED_V4_3` | `AF4.3.0` | `4.3.0` |
| `ALIGNED_V4_1` | `AF4.1.a` | `4.1.0` |

*Table 502. 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

*Table 503. Standard format and Neo4j version compatibility*

| Name | Store format version | Neo4j version |
|---|---|---|
| `STANDARD_V4_3` | `SF4.3.0` | `4.3.0` |
| `STANDARD_V4_0` | `SF4.0.0` | `4.0.0` |
| `STANDARD_V3_4` | `v0.A.9` | `3.4.0` |

*Table 504. 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) |

| Name | Limit |
| --- | --- |
| Labels | `2^31` (2 147 483 648) |
| Relationship types | `2^16` (65 536) |
| Relationship groups | `2^35` (34 359 738 368) |

### High_limit format

*Table 505. High_limit format and Neo4j version compatibility*

| Name | Store format version | Neo4j version |
| --- | --- | --- |
| `HIGH_LIMIT_V4_3_0` | `HL4.3.0` | `4.3.0` |
| `HIGH_LIMIT_V4_0_0` | `HL4.0.0` | `4.0.0` |
| `HIGH_LIMIT_V3_4_0` | `vE.H.4` | `3.4.0` |
| `HIGH_LIMIT_V3_2_0` | `vE.H.3` | `3.2.0` |
| `HIGH_LIMIT_V3_1_0` | `vE.H.2` | `3.1.0` |
| `HIGH_LIMIT_V3_0_6` | `vE.H.0b` | `3.0.6` |
| `HIGH_LIMIT_V3_0_0` | `vE.H.0` | `3.0.0` |

*Table 506. 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) |

## 18.2.9. Memory recommendations

Use the `memrec` command of `neo4j-admin` to get an initial recommendation on how to configure memory parameters for Neo4j:

```
neo4j-admin memrec --memory=<memory dedicated to Neo4j>, --verbose, --docker
```

The command gives heuristic memory setting recommendations for the Neo4j JVM heap and pagecache. The heuristic is based on the total memory of the system the command is running on, or on the amount of memory specified with 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 default output is formatted as such that it can be copy-pasted into *neo4j.conf*. 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

| Option | Default | Description |
| --- | --- | --- |
| `--memory=<size>` | The memory capacity of the machine. | The amount of memory to allocate to Neo4j. Valid units are: `k`, `K`, `m`, `M`, `g`, `G`. |
| `--verbose` | | Enable verbose output. |
| `--docker` | | Enable output formatted as environmental variables that can be passed to a Neo4j docker container. |

### Considerations

The `neo4j-admin memrec` command calculates a valid starting point for 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 121. Use the `memrec` command of `neo4j-admin`*

The following example illustrates how `neo4j-admin memrec` provides a recommendation on how to use 16g of memory:

```
$neo4j-home> bin/neo4j-admin memrec --memory=16g

...
...
...
# Based on the above, the following memory settings are recommended:
dbms.memory.heap.initial_size=5g
dbms.memory.heap.max_size=5g
dbms.memory.pagecache.size=7g
```

For an example on how to use the `neo4j-admin memrec` command, see Inspect the memory settings of all databases in a DBMS.

## 18.2.10. Import

There are two ways to import data from CSV files into Neo4j: via `neo4j-admin import` or `LOAD CSV`.

With the `neo4j-admin import` command, you can do batch imports of large amounts of data into a *previously unused database* database from CSV files. The command can be performed only **once** per database. By default, this database is set to `neo4j`, but you can use the `--database=<database>` option to import your data into a different database.

> **Note:**
>
> The user running `neo4j-admin import` must have `WRITE` capabilities into `dbms.directories.data` and `dbms.directories.log`.

With `LOAD CSV`, you can 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.

However, using the `import` command of `neo4j-admin` is generally faster since it is run against a stopped and empty database. This section describes the `neo4j-admin import` option.

> **Tip:**
>
> For information on `LOAD CSV`, see the Cypher Manual → `LOAD CSV`.
> For in-depth examples of using the command `neo4j-admin import`, refer to the Tutorials → Neo4j Admin import.
> To create a cluster based on imported data, see Seed a cluster using the import tool.

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 will not be 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 afterwards (see Cypher Manual → Indexes).

## Syntax

The syntax for importing a set of CSV files is:

```
neo4j-admin import [--expand-commands]
                   [--verbose]
                   [--auto-skip-subsequent-headers[=<true/false>]]
                   [--cache-on-heap[=<true/false>]]
```

```
[--force[=<true/false>]]
[--high-io[=<true/false>]]
[--ignore-empty-strings[=<true/false>]]
[--ignore-extra-columns[=<true/false>]]
[--legacy-style-quoting[=<true/false>]]
[--multiline-fields[=<true/false>]]
[--normalize-types[=<true/false>]]
[--skip-bad-entries-logging[=<true/false>]]
[--skip-bad-relationships[=<true/false>]]
[--skip-duplicate-nodes[=<true/false>]]
[--trim-strings[=<true/false>]]
[--additional-config=<path>]
[--array-delimiter=<char>]
[--bad-tolerance=<num>]
[--database=<database>]
[--delimiter=<char>]
[--id-type=<STRING|INTEGER|ACTUAL>]
[--input-encoding=<character-set>]
[--max-memory=<size>]
[--processors=<num>]
[--quote=<char>]
[--read-buffer-size=<size>]
[--report-file=<path>]
--nodes=[<label>[:<label>]...=]<files>...
[--nodes=[<label>[:<label>]...=]<files>...]...
[--relationships=[<type>=]<files>...]...
```

*Example 122. 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:

```
neo4j_home$ bin/neo4j-admin import --nodes import/movies_header.csv,import/movies.csv \
--nodes import/actors_header.csv,import/actors.csv \
--relationships import/roles_header.csv,import/roles.csv
```

*Example 123. Import data from CSV files using regular expression*

Assume that you want to include a header and then multiple files that matches 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:

```
neo4j_home$ bin/neo4j-admin import --nodes import/node_header.csv,import/node_data_\d+\.csv
```

*Example 124. 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:

```
neo4j_home$ bin/neo4j-admin import --nodes import/node_header.csv,'import/node_data_\d{1,5}.csv'
```



**Note:**

If importing to a database that has not explicitly been created prior to the import, it must be created subsequently in order to be used.

## Options

*Table 507. `neo4j-admin import` options*

| name |
| --- |
| `--expand-commands` |
| `--verbose` |
| `--auto-skip-subsequent-headers` |
| `--cache-on-heap` |
| `--force` |
| `--high-io` |
| `--ignore-empty-strings` |
| `--ignore-extra-columns` |
| `--legacy-style-quoting` |
| `--multiline-fields` |
| `--normalize-types` |
| `--skip-bad-entries-logging` |
| `--skip-bad-relationships` |
| `--skip-duplicate-nodes` |
| `--trim-strings` |
| `--additional-config` |
| `--array-delimiter` |
| `--bad-tolerance` |
| `--database` |

| name |
| --- |
| `--delimiter` |
| `--id-type` |
| `--input-encoding` |
| `--max-memory` |
| `--processors` |
| `--quote` |
| `--read-buffer-size` |
| `--report-file` |
| `--nodes` |
| `--relationships` |

> **Note:**
>
> 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.

`--expand-commands`

Allow command expansion in config value evaluation.

`--verbose`

Enable verbose output.

`--auto-skip-subsequent-headers`

Automatically skip accidental header lines in subsequent files in file groups with more than one file.

Default: `false`

`--cache-on-heap[=<true/false>]` *Advanced*

Determines whether or not to allow allocating memory for the cache on heap.

If `false`, then caches will still be allocated off-heap, but the additional free memory inside the JVM will not be allocated for the caches.

Use this to have better control over the heap memory.

Default: `false`

`--force[=<true/false>]`

Force deletes any existing database files prior to the import.

Default: `false`

Use `--force=true` to delete all files of a specified database and then import new data. For example:

- When using Neo4j Community Edition.
  Since the Community Edition only supports one database and does not support `DROP DATABASE name`, the only way to re-import data using `neo4j-admin import` is to use `--force=true`.
- When you first want to see how the data would get imported and maybe do some tweaking before you import your actual data. For example, you can first import a small batch of data (e.g., 1000 rows) and examine it. And then, tweak your actual data (e.g., 10 million rows) and use the option `--force=true` to re-import it.

`--high-io[=<true/false>]`

Ignore environment-based heuristics, and specify whether the target storage subsystem can support parallel IO with high throughput.

Typically this is `true` for SSDs, large raid arrays and network-attached storage.

Default: `false`

`--ignore-empty-strings[=<true/false>]`

Determines whether or not empty string fields, such as `""`, from input source are ignored (treated as null).

Default: `false`

`--ignore-extra-columns[=<true/false>]`

If unspecified columns should be ignored during the import.

Default: `false`

`--legacy-style-quoting[=<true/false>]`

Determines whether or not backslash-escaped quote e.g. `\"` is interpreted as inner quote.

Default: `false`

`--multiline-fields[=<true/false>]`

Determines whether or not fields from input source can span multiple lines, i.e. contain newline characters.

Setting `--multiline-fields=true` can severely degrade performance of the importer. Therefore, use it with care, especially with large imports.

Default: `false`

`--normalize-types[=<true/false>]`

Determines whether or not to normalize property types to Cypher types, e.g. `int` becomes `long` and `float` becomes `double`.

Default: `true`

`--skip-bad-entries-logging[=<true/false>]`

Determines whether or not to skip logging bad entries detected during import.

Default: `false`

`--skip-bad-relationships[=<true/false>]`

Determines whether or not to skip importing relationships that refer to missing node IDs, i.e. either start or end node ID/group referring to 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.

Default: `false`

`--skip-duplicate-nodes[=<true/false>]`

Determines 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.

Default: `false`

`--trim-strings[=<true/false>]`

Determines whether or not strings should be trimmed for whitespaces.

Default: `false`

`--additional-config=<config-file-path>`

Path to a configuration file that contain additional configuration options.

`--array-delimiter=<char>`

Determines the array delimiter within a value in CSV data.

- 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 `\`.

Default: `;`

`--bad-tolerance=<num>`

Number of bad entries before the import is considered failed.

This tolerance threshold is about relationships referring to missing nodes. Format errors in input data are still treated as errors.

Default: `1000`

`--database=<name>`

Name of the database to import.

Default: `neo4j`

`--delimiter=<char>`

Determines the delimiter between values in CSV data.

- 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 `\`.

Default: `,`

`--id-type=<STRING|INTEGER|ACTUAL>`

Each node must provide a unique ID in order to be used for creating relationships during the import.

Possible values are:

- `STRING` — arbitrary strings for identifying nodes.
- `INTEGER` — arbitrary integer values for identifying nodes.
- `ACTUAL` — actual node IDs. (**Advanced**)

Default: `STRING`

`--input-encoding=<character-set>`

Character set that input data is encoded in.

Default: `UTF-8`

`--max-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 gigabyte. It can also be specified as a

percentage of the available memory, for example `70%`.

Default: `90%`

`--processors=<num>` *Advanced*

Max number of processors 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 shouldn't be greater than the number of available processors.

`--quote=<char>`

Character to treat as quotation character for values in CSV data.

Quotes can be escaped as per RFC 4180 by doubling them, for example `""` would be interpreted as a literal `"`.

You cannot escape using `\`.

Default: `"`

`--read-buffer-size=<size>`

Size of each buffer for reading input data.

It has to at least be large enough to hold the biggest single value in the input data. Value can be a plain number or byte units string, e.g. `128k`, `1m`.

Default: `4m`

`--report-file=<filename>`

File in which to store the report of the csv-import.

Default: `import.report`

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 `--verbose` option.

`--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.

For an example, see Import data from CSV files using regular expression.

`--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.

For an example, see Import data from CSV files using regular expression.

`@<arguments-file-path>`

File containing all arguments, used as an alternative to supplying all arguments on the command line directly.

Each argument can be on a separate line, or multiple arguments per line and separated by space.

Arguments containing spaces must be quoted.

> **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 will result 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 high limit record format. This is achieved by setting the parameters `dbms.record_format=high_limit` and `dbms.allow_upgrade=true` in a configuration file, and supplying that file to the importer with `--additional-config`. The format is printed in the *debug.log* file.

The `high_limit` format is available for Enterprise Edition only.

## 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 for 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.

## Node files

Files containing node data can have an `ID` field, a `LABEL` field as well as properties.

*ID*

Each node must have a unique ID if it is to be connected by any relationships created in the import. The IDs are used to find the correct nodes when creating relationships. Note that the ID has to be unique across all nodes in the import; even for nodes with different labels. The unique ID can be 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 purpose of the import but not be available for reference later. If no ID is specified, the node will be imported but it will not be able to be connected by any relationships during the import. When a property name is provided, the type of that property can only be configured globally via the `--id-type` option, and can't be specified with a `<field_type>` in the header field (as is possible for properties)

*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`.

*Example 125. Define nodes 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.

*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 126. Define relationships files*

In this example you assume that the two nodes 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 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
```

## Properties

For properties, the `<name>` part of the field designates the property key, while the `<field_type>` part assigns a data type (see below). You can have properties in both node data files and relationship data files.

*Data types*

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. 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`. 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 127. 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 128. 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 which might be common between all or many of the values in the data file. It is therefor 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 129. 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 only unique 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 130. 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
```

## 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 131. 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`.

### Import compressed files

The import tool can handle files compressed with `zip` or `gzip`. Each compressed file must contain a single file.

*Example 132. 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
```

```
neo4j_home$ bin/neo4j-admin 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 cancelled import

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

## 18.2.11. Unbind a Neo4j cluster server

### Command

The cluster state of a Causal Cluster member can be removed by using the following command:

#### Syntax

```
neo4j-admin unbind [--verbose]
                   [--expand-commands]
                   [--archive-cluster-state=<true/false>]
                   [--archive-path=<path>]
```

#### Options

| Option | Default | Description |
| --- | --- | --- |
| `--verbose` | | Enable verbose output. |

| Option | Default | Description |
| --- | --- | --- |
| `--expand-commands` | | Allow command expansion in config value evaluation. |
| `--archive-cluster-state` | false | Enable or disable the cluster state archiving. |
| `--archive-path` | | Destination (file or folder) of the cluster state archive. |

### Limitations

The Neo4j server process must be shut down before running the `neo4j-admin unbind` command.

## Examples of usage

You can use the `neo4j-admin unbind` command to:

- Turn a cluster member into a standalone server:

  To start the Neo4j server in single (standalone) mode after unbinding it from the cluster, first set `dbms.mode=SINGLE` in *neo4j.conf*.

- Seed a Causal Cluster with existing store files:

  To seed a new cluster using the store files of another cluster, you must first run `neo4j-admin unbind` on each server. For more information about seeding Causal Clusters, see Seed a cluster.

  **Warning:**

  If a cluster holds a previous version of any of the databases being seeded, you must `DROP` those databases before seeding. Alternatively, you can stop every instance, unbind them from the cluster using `neo4j-admin unbind` and then forcefully restore the correct seeds (backups) for the databases in question. If you do not `DROP` or `unbind` before seeding, either with `neo4j-admin restore` or `neo4j-admin load`, the database's store files and cluster state will be out of sync, potentially leading to logical corruptions.

- Recover a Causal Cluster:

  In the event of serious failures you may need to recover an entire Causal Cluster from backups. Before restoring those backups, you must first run `neo4j-admin unbind` on each server. For more information about recovering databases from online backups, see Restore a database backup.

  **Note:**

  From Neo4j version 4.0.0 onwards, you **must** run the `neo4j-admin unbind` command on both Read Replicas and Core members.

## 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 unbind --archive-path=/path/to/archive-folder --archive-cluster-state=true
```

The default resultant file is named:

```
unbound_cluster_state.<YYYYMMDDHH24MM>.zip
```

## 18.2.12. Upload to Neo4j AuraDB

The `neo4j-admin push-to-cloud` command uploads a database or a dump into a Neo4j Aura instance. The following table shows the compatibility between the dump version that you want to upload and the version of the Neo4j Aura instance.

| Dump version | Aura version |
| --- | --- |
| v4.4 | v4 and v5 |



**Note:**

This operation is secured and TLS encrypted end to end.

### Prerequisites

Before you can use the `neo4j-admin push-to-cloud` 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.
- Verify that your local database is stopped. You cannot run the `push-to-cloud` command against a source database that is currently in use.



**Caution:**

The `neo4j-admin push-to-cloud` command is not compatible with instances encrypted with Customer Managed Keys. Use `neo4j-admin database upload` in Neo4j Admin version 5 to upload data to instances encrypted with Customer Managed Keys. For more information see Neo4j Admin database upload.

### Syntax

```
neo4j-admin push-to-cloud [--overwrite] [--verbose] --bolt-uri=<boltURI>
                          [--database=<database>] [--dump=<dump>]
                          [--dump-to=<tmpDumpFile>] [--password=<password>]
                          [--username=<username>]
```

## Options

| Option | Default | Description |
| --- | --- | --- |
| `--database` | `neo4j` | Name of the database to push.<br><br>This argument cannot be used together with `--dump`. |
| `--dump` | | Path to an existing database dump for upload, in the format */path/to/my-neo4j-database-dump-file*.<br><br>This argument cannot be used together with `--database`. |
| `--dump-to` | | Optional.<br><br>Target path for temporary database dump file to be uploaded, in the format */path/to/temp-file*.<br><br>Used in combination with the `--database` argument. |
| `--bolt-uri` | | Bolt URI of the target database. For example, `neo4j://mydatabaseid.databases.neo4j.io`. |
| `--username` | | Optional.<br><br>Username of the target database to push this database to. If you do not provide a username, you will be prompted to provide one. Alternatively, the `NEO4J_USERNAME` environment variable can be used. |
| `--password` | | Optional.<br><br>The password of the target database to push this database to. If you do not provide a password, you will be prompted to provide one. Alternatively, you can use the `NEO4J_PASSWORD` environment variable. |
| `--overwrite` | | Optional.<br><br>Overwrite the data in the target database. |
| `--verbose` | | Optional.<br><br>Enable verbose output. |

## Output

If the `push-to-cloud` function completes successfully, it exits with the following logline:

```
Your data was successfully pushed to Aura and is now running.
```

If the `push-to-cloud` 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.

## Examples

The following examples show how to use the `push-to-cloud` command to upload a database or a database dump to a Neo4j Aura instance. You need your AuraDB instance URI (`neo4j+s://your-databaseid.databases.neo4j.io`), as can be seen in the Aura console, and your AuraDB instance password.

> **Note:**
>
> You should use the `--overwrite` 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.

*Upload a database dump to a Neo4j Aura instance*

```
bin/neo4j-admin push-to-cloud --dump=/path/to/dumpfile/movies.dump --bolt-uri=neo4j+s://your-
databaseid.databases.neo4j.io --overwrite

# An example output:

Selecting JVM - Version:11.0.6+8-LTS, Name:Java HotSpot(TM) 64-Bit Server VM, Vendor:Oracle Corporation
Neo4j aura username (default: neo4j):
Neo4j aura password for neo4j:
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 (estimated)
..................  10%
..................  20%
..................  30%
..................  40%
..................  50%
..................  60%
..................  70%
..................  80%
..................  90%
.................. 100%
Your data was successfully pushed to Aura and is now running.
It is safe to delete the dump file now: /path/to/dumpfile/movies.dump
```

*Upload a database to a Neo4j Aura instance*

```
# Stop the `neo4j` database:

bin/cypher-shell -u neo4j -p <password>
neo4j@neo4j> :use system;
neo4j@system> stop database neo4j;

# Run the push-to-cloud command to upload the `neo4j` database into your Aura instance

bin/neo4j-admin push-to-cloud --database=neo4j --bolt-uri=neo4j+s://your-databaseid.databases.neo4j.io
```

```
--overwrite

# An example output:

Selecting JVM - Version:11.0.6+8-LTS, Name:Java HotSpot(TM) 64-Bit Server VM, Vendor:Oracle Corporation
Neo4j aura username (default: neo4j):
Neo4j aura password for neo4j:
Done: 70 files, 854.0KiB processed.
Dumped contents of database 'neo4j' into '/<neo4j-home>/dump-of-neo4j-1669732123683'
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 (estimated)
..................  10%
..................  20%
..................  30%
..................  40%
..................  50%
..................  60%
..................  70%
..................  80%
..................  90%
.................. 100%
Your data was successfully pushed to Aura and is now running.
```

# 18.3. Cypher Shell

## 18.3.1. About Cypher Shell CLI

Cypher Shell is a command-line tool that comes with the Neo4j distribution. It can also be downloaded from Neo4j Deployment Center and installed separately.

Cypher Shell CLI is 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.

## 18.3.2. Syntax

The Cypher Shell CLI is located in the `bin` directory if installed as part of the product.

**Syntax:**

```
cypher-shell  [-h, --help]
              [-a ADDRESS, --address ADDRESS]
              [-u USERNAME, --username USERNAME]
              [-p PASSWORD, --password PASSWORD]
              [--encryption {true,false,default}]
              [-d DATABASE, --database DATABASE]
              [--format {auto,verbose,plain}]
              [-P PARAM, --param PARAM]
              [--debug]
```

```
            [--non-interactive]
            [--sample-rows SAMPLE-ROWS]
            [--wrap {true,false}]
            [-v, --version]
            [--driver-version]
            [-f FILE, --file FILE]
            [--change-password]
            [--fail-fast]
            [--fail-at-end]
            [cypher]
```

Arguments:

| Argument | Type | Description | Default value |
|---|---|---|---|
| `-h`<br><br>`--help` | Optional argument | Show help message and exit. | |
| `-a ADDRESS`<br><br>`--address ADDRESS` | Connection argument | Address and port to connect to. | `neo4j://localhost:7687` |
| `-u USERNAME`<br><br>`--username USERNAME` | Connection argument | Username to connect as. It can also be specified by the environment variable `NEO4J_USERNAME`. | |
| `-p PASSWORD`<br><br>`--password PASSWORD` | Connection argument | Password to connect with. It can also be specified by the environment variable `NEO4J_PASSWORD`. | |
| `--encryption {true,false,default}` | Connection argument | Whether the connection to Neo4j should be encrypted; must be consistent with Neo4j's configuration. | `default`<br><br>The encryption setting is deduced from the specified address. For example, the `neo4j+ssc` protocol would use encryption. |
| `-d DATABASE`<br><br>`--database DATABASE` | Connection argument | Database to connect to. It can also be specified by the environment variable `NEO4J_DATABASE`. | |
| `--format {auto,verbose,plain}` | Optional argument | Desired output format. | `auto` (default): displays results in tabular format if you use the shell interactively and with minimal formatting if you use it for scripting.<br><br>`verbose`: displays results in tabular format and prints statistics.<br><br>`plain`: displays data with minimal formatting. |
| `--P PARAM`<br><br>`--param PARAM` | Optional argument | Add a parameter to this session. For example, `-P "number ⇒ 3"` or `-P "country ⇒ 'Spain'"`. This argument can be specified multiple times. | |

| Argument | Type | Description | Default value |
| --- | --- | --- | --- |
| `--debug` | Optional argument | Print additional debug information. | `false` |
| `--non-interactive` | Optional argument | Force non-interactive mode; only useful if auto-detection fails (e.g. Windows). | `false` |
| `--sample-rows SAMPLE-ROWS` | Optional argument | Number of rows sampled to compute table widths (only for format= `VERBOSE` ). | `1000` |
| `--wrap {true,false}` | Optional argument | Wrap table column values if column is too narrow (only for format= `VERBOSE` ). | `true` |
| `-v`<br><br>`--version` | Optional argument | Print version of cypher-shell and exit. | `false` |
| `--driver-version` | Optional argument | Print version of the Neo4j Driver used and exit. | `false` |
| `-f FILE`<br><br>`--file FILE` | Optional argument | Pass a file with cypher statements to be executed. After the statements have been executed cypher-shell shuts down. | |
| `--change-password` | Optional argument | Change Neo4j user password and exit | `false` |
| `--fail-fast` | Optional argument | Exit and report failure on first error when reading from file. | This is the default behavior. |
| `--fail-at-end` | Optional argument | Exit and report failures at end of input when reading from file. | |
| `cypher` | Positional argument | An optional string of cypher to execute and then exit. | |

## 18.3.3. 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.
```

## 18.3.4. 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 a JDK and Java 11.

> **Note:**
>
> DEB/RPM distributions both install OpenJDK if it is not already installed. 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 the JDK 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.
   ```

## 18.3.5. Available commands

Once in the interactive shell, run the following command to display all available commands:

*Example 133. Running* `help`

```
:help
```

The output is the following:

```
Available commands:
  :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    Print a list of the last commands executed
  :param      Set the value of a query parameter
  :params     Print all currently set query parameters and their values
  :rollback   Rollback the currently open transaction
  :source     Interactively executes cypher statements from a file
  :use        Set the active database

For help on a specific command type:
  :help command
```

## 18.3.6. 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 134. 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 135. 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 136. 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
```

## 18.3.7. Query parameters

Cypher Shell CLI supports querying based on parameters. This is often used while scripting.

*Example 137. Use parameters within Cypher Shell*

1. Set the parameter `thisAlias` to `Robin` using the `:param` keyword:

```
:param thisAlias => 'Robin'
```

2. Check the parameter using the `:params` keyword:

```
:params
```

```
:param thisAlias => 'Robin'
```

3. Now use the parameter `thisAlias` in a Cypher query:

```
CREATE (:Person {name : 'Dick Grayson', alias : $thisAlias });
```

```
Added 1 nodes, Set 2 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"})                  |
+-------------------------------------------------------------------+
3 rows available after 2 ms, consumed after another 2 ms
```

## 18.3.8. Transactions

Cypher Shell supports explicit transactions. Transaction states are controlled using the keywords `:begin`, `:commit`, and `:rollback`.

*Example 138. Use fine-grained transaction control*

The example uses the dataset from the built-in Neo4j Browser guide, called MovieGraph. For more information, see the Neo4j Browser documentation.

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
```

## 18.3.9. Procedures

Cypher Shell supports running any procedures for which the current user is authorized.

*Example 139. Call the `dbms.showCurrentUser` procedure*

```
CALL dbms.showCurrentUser();
```

```
+------------------------------+
| username | roles     | flags |
+------------------------------+
| "neo4j"  | ["admin"] | []    |
+------------------------------+

1 row available after 66 ms, consumed after another 2 ms
```

## 18.3.10. 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.

## 18.3.11. Keyboard shortcuts

The following keyboard commands are available in interactive mode.

| Key | Operation |
| --- | --- |
| 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. |

# Appendix

## Appendix A: Reference

This appendix contains the following topic:

- Procedures

## .A.1. Procedures

This page provides a complete reference to the Neo4j procedures. Available procedures depend on the type of installation you have:

- Neo4j Enterprise Edition provides a larger set of procedures than Neo4j Community Edition.
- 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`:

*Example 140. List available procedures*

```
SHOW PROCEDURES
```



**Note:**

Some procedures can only be run by users with `Admin` privileges. These are labeled with Admin Only.

For more information, see Cypher Manual → Manage Privileges.

### List of procedures

*Table 508. Neo4j procedures*

| Name | Community Edition | Enterprise Edition | Comment |
|---|---|---|---|
| `db.awaitIndex()` | Yes | Yes | |
| `db.awaitIndexes()` | Yes | Yes | |
| `db.checkpoint()` | No | Yes | |
| `db.clearQueryCaches()` | Yes | Yes | Admin Only |

| Name | Community Edition | Enterprise Edition | Comment |
| --- | --- | --- | --- |
| `db.constraints()` | Yes | Yes | Deprecated in 4.2<br>In 4.1, signature changed to `db.constraints() :: (name :: STRING?, description :: STRING?, details :: STRING?)`.<br>Replaced by: `SHOW CONSTRAINTS`. |
| `db.createIndex()` | Yes | Yes | Deprecated in 4.2<br>Replaced by: `CREATE INDEX`. |
| `db.createLabel()` | Yes | Yes | |
| `db.createNodeKey()` | No | Yes | Deprecated in 4.2<br>Replaced by: `CREATE CONSTRAINT … IS NODE KEY`. |
| `db.createProperty()` | Yes | Yes | |
| `db.createRelationshipType()` | Yes | Yes | |
| `db.createUniquePropertyConstraint()` | Yes | Yes | Deprecated in 4.2<br>Replaced by: `CREATE CONSTRAINT … IS UNIQUE`. |
| `db.index.fulltext.awaitEventuallyConsistentIndexRefresh()` | Yes | Yes | |
| `db.index.fulltext.createNodeIndex()` | Yes | Yes | Deprecated in 4.3<br>Replaced by: `CREATE FULLTEXT INDEX …`. |
| `db.index.fulltext.createRelationshipIndex()` | Yes | Yes | Deprecated in 4.3<br>Replaced by: `CREATE FULLTEXT INDEX …`. |
| `db.index.fulltext.drop()` | Yes | Yes | Deprecated in 4.3<br>Replaced by: `DROP INDEX …`. |
| `db.index.fulltext.listAvailableAnalyzers()` | Yes | Yes | |
| `db.index.fulltext.queryNodes()` | Yes | Yes | In 4.1, signature changed to `db.index.fulltext.queryNodes(indexName :: STRING?, queryString :: STRING?, options = {} :: MAP?) :: (node :: NODE?, score :: FLOAT?)`. |

| Name | Community Edition | Enterprise Edition | Comment |
| --- | --- | --- | --- |
| `db.index.fulltext.queryRelationships()` | Yes | Yes | In 4.1, signature changed to `db.index.fulltext.queryRelationships(indexName :: STRING?, queryString :: STRING?, options = {} :: MAP?) :: (relationship :: RELATIONSHIP?, score :: FLOAT?)`. |
| `db.indexDetails()` | Yes | Yes | Deprecated in 4.2<br>Replaced by: `SHOW INDEXES YIELD *`. |
| `db.indexes()` | Yes | Yes | Deprecated in 4.2<br>Replaced by: `SHOW INDEXES`. |
| `db.info()` | Yes | Yes | |
| `db.labels()` | Yes | Yes | |
| `db.listLocks()` | No | Yes | Admin Only<br>In 4.2, signature changed to `db.listLocks() :: (mode :: STRING?, resourceType :: STRING?, resourceId :: INTEGER?, transactionId :: STRING?)`. |
| `db.ping()` | Yes | Yes | |
| `db.prepareForReplanning()` | Yes | Yes | Admin Only |
| `db.propertyKeys()` | Yes | Yes | |
| `db.relationshipTypes()` | Yes | Yes | |
| `db.resampleIndex()` | Yes | Yes | |
| `db.resampleOutdatedIndexes()` | Yes | Yes | |
| `db.schema.nodeTypeProperties()` | Yes | Yes | |
| `db.schema.relTypeProperties()` | Yes | Yes | |
| `db.schema.visualization()` | Yes | Yes | |
| `db.schemaStatements()` | Yes | Yes | Deprecated in 4.2<br>Replaced by: `SHOW INDEXES YIELD *` and `SHOW CONSTRAINTS YIELD *`. |
| `db.stats.clear()` | Yes | Yes | Admin Only |
| `db.stats.collect()` | Yes | Yes | Admin Only |
| `db.stats.retrieve()` | Yes | Yes | Admin Only |
| `db.stats.retrieveAllAnonymized()` | Yes | Yes | Admin Only |

| Name | Community Edition | Enterprise Edition | Comment |
| --- | --- | --- | --- |
| `db.stats.status()` | Yes | Yes | Admin Only |
| `db.stats.stop()` | Yes | Yes | Admin Only |
| `dbms.cluster.routing.getRoutingTable()` | No | Yes | |
| `dbms.cluster.overview()` | No | Yes | |
| `dbms.cluster.protocols()` | No | Yes | |
| `dbms.cluster.quarantineDatabase()` | No | Yes | Deprecated in 4.3<br>Replaced by: `dbms.quarantineDatabase()`. |
| `dbms.cluster.readReplicaToggle()` | No | Yes | |
| `dbms.cluster.role()` | No | Yes | |
| `dbms.cluster.setDefaultDatabase()` | No | Yes | |
| `dbms.components()` | Yes | Yes | |
| `dbms.database.state()` | Yes | Yes | |
| `dbms.functions()` | Yes | Yes | Deprecated in 4.3<br>In 4.2, signature changed to `dbms.functions() :: (name :: STRING?, signature :: STRING?, category :: STRING?, description :: STRING?, aggregating :: BOOLEAN?, defaultBuiltInRoles :: LIST? OF STRING?)`.<br>Replaced by: `SHOW FUNCTIONS`. |
| `dbms.info()` | Yes | Yes | |
| `dbms.killConnection()` | Available since 4.2 | Yes | |
| `dbms.killConnections()` | Available since 4.2 | Yes | |
| `dbms.killQueries()` | Available since 4.2 | Yes | Deprecated in 4.4<br>Replaced by: `TERMINATE TRANSACTIONS`. |
| `dbms.killQuery()` | Available since 4.2 | Yes | Deprecated in 4.4<br>Replaced by: `TERMINATE TRANSACTIONS`. |
| `dbms.killTransaction()` | Available since 4.2 | Yes | Deprecated in 4.4<br>Replaced by: `TERMINATE TRANSACTIONS`. |
| `dbms.killTransactions()` | Available since 4.2 | Yes | Deprecated in 4.4<br>Replaced by: `TERMINATE TRANSACTIONS`. |

| Name | Community Edition | Enterprise Edition | Comment |
| --- | --- | --- | --- |
| `dbms.listActiveLocks()` | Yes | Yes | |
| `dbms.listConfig()` | Yes | Yes | Admin Only |
| `dbms.listConnections()` | Available since 4.2 | Yes | |
| `dbms.listPools()` | No | Yes | |
| `dbms.listQueries()` | Available since 4.2 | Yes | Deprecated in 4.4<br>In 4.1, the `queryId` procedure format changed to no longer include the database name. For example, `mydb-query-123` became query-123.<br>Replaced by: `SHOW TRANSACTIONS`. |
| `dbms.listTransactions()` | Available since 4.2 | Yes | Deprecated in 4.4<br>In 4.1, signature changed to `dbms.listTransactions() :: (transactionId :: STRING?, username :: STRING?, metaData :: MAP?, startTime :: STRING?, protocol :: STRING?, clientAddress :: STRING?, requestUri :: STRING?, currentQueryId :: STRING?, currentQuery :: STRING?, activeLockCount :: INTEGER?, status :: STRING?, resourceInformation :: MAP?, elapsedTimeMillis :: INTEGER?, cpuTimeMillis :: INTEGER?, waitTimeMillis :: INTEGER?, idleTimeMillis :: INTEGER?, allocatedBytes :: INTEGER?, allocatedDirectBytes :: INTEGER?, pageHits :: INTEGER?, pageFaults :: INTEGER?, connectionId :: STRING?, initializationStackTrace :: STRING?, database :: STRING?, estimatedUsedHeapMemory :: INTEGER?)`.<br>Replaced by: `SHOW TRANSACTIONS`. |
| `dbms.procedures()` | Yes | Yes | Deprecated in 4.3<br>Replaced by: `SHOW PROCEDURES`. |
| `dbms.quarantineDatabase()` | No | Yes | |
| `dbms.queryJmx()` | Yes | Yes | |
| `dbms.routing.getRoutingTable()` | Yes | Yes | |
| `dbms.scheduler.failedJobs()` | No | Yes | Admin Only |

| Name | Community Edition | Enterprise Edition | Comment |
| --- | --- | --- | --- |
| `dbms.scheduler.groups()` | No | Yes | Admin Only |
| `dbms.scheduler.jobs()` | No | Yes | Admin Only |
| `dbms.scheduler.profile()` | No | Yes | Admin Only |
| `dbms.security.activateUser()` | No | Yes | Deprecated in 4.0 Admin Only<br>In 4.1, mode changed to `write`.<br>Replaced by: `ALTER USER`. |
| `dbms.security.addRoleToUser()` | No | Yes | Deprecated in 4.0 Admin Only<br>In 4.1, mode changed to `write`.<br>Replaced by: `GRANT ROLE TO USER`. |
| `dbms.security.changePassword()` | Yes | Yes | Deprecated in 4.0 Admin Only<br>In 4.1, mode changed to `write`.<br>Replaced by: `ALTER CURRENT USER SET PASSWORD`. |
| `dbms.security.changeUserPassword()` | No | Yes | Deprecated in 4.0 Admin Only<br>In 4.1, mode changed to `write`.<br>Replaced by: `ALTER USER`. |
| `dbms.security.clearAuthCache()` | No | Yes | Admin Only |
| `dbms.security.createRole()` | No | Yes | Deprecated in 4.0 Admin Only<br>In 4.1, mode changed to `write`.<br>Replaced by: `CREATE ROLE`. |
| `dbms.security.createUser()` | Yes | Yes | Deprecated in 4.0 Admin Only<br>In 4.1, mode changed to `write`.<br>Replaced by: `CREATE USER`. |
| `dbms.security.deleteRole()` | No | Yes | Deprecated in 4.0 Admin Only<br>In 4.1, mode changed to `write`.<br>Replaced by: `DROP ROLE`. |
| `dbms.security.deleteUser()` | Yes | Yes | Deprecated in 4.0 Admin Only<br>In 4.1, mode changed to `write`.<br>Replaced by: `DROP USER`. |
| `dbms.security.listRoles()` | Yes | Yes | Deprecated in 4.0 Admin Only<br>In 4.1, mode changed to `read`.<br>Replaced by: `SHOW ROLES`. |
| `dbms.security.listRolesForUser()` | No | Yes | Deprecated in 4.0<br>In 4.1, mode changed to `read`.<br>Replaced by: `SHOW USERS`. |

| Name | Community Edition | Enterprise Edition | Comment |
|---|---|---|---|
| `dbms.security.listUsers()` | Yes | Yes | Deprecated in 4.0 Admin Only<br>In 4.1, mode changed to `read`.<br>Replaced by: `SHOW USERS`. |
| `dbms.security.listUsersForRole()` | No | Yes | Deprecated in 4.0 Admin Only<br>In 4.1, mode changed to `read`.<br>Replaced by: `SHOW ROLES WITH USERS`. |
| `dbms.security.removeRoleFromUser()` | No | Yes | Deprecated in 4.0 Admin Only<br>In 4.1, mode changed to `write`.<br>Replaced by: `REVOKE ROLE FROM USER`. |
| `dbms.security.suspendUser()` | No | Yes | Deprecated in 4.0 Admin Only<br>In 4.1, mode changed to `write`.<br>Replaced by: `ALTER USER`. |
| `dbms.setConfigValue()` | No | Yes | Admin Only |
| `dbms.showCurrentUser()` | Yes | Yes | |
| `dbms.upgrade()` | Yes | Yes | Admin Only |
| `dbms.upgradeStatus()` | Yes | Yes | Admin Only |
| `tx.getMetaData()` | Yes | Yes | |
| `tx.setMetaData()` | Yes | Yes | |

## Procedure descriptions

*Table 509. db.awaitIndex()*

| | |
|---|---|
| Description | Wait for an index to come online.<br>Example: `CALL db.awaitIndex("MyIndex", 300)` |
| Signature | `db.awaitIndex(indexName :: STRING?, timeOutSeconds = 300 :: INTEGER?) :: VOID` |
| Mode | READ |

*Table 510. db.awaitIndexes()*

| | |
|---|---|
| Description | Wait for all indexes to come online.<br>Example: `CALL db.awaitIndexes(300)` |
| Signature | `db.awaitIndexes(timeOutSeconds = 300 :: INTEGER?) :: VOID` |
| Mode | READ |

*Table 511. db.checkpoint()* Enterprise Edition

| | |
|---|---|
| Description | Initiate and wait for a new check point, or wait any already on-going check point to complete.<br><br>Note that this temporarily disables the `dbms.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. |
| Signature | `db.checkpoint() :: (success :: BOOLEAN?, message :: STRING?)` |
| Mode | `DBMS` |

*Table 512. db.clearQueryCaches()* Admin Only

| | |
|---|---|
| Description | Clears all query caches. |
| Signature | `db.clearQueryCaches() :: (value :: STRING?)` |
| Mode | `DBMS` |

*Table 513. db.constraints()* Deprecated in 4.2

| | |
|---|---|
| Description | List all constraints in the database. |
| Signature | `db.constraints() :: (name :: STRING?, description :: STRING?, details :: STRING?)` |
| Mode | `READ` |
| Replaced by | `SHOW CONSTRAINTS`. For more information, see Cypher Manual → Database administration. |

*Table 514. db.createIndex()* Deprecated in 4.2

| | |
|---|---|
| Description | Create a named schema index with specified index provider and configuration (optional).<br><br>Yield: name, labels, properties, providerName, status |
| Signature | `db.createIndex(indexName :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, config = {} :: MAP?) :: (name :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, status :: STRING?)` |
| Mode | `SCHEMA` |
| Replaced by | `CREATE INDEX`. For more information, see Cypher Manual → Database administration. |

*Table 515. db.createLabel()*

| | |
|---|---|
| Description | Create a label |
| Signature | `db.createLabel(newLabel :: STRING?) :: VOID` |
| Mode | `WRITE` |

*Table 516. db.createNodeKey()* Enterprise Edition Deprecated in 4.2

| | |
|---|---|
| Description | Create a named node key constraint. Backing index will use specified index provider and configuration (optional).<br><br>Yield: name, labels, properties, providerName, status |

| | |
|---|---|
| Signature | `db.createNodeKey(constraintName :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, config = {} :: MAP?) :: (name :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, status :: STRING?)` |
| Mode | `SCHEMA` |
| Replaced by | `CREATE CONSTRAINT … IS NODE KEY`. For more information, see Cypher Manual → Database administration. |

*Table 517. db.createProperty()*

| | |
|---|---|
| Description | Create a Property |
| Signature | `db.createProperty(newProperty :: STRING?) :: VOID` |
| Mode | `WRITE` |

*Table 518. db.createRelationshipType()*

| | |
|---|---|
| Description | Create a RelationshipType |
| Signature | `db.createRelationshipType(newRelationshipType :: STRING?) :: VOID` |
| Mode | `WRITE` |

*Table 519. db.createUniquePropertyConstraint()* Deprecated in 4.2

| | |
|---|---|
| Description | Create a named unique property constraint.<br><br>Backing index will use specified index provider and configuration (optional).<br><br>Yield: name, labels, properties, providerName, status |
| Signature | `db.createUniquePropertyConstraint(constraintName :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, config = {} :: MAP?) :: (name :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, status :: STRING?)` |
| Mode | `SCHEMA` |
| Replaced by | `CREATE CONSTRAINT … IS UNIQUE`. For more information, see Cypher Manual → Database administration. |

*Table 520. db.index.fulltext.awaitEventuallyConsistentIndexRefresh()*

| | |
|---|---|
| Description | Wait for the updates from recently committed transactions to be applied to any eventually-consistent full-text indexes. |
| Signature | `db.index.fulltext.awaitEventuallyConsistentIndexRefresh() :: VOID` |
| Mode | `READ` |

*Table 521. db.index.fulltext.createNodeIndex()* Deprecated in 4.3

| | |
|---|---|
| Description | Create a node full-text index for the given labels and properties.<br><br>The optional 'config' map parameter can be used to supply settings to the index. Supported settings are 'analyzer', for specifying what analyzer to use when indexing and querying. Use the `db.index.fulltext.listAvailableAnalyzers` procedure to see what options are available. And 'eventually_consistent' which can be set to 'true' to make this index eventually consistent, such that updates from committing transactions are applied in a background thread. |
| Signature | `db.index.fulltext.createNodeIndex(indexName :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, config = {} :: MAP?) :: VOID` |
| Mode | SCHEMA |
| Replaced by | `CREATE FULLTEXT INDEX` |

*Table 522. db.index.fulltext.createRelationshipIndex()* Deprecated in 4.3

| | |
|---|---|
| Description | Create a relationship full-text index for the given relationship types and properties.<br><br>The optional 'config' map parameter can be used to supply settings to the index. Supported settings are 'analyzer', for specifying what analyzer to use when indexing and querying. Use the `db.index.fulltext.listAvailableAnalyzers` procedure to see what options are available. And 'eventually_consistent' which can be set to 'true' to make this index eventually consistent, such that updates from committing transactions are applied in a background thread. |
| Signature | `db.index.fulltext.createRelationshipIndex(indexName :: STRING?, relationshipTypes :: LIST? OF STRING?, properties :: LIST? OF STRING?, config = {} :: MAP?) :: VOID` |
| Mode | SCHEMA |
| Replaced by | `CREATE FULLTEXT INDEX …` |

*Table 523. db.index.fulltext.drop()* Deprecated in 4.3

| | |
|---|---|
| Description | Drop the specified index. |
| Signature | `db.index.fulltext.drop(indexName :: STRING?) :: VOID` |
| Mode | SCHEMA |
| Replaced by | `DROP INDEX …` |

*Table 524. db.index.fulltext.listAvailableAnalyzers()*

| | |
|---|---|
| Description | List the available analyzers that the full-text indexes can be configured with. |
| Signature | `db.index.fulltext.listAvailableAnalyzers() :: (analyzer :: STRING?, description :: STRING?, stopwords :: LIST? OF STRING?)` |
| Mode | READ |

*Table 525. db.index.fulltext.queryNodes()*

| | |
|---|---|
| Description | Query the given full-text index.<br><br>Returns the matching nodes, and their Lucene query score, ordered by score.<br><br>Valid keys for the options map are: 'skip' to skip the top N results; 'limit' to limit the number of results returned. |
| Signature | `db.index.fulltext.queryNodes(indexName :: STRING?, queryString :: STRING?, options = {} :: MAP?) :: (node :: NODE?, score :: FLOAT?)` |
| Mode | READ |

*Table 526. db.index.fulltext.queryRelationships()*

| | |
|---|---|
| Description | Query the given full-text index.<br><br>Returns the matching relationships, and their Lucene query score, ordered by score.<br><br>Valid keys for the options map are: 'skip' to skip the top N results; 'limit' to limit the number of results returned. |
| Signature | `db.index.fulltext.queryRelationships(indexName :: STRING?, queryString :: STRING?, options = {} :: MAP?) :: (relationship :: RELATIONSHIP?, score :: FLOAT?)` |
| Mode | READ |

*Table 527. db.indexDetails()* Deprecated in 4.2

| | |
|---|---|
| Description | Detailed description of specific index. |
| Signature | `db.indexDetails(indexName :: STRING?) :: (id :: INTEGER?, name :: STRING?, state :: STRING?, populationPercent :: FLOAT?, uniqueness :: STRING?, type :: STRING?, entityType :: STRING?, labelsOrTypes :: LIST? OF STRING?, properties :: LIST? OF STRING?, provider :: STRING?, indexConfig :: MAP?, failureMessage :: STRING?)` |
| Mode | READ |
| Replaced by | `SHOW INDEXES YIELD *` |

*Table 528. db.indexes()* Deprecated in 4.2

| | |
|---|---|
| Description | List all indexes in the database. |
| Signature | `db.indexes() :: (id :: INTEGER?, name :: STRING?, state :: STRING?, populationPercent :: FLOAT?, uniqueness :: STRING?, type :: STRING?, entityType :: STRING?, labelsOrTypes :: LIST? OF STRING?, properties :: LIST? OF STRING?, provider :: STRING?)` |
| Mode | READ |
| Replaced by | `SHOW INDEXES` |

*Table 529. db.info()*

| | |
|---|---|
| Description | Provides information regarding the database. |
| Signature | `db.info() :: (id :: STRING?, name :: STRING?, creationDate :: STRING?)` |

| | |
|---|---|
| Mode | READ |

*Table 530. db.labels()*

| | |
|---|---|
| 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. |
| Signature | `db.labels() :: (label :: STRING?)` |
| Mode | READ |

*Table 531. db.listLocks()* Enterprise Edition Admin Only

| | |
|---|---|
| Description | List all locks at this database. |
| Signature | `db.listLocks() :: (mode :: STRING?, resourceType :: STRING?, resourceId :: INTEGER?, transactionId :: STRING?)` |
| Mode | DBMS |

*Table 532. db.ping()*

| | |
|---|---|
| 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. |
| Signature | `db.ping() :: (success :: BOOLEAN?)` |
| Mode | READ |

*Table 533. db.prepareForReplanning()* Admin Only

| | |
|---|---|
| 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. |
| Signature | `db.prepareForReplanning(timeOutSeconds = 300 :: INTEGER?) :: VOID` |
| Mode | READ |

*Table 534. db.propertyKeys()*

| | |
|---|---|
| Description | List all property keys in the database. |
| Signature | `db.propertyKeys() :: (propertyKey :: STRING?)` |
| Mode | READ |

*Table 535. db.relationshipTypes()*

| | |
|---|---|
| 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. |
| Signature | `db.relationshipTypes() :: (relationshipType :: STRING?)` |
| Mode | READ |

*Table 536. db.resampleIndex()*

| | |
|---|---|
| Description | Schedule resampling of an index.<br><br>Example: `CALL db.resampleIndex("MyIndex")` |
| Signature | `db.resampleIndex(indexName :: STRING?) :: VOID` |
| Mode | READ |

*Table 537. db.resampleOutdatedIndexes()*

| | |
|---|---|
| Description | Schedule resampling of all outdated indexes. |
| Signature | `db.resampleOutdatedIndexes() :: VOID` |
| Mode | READ |

*Table 538. db.schema.nodeTypeProperties()*

| | |
|---|---|
| Description | Show the derived property schema of the nodes in tabular form. |
| Signature | `db.schema.nodeTypeProperties() :: (nodeType :: STRING?, nodeLabels :: LIST? OF STRING?, propertyName :: STRING?, propertyTypes :: LIST? OF STRING?, mandatory :: BOOLEAN?)` |
| Mode | READ |

*Table 539. db.schema.relTypeProperties()*

| | |
|---|---|
| Description | Show the derived property schema of the relationships in tabular form. |
| Signature | `db.schema.relTypeProperties() :: (relType :: STRING?, propertyName :: STRING?, propertyTypes :: LIST? OF STRING?, mandatory :: BOOLEAN?)` |
| Mode | READ |

*Table 540. db.schema.visualization()*

| | |
|---|---|
| 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. Note that this may include additional relationships that do not exist in the data due to the information available in the count store. |
| Signature | `db.schema.visualization() :: (nodes :: LIST? OF NODE?, relationships :: LIST? OF RELATIONSHIP?)` |
| Mode | READ |

*Table 541. db.schemaStatements()* Deprecated in 4.2

| | |
|---|---|
| Description | List all statements for creating and dropping existing indexes and constraints. Note that only index types introduced before Neo4j 4.3 are included. |
| Signature | `db.schemaStatements() :: (name :: STRING?, type :: STRING?, createStatement :: STRING?, dropStatement :: STRING?)` |
| Mode | READ |

| | |
|---|---|
| Mode | READ |
| Replaced by | `SHOW INDEXES YIELD *` and `SHOW CONSTRAINTS YIELD *`. For more information, see Cypher Manual → Database administration. |

*Table 542. db.stats.clear()* Admin Only

| | |
|---|---|
| Description | Clear collected data of a given data section.<br><br>Valid sections are `'QUERIES'` |
| Signature | `db.stats.clear(section :: STRING?) :: (section :: STRING?, success :: BOOLEAN?, message :: STRING?)` |
| Mode | READ |

*Table 543. db.stats.collect()* Admin Only

| | |
|---|---|
| Description | Start data collection of a given data section.<br><br>Valid sections are `'QUERIES'` |
| Signature | `db.stats.collect(section :: STRING?, config = {} :: MAP?) :: (section :: STRING?, success :: BOOLEAN?, message :: STRING?)` |
| Mode | READ |

*Table 544. db.stats.retrieve()* Admin Only

| | |
|---|---|
| Description | Retrieve statistical data about the current database.<br><br>Valid sections are `'GRAPH COUNTS', 'TOKENS', 'QUERIES', 'META'` |
| Signature | `db.stats.retrieve(section :: STRING?, config = {} :: MAP?) :: (section :: STRING?, data :: MAP?)` |
| Mode | READ |

*Table 545. db.stats.retrieveAllAnonymized()* Admin Only

| | |
|---|---|
| Description | Retrieve all available statistical data about the current database, in an anonymized form. |
| Signature | `db.stats.retrieveAllAnonymized(graphToken :: STRING?, config = {} :: MAP?) :: (section :: STRING?, data :: MAP?)` |
| Mode | READ |

*Table 546. db.stats.status()* Admin Only

| | |
|---|---|
| Description | Retrieve the status of all available collector daemons, for this database. |
| Signature | `db.stats.status() :: (section :: STRING?, status :: STRING?, data :: MAP?)` |
| Mode | READ |

*Table 547. db.stats.stop()* Admin Only

| | |
|---|---|
| Description | Stop data collection of a given data section.<br><br>Valid sections are `'QUERIES'` |
| Signature | `db.stats.stop(section :: STRING?) :: (section :: STRING?, success :: BOOLEAN?, message :: STRING?)` |
| Mode | READ |

*Table 548. dbms.cluster.routing.getRoutingTable()*

| | |
|---|---|
| Description | Returns endpoints of this instance. |
| Signature | `dbms.cluster.routing.getRoutingTable(context :: MAP?, database = null :: STRING?) :: (ttl :: INTEGER?, servers :: LIST? OF MAP?)` |
| Mode | DBMS |

*Table 549. dbms.cluster.overview()* Enterprise Edition

| | |
|---|---|
| Description | Overview of all currently accessible cluster members, their databases and roles. |
| Signature | `dbms.cluster.overview() :: (id :: STRING?, addresses :: LIST? OF STRING?, databases :: MAP?, groups :: LIST? OF STRING?)` |
| Mode | READ |

*Table 550. dbms.cluster.protocols()* Enterprise Edition

| | |
|---|---|
| Description | Overview of installed protocols.<br><br>Note that this can only be executed on a cluster core member. |
| Signature | `dbms.cluster.protocols() :: (orientation :: STRING?, remoteAddress :: STRING?, applicationProtocol :: STRING?, applicationProtocolVersion :: INTEGER?, modifierProtocols :: STRING?)` |
| Mode | READ |

*Table 551. dbms.cluster.quarantineDatabase()* Enterprise Edition Deprecated in 4.3

| | |
|---|---|
| Description | Place a database in quarantine or remove thereof. |
| Signature | `dbms.cluster.quarantineDatabase(databaseName :: STRING?, setStatus :: BOOLEAN?, reason = No reason given :: STRING?) :: (databaseName :: STRING?, quarantined :: BOOLEAN?, result :: STRING?)` |
| Mode | DBMS |
| Replaced by | `dbms.quarantineDatabase()` |

*Table 552. dbms.cluster.readReplicaToggle()* Enterprise Edition

| | |
|---|---|
| Description | The toggle can pause or resume the pulling of new transactions for a specific database. If paused, the Read Replica does not pull new transactions from the other cluster members for the specific database. The Read Replica is still available for reads, you can perform a backup, etc.<br><br>**Tip:**<br><br>*What is it for?*<br><br>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.<br><br>1. Connect directly to the Read Replica cluster member. (Neo4j Driver use `bolt://` or use the HTTP API).<br>2. Pause transaction pulling for the specified database.<br>3. Create a point in time backup, see Back up an online database.<br><br>If connected directly to a Read Replica, Data Scientists can execute analysis on a specific database that is paused, the data will not unexpectedly change while performing the analysis.<br><br>**Note:**<br><br>This procedure can only be executed on a Read Replica cluster member.<br><br>*Pause transaction pulling for database* `neo4j`<br><br>`CALL dbms.cluster.readReplicaToggle("neo4j", true)`<br><br>*Resume transaction pulling for database* `neo4j`<br><br>`CALL dbms.cluster.readReplicaToggle("neo4j", false)` |
| Signature | `dbms.cluster.readReplicaToggle(databaseName :: STRING?, pause :: BOOLEAN?) :: (state :: STRING?)` |
| Mode | READ |

*Table 553. dbms.cluster.role()* Enterprise Edition

| | |
|---|---|
| Description | The role of this instance in the cluster for the specified database. |
| Signature | `dbms.cluster.role(database :: STRING?) :: (role :: STRING?)` |
| Mode | READ |

*Table 554. dbms.cluster.setDefaultDatabase()* Enterprise Edition

| | |
|---|---|
| Description | Change the default database to the provided value.<br><br>The database must exist and the old default database must be stopped.<br><br>For more information see Change the default database.<br><br>Note that this can only be executed on a cluster core member. |
| Signature | `dbms.cluster.setDefaultDatabase(databaseName :: STRING?) :: (result :: STRING?)` |
| Mode | `WRITE` |

*Table 555. dbms.components()*

| | |
|---|---|
| Description | List DBMS components and their versions. |
| Signature | `dbms.components() :: (name :: STRING?, versions :: LIST? OF STRING?, edition :: STRING?)` |
| Mode | `DBMS` |

*Table 556. dbms.database.state()*

| | |
|---|---|
| Description | The actual status of the database with the provided name on this neo4j instance. |
| Signature | `dbms.database.state(databaseName :: STRING?) :: (role :: STRING?, address :: STRING?, status :: STRING?, error :: STRING?)` |
| Mode | `READ` |

*Table 557. dbms.functions()* Deprecated in 4.3

| | |
|---|---|
| Description | List all functions in the DBMS. |
| Signature | `dbms.functions() :: (name :: STRING?, signature :: STRING?, category :: STRING?, description :: STRING?, aggregating :: BOOLEAN?, defaultBuiltInRoles :: LIST? OF STRING?)` |
| Mode | `DBMS` |
| Replaced by | `SHOW FUNCTIONS` |

*Table 558. dbms.info()*

| | |
|---|---|
| Description | Provides information regarding the DBMS. |
| Signature | `dbms.info() :: (id :: STRING?, name :: STRING?, creationDate :: STRING?)` |
| Mode | `DBMS` |

*Table 559. dbms.killConnection()*

| | |
|---|---|
| Description | Kill network connection with the given connection id. |
| Signature | `dbms.killConnection(id :: STRING?) :: (connectionId :: STRING?, username :: STRING?, message :: STRING?)` |
| Mode | `DBMS` |

Table 560. dbms.killConnections()

| | |
|---|---|
| Description | Kill all network connections with the given connection ids. |
| Signature | `dbms.killConnections(ids :: LIST? OF STRING?) :: (connectionId :: STRING?, username :: STRING?, message :: STRING?)` |
| Mode | DBMS |

Table 561. dbms.killQueries() Deprecated in 4.4

| | |
|---|---|
| Description | Kill all transactions executing a query with any of the given query ids. |
| Signature | `dbms.killQueries(ids :: LIST? OF STRING?) :: (queryId :: STRING?, username :: STRING?, message :: STRING?)` |
| Mode | DBMS |
| Replaced by | `TERMINATE TRANSACTIONS` |

Table 562. dbms.killQuery() Deprecated in 4.4

| | |
|---|---|
| Description | Kill all transactions executing the query with the given query id. |
| Signature | `dbms.killQuery(id :: STRING?) :: (queryId :: STRING?, username :: STRING?, message :: STRING?)` |
| Mode | DBMS |
| Replaced by | `TERMINATE TRANSACTIONS` |

Table 563. dbms.killTransaction() Deprecated in 4.4

| | |
|---|---|
| Description | Kill transaction with provided id. |
| Signature | `dbms.killTransaction(id :: STRING?) :: (transactionId :: STRING?, username :: STRING?, message :: STRING?)` |
| Mode | DBMS |
| Replaced by | `TERMINATE TRANSACTIONS` |

Table 564. dbms.killTransactions() Deprecated in 4.4

| | |
|---|---|
| Description | Kill transactions with provided ids. |
| Signature | `dbms.killTransactions(ids :: LIST? OF STRING?) :: (transactionId :: STRING?, username :: STRING?, message :: STRING?)` |
| Mode | DBMS |
| Replaced by | `TERMINATE TRANSACTIONS` |

Table 565. dbms.listActiveLocks() Enterprise Edition

| | |
|---|---|
| Description | List the active lock requests granted for the transaction executing the query with the given query id. |

| | |
|---|---|
| Signature | `dbms.listActiveLocks(queryId :: STRING?) :: (mode :: STRING?, resourceType :: STRING?, resourceId :: INTEGER?)` |
| Mode | DBMS |

*Table 566. dbms.listConfig()* Admin Only

| | |
|---|---|
| Description | List the currently active config of Neo4j. |
| Signature | `dbms.listConfig(searchString = :: STRING?) :: (name :: STRING?, description :: STRING?, value :: STRING?, dynamic :: BOOLEAN?)` |
| Mode | DBMS |

*Table 567. dbms.listConnections()*

| | |
|---|---|
| Description | List all accepted network connections at this instance that are visible to the user. |
| Signature | `dbms.listConnections() :: (connectionId :: STRING?, connectTime :: STRING?, connector :: STRING?, username :: STRING?, userAgent :: STRING?, serverAddress :: STRING?, clientAddress :: STRING?)` |
| Mode | DBMS |

*Table 568. dbms.listPools()* Enterprise Edition

| | |
|---|---|
| Description | List all memory pools, including sub pools, currently registered at this instance that are visible to the user. |
| Signature | `dbms.listPools() :: (pool :: STRING?, databaseName :: STRING?, heapMemoryUsed :: STRING?, heapMemoryUsedBytes :: STRING?, nativeMemoryUsed :: STRING?, nativeMemoryUsedBytes :: STRING?, freeMemory :: STRING?, freeMemoryBytes :: STRING?, totalPoolMemory :: STRING?, totalPoolMemoryBytes :: STRING?)` |
| Mode | DBMS |

*Table 569. dbms.listQueries()* Deprecated in 4.4

| | |
|---|---|
| Description | List all queries currently executing at this instance that are visible to the user. |
| Signature | `dbms.listQueries() :: (queryId :: STRING?, username :: STRING?, metaData :: MAP?, query :: STRING?, parameters :: MAP?, planner :: STRING?, runtime :: STRING?, indexes :: LIST? OF MAP?, startTime :: STRING?, protocol :: STRING?, clientAddress :: STRING?, requestUri :: STRING?, status :: STRING?, resourceInformation :: MAP?, activeLockCount :: INTEGER?, elapsedTimeMillis :: INTEGER?, cpuTimeMillis :: INTEGER?, waitTimeMillis :: INTEGER?, idleTimeMillis :: INTEGER?, allocatedBytes :: INTEGER?, pageHits :: INTEGER?, pageFaults :: INTEGER?, connectionId :: STRING?, database :: STRING?)` |
| Mode | DBMS |
| Replaced by | `SHOW TRANSACTIONS` |

*Table 570. dbms.listTransactions()* Deprecated in 4.4

| | |
|---|---|
| Description | List all transactions currently executing at this instance that are visible to the user. |

| | |
|---|---|
| Signature | `dbms.listTransactions() :: (transactionId :: STRING?, username :: STRING?, metaData :: MAP?, startTime :: STRING?, protocol :: STRING?, clientAddress :: STRING?, requestUri :: STRING?, currentQueryId :: STRING?, currentQuery :: STRING?, activeLockCount :: INTEGER?, status :: STRING?, resourceInformation :: MAP?, elapsedTimeMillis :: INTEGER?, cpuTimeMillis :: INTEGER?, waitTimeMillis :: INTEGER?, idleTimeMillis :: INTEGER?, allocatedBytes :: INTEGER?, allocatedDirectBytes :: INTEGER?, pageHits :: INTEGER?, pageFaults :: INTEGER?, connectionId :: STRING?, initializationStackTrace :: STRING?, database :: STRING?, estimatedUsedHeapMemory :: INTEGER?)` |
| Mode | DBMS |
| Replaced by | `SHOW TRANSACTIONS` |

*Table 571. dbms.procedures()* Deprecated in 4.3

| | |
|---|---|
| Description | List all procedures in the DBMS. |
| Signature | `dbms.procedures() :: (name :: STRING?, signature :: STRING?, description :: STRING?, mode :: STRING?, defaultBuiltInRoles :: LIST? OF STRING?, worksOnSystem :: BOOLEAN?)` |
| Mode | DBMS |
| Replaced by | `SHOW PROCEDURES` |

*Table 572. dbms.quarantineDatabase*

| | |
|---|---|
| Description | Place a database in quarantine or remove thereof. |
| Signature | `dbms.quarantineDatabase(databaseName :: STRING?, setStatus :: BOOLEAN?, reason = No reason given :: STRING?) :: (databaseName :: STRING?, quarantined :: BOOLEAN?, result :: STRING?)` |
| Mode | DBMS |

*Table 573. dbms.queryJmx()*

| | |
|---|---|
| Description | Query JMX management data by domain and name.<br><br>Valid queries should use the syntax outlined in the javax.management.ObjectName API documentation. For instance, use `"*:*"` to find all JMX beans. |
| Signature | `dbms.queryJmx(query :: STRING?) :: (name :: STRING?, description :: STRING?, attributes :: MAP?)` |
| Mode | DBMS |

*Table 574. dbms.routing.getRoutingTable()*

| | |
|---|---|
| Description | Returns endpoints of this instance. |
| Signature | `dbms.routing.getRoutingTable(context :: MAP?, database = null :: STRING?) :: (ttl :: INTEGER?, servers :: LIST? OF MAP?)` |
| Mode | DBMS |

*Table 575. dbms.scheduler.failedJobs()* Enterprise Edition Admin Only

| | |
|---|---|
| Description | List failed job runs. There is a limit for amount of historical data. |
| Signature | `dbms.scheduler.failedJobs() :: (jobId :: STRING?, group :: STRING?, database :: STRING?, submitter :: STRING?, description :: STRING?, type :: STRING?, submitted :: STRING?, executionStart :: STRING?, failureTime :: STRING?, failureDescription :: STRING?)` |
| Mode | DBMS |

Table 576. dbms.scheduler.groups() Enterprise Edition Admin Only

| | |
|---|---|
| Description | List the job groups that are active in the database internal job scheduler. |
| Signature | `dbms.scheduler.groups() :: (group :: STRING?, threads :: INTEGER?)` |
| Mode | DBMS |

Table 577. dbms.scheduler.jobs() Enterprise Edition Admin Only

| | |
|---|---|
| Description | List all jobs that are active in the database internal job scheduler. |
| Signature | `dbms.scheduler.jobs() :: (jobId :: STRING?, group :: STRING?, submitted :: STRING?, database :: STRING?, submitter :: STRING?, description :: STRING?, type :: STRING?, scheduledAt :: STRING?, period :: STRING?, state :: STRING?, currentStateDescription :: STRING?)` |
| Mode | DBMS |

Table 578. dbms.scheduler.profile() Enterprise Edition Admin Only

| | |
|---|---|
| Description | Begin profiling all threads within the given job group, for the specified duration.<br><br>Note that profiling incurs overhead to a system, and will slow it down. |
| Signature | `dbms.scheduler.profile(method :: STRING?, group :: STRING?, duration :: STRING?) :: (profile :: STRING?)` |
| Mode | DBMS |

Table 579. dbms.security.activateUser() Enterprise Edition Admin Only Deprecated in 4.0

| | |
|---|---|
| Description | Activate a suspended user. |
| Signature | `dbms.security.activateUser(username :: STRING?, requirePasswordChange = true :: BOOLEAN?) :: VOID` |
| Mode | WRITE |

Table 580. dbms.security.addRoleToUser() Enterprise Edition Admin Only Deprecated in 4.0

| | |
|---|---|
| Description | Assign a role to the user. |
| Signature | `dbms.security.addRoleToUser(roleName :: STRING?, username :: STRING?) :: VOID` |
| Mode | WRITE |

Table 581. dbms.security.changePassword() Admin Only Deprecated in 4.0

| | |
|---|---|
| Description | Change the current user's password. |
| Signature | `dbms.security.changePassword(password :: STRING?, requirePasswordChange = false :: BOOLEAN?) :: VOID` |
| Mode | `WRITE` |

*Table 582. dbms.security.changeUserPassword()* Enterprise Edition Admin Only Deprecated in 4.0

| | |
|---|---|
| Description | Change the given user's password. |
| Signature | `dbms.security.changeUserPassword(username :: STRING?, newPassword :: STRING?, requirePasswordChange = true :: BOOLEAN?) :: VOID` |
| Mode | `WRITE` |

*Table 583. dbms.security.clearAuthCache()* Enterprise Edition Admin Only

| | |
|---|---|
| Description | Clears authentication and authorization cache. |
| Signature | `dbms.security.clearAuthCache() :: VOID` |
| Mode | `DBMS` |

*Table 584. dbms.security.createRole()* Enterprise Edition Admin Only Deprecated in 4.0

| | |
|---|---|
| Description | Create a new role. |
| Signature | `dbms.security.createRole(roleName :: STRING?) :: VOID` |
| Mode | `WRITE` |

*Table 585. dbms.security.createUser()* Admin Only Deprecated in 4.0

| | |
|---|---|
| Description | Create a new user. |
| Signature | `dbms.security.createUser(username :: STRING?, password :: STRING?, requirePasswordChange = true :: BOOLEAN?) :: VOID` |
| Mode | `WRITE` |

*Table 586. dbms.security.deleteRole()* Enterprise Edition Admin Only Deprecated in 4.0

| | |
|---|---|
| Description | Delete the specified role. Any role assignments will be removed. |
| Signature | `dbms.security.deleteRole(roleName :: STRING?) :: VOID` |
| Mode | `WRITE` |

*Table 587. dbms.security.deleteUser()* Admin Only Deprecated in 4.0

| | |
|---|---|
| Description | Delete the specified user. |
| Signature | `dbms.security.deleteUser(username :: STRING?) :: VOID` |

| | |
|---|---|
| Mode | WRITE |

Table 588. dbms.security.listRoles() Enterprise Edition Admin Only Deprecated in 4.0

| | |
|---|---|
| Description | List all available roles. |
| Signature | dbms.security.listRoles() :: (role :: STRING?, users :: LIST? OF STRING?) |
| Mode | READ |

Table 589. dbms.security.listRolesForUser() Enterprise Edition Deprecated in 4.0

| | |
|---|---|
| Description | List all roles assigned to the specified user. |
| Signature | dbms.security.listRolesForUser(username :: STRING?) :: (value :: STRING?) |
| Mode | READ |

Table 590. dbms.security.listUsers() Admin Only Deprecated in 4.0

| | |
|---|---|
| Description | List all native users. |
| Signature | dbms.security.listUsers() :: (username :: STRING?, roles :: LIST? OF STRING?, flags :: LIST? OF STRING?) |
| Mode | READ |

Table 591. dbms.security.listUsersForRole() Enterprise Edition Admin Only Deprecated in 4.0

| | |
|---|---|
| Description | List all users currently assigned the specified role. |
| Signature | dbms.security.listUsersForRole(roleName :: STRING?) :: (value :: STRING?) |
| Mode | READ |

Table 592. dbms.security.removeRoleFromUser() Enterprise Edition Admin Only Deprecated in 4.0

| | |
|---|---|
| Description | Unassign a role from the user. |
| Signature | dbms.security.removeRoleFromUser(roleName :: STRING?, username :: STRING?) :: VOID |
| Mode | WRITE |

Table 593. dbms.security.suspendUser() Enterprise Edition Admin Only Deprecated in 4.0

| | |
|---|---|
| Description | Suspend the specified user. |
| Signature | dbms.security.suspendUser(username :: STRING?) :: VOID |
| Mode | WRITE |

Table 594. dbms.setConfigValue() Enterprise Edition Admin Only

| | |
|---|---|
| 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. |
| Signature | `dbms.setConfigValue(setting :: STRING?, value :: STRING?) :: VOID` |
| Mode | DBMS |

*Table 595. dbms.showCurrentUser()*

| | |
|---|---|
| Description | Show the current user. |
| Signature | `dbms.showCurrentUser() :: (username :: STRING?, roles :: LIST? OF STRING?, flags :: LIST? OF STRING?)` |
| Mode | DBMS |

*Table 596. dbms.upgrade()* Admin Only

| | |
|---|---|
| Description | Upgrade the system database schema if it is not the current schema. |
| Signature | `dbms.upgrade() :: (status :: STRING?, upgradeResult :: STRING?)` |
| Mode | WRITE |

*Table 597. dbms.upgradeStatus()* Admin Only

| | |
|---|---|
| Description | Report the current status of the system database sub-graph schema. |
| Signature | `dbms.upgradeStatus() :: (status :: STRING?, description :: STRING?, resolution :: STRING?)` |
| Mode | READ |

*Table 598. tx.getMetaData()*

| | |
|---|---|
| Description | Provides attached transaction metadata. |
| Signature | `tx.getMetaData() :: (metadata :: MAP?)` |
| Mode | DBMS |

*Table 599. tx.setMetaData()*

| | |
|---|---|
| Description | Attaches a map of data to the transaction. The data will be printed when listing queries, and inserted into the query log. |
| Signature | `tx.setMetaData(data :: MAP?) :: VOID` |
| Mode | DBMS |

# Appendix B: Tutorials

The following step-by-step tutorials cover common operational tasks or otherwise exemplify working with Neo4j:

- Set up a local Causal Cluster — This tutorial walks through the basics of setting up a Neo4j Causal Cluster.
- Back up and restore a database in Causal Cluster — This tutorial provides a detailed example of how to back up and restore a database in a running Causal Cluster.
- Neo4j Admin import — This tutorial provides detailed examples to illustrate the capabilities of importing data from CSV files with the command `neo4j-admin import`.
- SSO configuration — This tutorial presents examples and solutions to common problems when configuring SSO.
- Set up and use Fabric — This tutorial walks through the basics of setting up and using Neo4j Fabric.

## .B.1. Set up a local Causal Cluster

In this tutorial, you will learn how to deploy a Causal Cluster locally on a single machine.

> **Note:**
>
> Keep in mind that a cluster on a single machine has no fault tolerance and is therefore not suitable for production use.

A typical Causal Cluster consists of three Core instances and three Read Replicas. The Core instances are responsible for keeping the data safe, and the Read Replicas are responsible for scaling the capacity of the cluster. For details on the number of servers required for a Causal Cluster, see Primary servers.

The Core of the Causal Cluster is intended to remain stable over time. The roles within the Core may change as needed, but the Core itself is long-lived and stable.
Read Replicas live at the edge of the cluster and can be brought up and taken down without affecting the Core. They can be added as needed to increase the operational capacity of the cluster as a whole.

For more information about Causal Clustering architecture, configuration, and operation, see Clustering.

### Download Neo4j

You download Neo4j and prepare your local environment.

1. Create a local working directory.
2. Download a copy of the Neo4j Enterprise Edition from the Neo4j download site.
3. Unpack Neo4j in the working directory.

### Set up the Core servers

You create and configure three Core instances.

#### Configure and start the first Core instance

You create and configure the first Core instance.

1. Make a copy of the *neo4j-enterprise-4.4.48* directory and name it *core-01*.
   You have to keep the original directory for setting up the other Core instances and Read Replicas. The *core-01* directory will contain the first Core instance.
2. Open the Neo4j configuration file, *conf/neo4j.conf*, and configure the following settings:

   > **Tip:**
   >
   > If you cannot find the configuration file, see File locations.

   a. Locate and uncomment the setting `dbms.mode=CORE`.
   b. Locate and uncomment the setting `causal_clustering.minimum_core_cluster_size_at_formation=3`.
   c. Locate and uncomment the setting `causal_clustering.minimum_core_cluster_size_at_runtime=3`.
   d. Locate and uncomment the setting `causal_clustering.initial_discovery_members=localhost:5000,localhost:5001,localhost:5002`.
   e. Locate and uncomment the setting `causal_clustering.discovery_listen_address=:5000`.
   f. Locate and uncomment the setting `causal_clustering.transaction_listen_address=:6000`.
   g. Locate and uncomment the setting `causal_clustering.raft_listen_address=:7000`.
   h. Locate and uncomment the setting `dbms.connector.bolt.listen_address=:7687`.
   i. Locate and uncomment the setting `dbms.connector.http.listen_address=:7474`.
   j. Locate and uncomment the setting `dbms.connector.https.listen_address`, and change the value to `:6474`.
   k. Locate and uncomment the setting `dbms.backup.listen_address=0.0.0.0:6362`.
3. Save the file.
4. Open a command-line tool and navigate to *core-01* directory.
5. Run the following command to start *core-01*:

```
core-01$ ./bin/neo4j start
```

### Create and configure the second Core instance

You create and configure the second Core instance.

1. Make a new copy of the *neo4j-enterprise-4.4.48* directory and name it *core-02*.
2. Overwrite *core-02/conf/neo4j.conf* with the just modified *core-01/conf/neo4j.conf*. Then in the new *core-02* directory, open the *conf/neo4j.conf* file and configure the following settings:
   a. Locate the setting `causal_clustering.discovery_listen_address` and change the value to `:5001`.
   b. Locate the setting `causal_clustering.transaction_listen_address` and change the value to `:6001`.

c. Locate the setting `causal_clustering.raft_listen_address` and change the value to `:7001`.
d. Locate the setting `dbms.connector.bolt.listen_address` and change the value to `:7688`.
e. Locate the setting `dbms.connector.http.listen_address` and change the value to `:7475`.
f. Locate the setting `dbms.connector.https.listen_address` and change the value to `:6475`.
g. Locate the setting `dbms.backup.listen_address` and change the value to `0.0.0.0:6363`.

3. Save the file.
4. Open a command-line tool and navigate to *core-02* directory.
5. Run the following command to start *core-02*:

```
core-02$ ./bin/neo4j start
```

### Create and configure the third Core instance

You create and configure the third Core instance.

1. Make a new copy of the *neo4j-enterprise-4.4.48* directory and name it *core-03*.
2. Overwrite *core-03/conf/neo4j.conf* with the just modified *core-02/conf/neo4j.conf*. Then in the new *core-03* directory, open the *conf/neo4j.conf* file and configure the following settings:
   a. Locate the setting `causal_clustering.discovery_listen_address` and change the value to `:5002`.
   b. Locate the setting `causal_clustering.transaction_listen_address` and change the value to `:6002`.
   c. Locate the setting `causal_clustering.raft_listen_address` and change the value to `:7002`.
   d. Locate the setting `dbms.connector.bolt.listen_address` and change the value to `:7689`.
   e. Locate the setting `dbms.connector.http.listen_address` and change the value to `:7476`.
   f. Locate the setting `dbms.connector.https.listen_address` and change the value to `:6476`.
   g. Locate the setting `dbms.backup.listen_address` and change the value to `0.0.0.0:6364`.
3. Save the file.
4. Open a command-line tool and navigate to *core-03* directory.
5. Run the following command to start *core-03*:

```
core-03$ ./bin/neo4j start
```



**Tip:**

**Startup Time**

To follow along with the startup of a server, check the messages in *<instance-home>/logs/neo4j.log*:

- On a Unix system, run the command `tail -n100 logs/neo4j.log`.

- On Windows Server, run `Get-Content .\logs\neo4j.log -Tail 10 -Wait`.

While an instance is joining the cluster, the server may appear unavailable. In the case where an instance is joining a cluster with lots of data, it may take a number of minutes for the new instance to download the data from the cluster and become available.

## Check the status of the cluster

The minimal cluster of three Core servers is operational and is ready to serve requests.

Connect to any of the three Core instances to check the cluster status.

1. Open core-01 at *http://localhost:7474*.
2. Authenticate with the default `neo4j/neo4j` credentials, and set a new password when prompted.
3. Check the status of the cluster by running the following in Neo4j Browser:

```
:sysinfo
```

*Example 141. A cluster of three Core instances.*

| Name | Address | Role | Status | Default | Error |
|---|---|---|---|---|---|
| neo4j | localhost:7689 | follower | online | true | - |
| neo4j | localhost:7688 | follower | online | true | - |
| neo4j | localhost:7687 | leader | online | true | - |
| system | localhost:7689 | follower | online | - | - |
| system | localhost:7688 | follower | online | - | - |
| system | localhost:7687 | leader` | online | - | - |

4. Run the following query to create nodes and relationships.

```
UNWIND range(0, 100) AS value
MERGE (person1:Person {id: value})
MERGE (person2:Person {id: toInteger(100.0 * rand())})
MERGE (person1)-[:FRIENDS]->(person2)
```

5. Open a new tab and point your web browser to a follower, for example, core-02 at *http://localhost:7475*.
6. Authenticate with the credentials you have set up for core-01.
7. Run the following query to verify that the data has been replicated:

```
MATCH path = (person:Person)-[:FRIENDS]-(friend)
RETURN path
```

```
LIMIT 10
```

## Set up the Read Replicas

Because the Read Replicas do not participate in quorum decisions, their configuration is simpler than the configuration of the Core servers.

You configure a Read Replica by setting the address of a Core instance that it can bind to in order to discover the cluster. For details, see Discovery protocol.
After the initial discovery, the Read Replicas can choose a Core instance from which to catch up. For details, see Catchup protocol.

### Configure and start the first Read Replica

You create and configure the first Read Replica.

1. Make a copy of the *neo4j-enterprise-4.4.48* directory and name it *replica-01*.
2. In the new *replica-01* directory, open the *conf/neo4j.conf* file and configure the following settings:
   a. Locate and uncomment the setting `dbms.mode`, and change the value to `READ_REPLICA`.
   b. Locate and uncomment the setting `causal_clustering.initial_discovery_members=localhost:5000,localhost:5001,localhost:5002`.
   c. Locate and uncomment the setting `causal_clustering.discovery_listen_address`, and change the value to `:5003`.
   d. Locate and uncomment the setting `causal_clustering.transaction_listen_address`, and change the value to `:6003`.
   e. Locate and uncomment the setting `dbms.connector.bolt.listen_address`, and change the value to `:7690`.
   f. Locate and uncomment the setting `dbms.connector.http.listen_address`, and change the value to `:7477`.
   g. Locate and uncomment the setting `dbms.connector.https.listen_address`, and change the value to `:6477`.
   h. Locate and uncomment the setting `dbms.backup.listen_address`, and change the values to `0.0.0.0:6365`.
3. Save the file.
4. Open a command-line tool and navigate to *replica-01* directory.
5. Run the following command to start *replica-01*:

```
replica-01$ ./bin/neo4j start
```

### Configure and start the second Read Replica

You create and configure the second Read Replica.

1. Make a new copy of the *neo4j-enterprise-4.4.48* directory and name it *replica-02*.
2. Overwrite *replica-02/conf/neo4j.conf* with the just modified *replica-01/conf/neo4j.conf*. Then in the new *replica-02* directory, open the *conf/neo4j.conf* file and configure the following settings:
   a. Locate the setting `causal_clustering.discovery_listen_address` and change the value to `:5004`.
   b. Locate the setting `causal_clustering.transaction_listen_address` and change the value to `:6004`.
   c. Locate the setting `dbms.connector.bolt.listen_address` and change the value to `:7691`.
   d. Locate the setting `dbms.connector.http.listen_address` and change the value to `:7478`.
   e. Locate the setting `dbms.connector.https.listen_address` and change the value to `:6478`.
   f. Locate the setting `dbms.backup.listen_address` and change the value to `0.0.0.0:6366`.
3. Save the file.
4. Open a command-line tool and navigate to *replica-02* directory.
5. Run the following command to start *replica-02*:

```
replica-02$ ./bin/neo4j start
```

### Configure and start the third Read Replica

You create and configure the third Read Replica.

1. Make a new copy of the *neo4j-enterprise-4.4.48* directory and name it *replica-03*.
2. Overwrite *replica-03/conf/neo4j.conf* with the just modified *replica-02/conf/neo4j.conf*. Then in the new *replica-03* directory, open the *conf/neo4j.conf* file and configure the following settings:
   a. Locate the setting `causal_clustering.discovery_listen_address` and change the value to `:5005`.
   b. Locate the setting `causal_clustering.transaction_listen_address` and change the value to `:6005`.
   c. Locate the setting `dbms.connector.bolt.listen_address` and change the value to `:7692`.
   d. Locate the setting `dbms.connector.http.listen_address` and change the value to `:7479`.
   e. Locate the setting `dbms.connector.https.listen_address` and change the value to `:6479`.
   f. Locate the setting `dbms.backup.listen_address` and change the value to `0.0.0.0:6367`.
3. Save the file.
4. Open a command-line tool and navigate to *replica-03* directory.
5. Run the following command to start *replica-03*:

```
replica-03$ ./bin/neo4j start
```

## Check the status of the cluster

Your cluster of three Core servers and three Read Replicas is operational and is ready to serve requests.

In your core-01 browser, check the cluster status by running the following in Neo4j Browser:

```
:sysinfo
```

*Example 142. A cluster of three Core instances and three Read Replicas.*

| Name | Address | Role | Status | Default | Error |
|---|---|---|---|---|---|
| neo4j | localhost:7689 | follower | online | true | - |
| neo4j | localhost:7688 | follower | online | true | - |
| neo4j | localhost:7687 | leader | online | true | - |
| neo4j | localhost:7692 | read_replica | online | true | - |
| neo4j | localhost:7691 | read_replica | online | true | - |
| neo4j | localhost:76890 | read_replica | online | true | - |
| system | localhost:7689 | follower | online | - | - |
| system | localhost:7688 | follower | online | - | - |
| system | localhost:7687 | leader | online | - | - |
| system | localhost:7692 | read_replica | online | - | - |
| system | localhost:7691 | read_replica | online | - | - |
| system | localhost:7690 | read_replica | online | - | - |

1. Open a new tab and point your web browser to a Read Replica, for example, *replica-01* at *http://localhost:7477*.
2. Login with `neo4j` and the previously set password and **use the `bolt://` schema**.
3. Run the following query to verify that the data has been replicated:

```
MATCH path = (person:Person)-[:FRIENDS]-(friend)
RETURN path
LIMIT 10
```

## .B.2. Back up and restore a database in Causal Cluster

This tutorial provides an example that assumes that you want to restore a database backup, which has users and roles associated with it, in a running Causal Cluster with three core servers. For more information on how to set up a Causal Cluster with three cores, see Set up a local Causal Cluster.



**Note:**

In a Neo4j DBMS, every database is backed up individually. Therefore, it is very important to

plan your backup strategy for each of them. For more detailed information on how to design an appropriate backup strategy for your setup, see Backup and restore.

## Prepare to back up your database

Before you perform the backup, it is good to take a note of the data and metadata of the database that you want to restore. You can use this information to later verify that the restore is successful and to recreate the database users and roles. In this example, the database is called `movies1` and uses the Movie Graph dataset from the Neo4j Browser → Favorites → Example Graphs.

**Note:**

This tutorial uses the Linux or macOS tarball installation. It assumes that your current work directory is the *<neo4j-home>* directory of the tarball installation.

1. In the Neo4j instance, where the database is running, log in to the Cypher Shell command-line console with your credentials. For more information about the Cypher Shell command-line interface (CLI) and how to use it, see Cypher Shell.

```
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. Change the active database to `movies1`.

```
:use movies1
```

3. Run a query to count the number of nodes in the database.

```
MATCH (n) RETURN count(n) AS countNode;
```

```
+-----------+
| countNode |
+-----------+
| 171       |
+-----------+

1 row available after 22 ms, consumed after another 1 ms
```

4. Run a query to count the number of relationships.

```
MATCH (n)-[r]->() RETURN count(r) AS countRelationships;
```

```
+--------------------+
| countRelationships |
+--------------------+
| 253                |
```

```
+--------------------+

1 row available after 29 ms, consumed after another 0 ms
```

5. Change the active database to `system`, and run a query to see if there are any custom roles, associated with this database, and their privileges.

```
SHOW ALL PRIVILEGES AS COMMANDS;
```

```
+-------------------------------------------------------------+
| command                                                     |
+-------------------------------------------------------------+
| "GRANT ACCESS ON HOME DATABASE TO `PUBLIC`"                 |
| "GRANT EXECUTE FUNCTION * ON DBMS TO `PUBLIC`"              |
| "GRANT EXECUTE PROCEDURE * ON DBMS TO `PUBLIC`"             |
| "GRANT ACCESS ON DATABASE * TO `admin`"                     |
| "GRANT MATCH {*} ON GRAPH * NODE * TO `admin`"              |
| "GRANT MATCH {*} ON GRAPH * RELATIONSHIP * TO `admin`"      |
| "GRANT WRITE ON GRAPH * TO `admin`"                         |
| "GRANT NAME MANAGEMENT ON DATABASE * TO `admin`"            |
| "GRANT INDEX MANAGEMENT ON DATABASE * TO `admin`"           |
| "GRANT CONSTRAINT 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 ALL DBMS PRIVILEGES ON DBMS TO `admin`"              |
| "GRANT ACCESS ON DATABASE * TO `architect`"                 |
| "GRANT MATCH {*} ON GRAPH * NODE * TO `architect`"          |
| "GRANT MATCH {*} ON GRAPH * RELATIONSHIP * TO `architect`"  |
| "GRANT WRITE ON GRAPH * TO `architect`"                     |
| "GRANT NAME MANAGEMENT ON DATABASE * TO `architect`"        |
| "GRANT INDEX MANAGEMENT ON DATABASE * TO `architect`"       |
| "GRANT CONSTRAINT MANAGEMENT ON DATABASE * TO `architect`"  |
| "GRANT ACCESS ON DATABASE * TO `publisher`"                 |
| "GRANT MATCH {*} ON GRAPH * NODE * TO `publisher`"          |
| "GRANT MATCH {*} ON GRAPH * RELATIONSHIP * TO `publisher`"  |
| "GRANT WRITE ON GRAPH * TO `publisher`"                     |
| "GRANT NAME MANAGEMENT ON DATABASE * TO `publisher`"        |
| "GRANT ACCESS ON DATABASE * TO `editor`"                    |
| "GRANT MATCH {*} ON GRAPH * NODE * TO `editor`"             |
| "GRANT MATCH {*} ON GRAPH * RELATIONSHIP * TO `editor`"     |
| "GRANT WRITE ON GRAPH * TO `editor`"                        |
| "GRANT ACCESS ON DATABASE * TO `reader`"                    |
| "GRANT MATCH {*} ON GRAPH * NODE * TO `reader`"             |
| "GRANT MATCH {*} ON GRAPH * RELATIONSHIP * TO `reader`"     |
| "GRANT ACCESS ON DATABASE * TO `myrole`"                    |
| "GRANT MATCH {*} ON GRAPH * NODE * TO `myrole`"             |
| "GRANT MATCH {*} ON GRAPH * RELATIONSHIP * TO `myrole`"     |
| "GRANT WRITE ON GRAPH movies1 TO `myrole`"                  |
+-------------------------------------------------------------+

39 rows available after 868 ms, consumed after another 80 ms
```

The result shows that there is one custom role `myrole`.

6. Run a query to see all users associated with this role.

```
SHOW USERS;
```

```
+-------------------------------------------------------------------------+
| user    | roles                | passwordChangeRequired | suspended |
+-------------------------------------------------------------------------+
| "neo4j" | ["admin", "PUBLIC"]  | FALSE                  | FALSE     |
| "user1" | ["myrole", "PUBLIC"] | TRUE                   | FALSE     |
+-------------------------------------------------------------------------+
```

```
2 rows available after 36 ms, consumed after another 2 ms
```

7. Exit the Cypher Shell command-line console.

```
:exit
```

## Back up your database

Now you are ready to back up the database.

Run the following command to back up the database in your targeted folder. If the folder where you want to place your backup does not exist, you have to create it. In this example, it is called */tmp/4.4.48*.

To perform the backup, run the following command:

```
bin/neo4j-admin backup --backup-dir=/tmp/4.4.48 --database=movies1 --include-metadata=all
```

The option `--include-metadata=all` creates a cypher script, which you can later use to restore the database's users, roles, and privileges.

For details on performing a backup and the different command options, see Back up an online 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. If you want to restore the backup as an *additional* database in your DBMS, then you can proceed to Restore the database backup on all cluster members directly.

On one of the cluster members, run the Cypher command `DROP DATABASE` to delete the database that you want to replace. The command is automatically routed to the leader and from there to the other cluster members.



**Warning:**

Dropping a database also deletes the users and roles associated with it.

1. In the Cypher Shell command-line console on one of the cluster members, change the active database to `system`, and run the command `DROP DATABASE` to delete the database that you want to replace. In this example, the database is called `movies`.

```
DROP DATABASE movies;
```

```
0 rows available after 82 ms, consumed after another 0 ms
```



**Warning:**

If you are unable to delete the database (e.g., because Neo4j is not running), you must

run `neo4j-admin unbind` first instead. If you fail to do this, the store files you have (post restore) will be out of sync with the cluster state you have for that database, leading to logical corruption.

2. You can run `SHOW DATABASES` to verify that the database `movies` does not exist.

```
SHOW DATABASES;
```

```
+-----------------------------------------------------------------------------------------------------------
-------------------------+
| name     | aliases | access       | address          | role       | requestedStatus | currentStatus
| error | default | home  |
+-----------------------------------------------------------------------------------------------------------
-------------------------+
| "neo4j"  | []      | "read-write" | "localhost:7687" | "follower" | "online"        | "online"
| ""    | TRUE    | TRUE  |
| "neo4j"  | []      | "read-write" | "localhost:7688" | "leader"   | "online"        | "online"
| ""    | TRUE    | TRUE  |
| "neo4j"  | []      | "read-write" | "localhost:7689" | "follower" | "online"        | "online"
| ""    | TRUE    | TRUE  |
| "system" | []      | "read-write" | "localhost:7687" | "follower" | "online"        | "online"
| ""    | FALSE   | FALSE |
| "system" | []      | "read-write" | "localhost:7688" | "follower" | "online"        | "online"
| ""    | FALSE   | FALSE |
| "system" | []      | "read-write" | "localhost:7689" | "leader"   | "online"        | "online"
| ""    | FALSE   | FALSE |
+-----------------------------------------------------------------------------------------------------------
-------------------------+

6 rows available after 7 ms, consumed after another 3 ms
```

3. Exit the Cypher Shell command-line console.

```
:exit
```

## Restore the database backup on all cluster members

On each cluster member, run the following command to restore the database backup. For details on performing a restore and the different command options, see Restore a database backup.

```
bin/neo4j-admin restore --from=/tmp/4.4.48/movies1 --database=movies1
```

```
You need to execute $HOME/path/to/core-member/data/databases/movies1/tools/metadata_script.cypher. To
execute the file use cypher-shell command with parameter `movies1`
restorePath=/tmp/{neo4j-version-exact}/movies1, restoreStatus=successful, reason=
```

Then, on each cluster member, run the following command to verify that the database `movies1` exists:

```
ls -al data/databases
```

```
total 0
drwxr-xr-x@  7 username  staff   224 17 Nov 15:50 .
drwxr-xr-x@  8 username  staff   256 17 Nov 15:50 ..
drwxr-xr-x  40 username  staff  1280 17 Nov 15:50 movies1
drwxr-xr-x  37 username  staff  1184 16 Nov 15:00 neo4j
-rw-r--r--   1 username  staff     0 16 Nov 15:00 store_lock
```

```
drwxr-xr-x  38 username  staff  1216 16 Nov 15:00 system
```

However, restoring a database does not automatically create it. Therefore, it will not be visible if you do `SHOW DATABASES` in Cypher Shell or Neo4j Browser.

## Create the database backup on the cluster leader

You create the database backup **only on one of your cluster members** using the command `CREATE DATABASE`. The command is automatically routed to the leader and from there to the other cluster members.

1. In the Cypher Shell command-line console on one of the cluster members, use the `system` database and create the database `movies1`.

```
CREATE DATABASE movies1;
```

```
0 rows available after 132 ms, consumed after another 0 ms
```

2. Verify that the `movies1` database is online on all members.

```
SHOW DATABASES;
```

```
+-------------------------------------------------------------------------------------------------------------
--------------------------+
| name      | aliases | access       | address          | role       | requestedStatus | currentStatus
| error | default | home  |
+-------------------------------------------------------------------------------------------------------------
--------------------------+
| "movies1" | []      | "read-write" | "localhost:7688" | "follower" | "online"        | "online"
| ""    | FALSE   | FALSE |
| "movies1" | []      | "read-write" | "localhost:7687" | "leader"   | "online"        | "online"
| ""    | FALSE   | FALSE |
| "movies1" | []      | "read-write" | "localhost:7689" | "follower" | "online"        | "online"
| ""    | FALSE   | FALSE |
| "neo4j"   | []      | "read-write" | "localhost:7688" | "leader"   | "online"        | "online"
| ""    | TRUE    | TRUE  |
| "neo4j"   | []      | "read-write" | "localhost:7687" | "follower" | "online"        | "online"
| ""    | TRUE    | TRUE  |
| "neo4j"   | []      | "read-write" | "localhost:7689" | "follower" | "online"        | "online"
| ""    | TRUE    | TRUE  |
| "system"  | []      | "read-write" | "localhost:7688" | "follower" | "online"        | "online"
| ""    | FALSE   | FALSE |
| "system"  | []      | "read-write" | "localhost:7687" | "leader"   | "online"        | "online"
| ""    | FALSE   | FALSE |
| "system"  | []      | "read-write" | "localhost:7689" | "follower" | "online"        | "online"
| ""    | FALSE   | FALSE |
+-------------------------------------------------------------------------------------------------------------
--------------------------+

9 rows available after 3 ms, consumed after another 1 ms
```

3. Exit the Cypher Shell command-line console.

```
:exit
```

## Recreate the database users and roles

On one of the cluster members, run the restore cypher script *metadata_script.cypher* to create the database and recreate all users and roles of the database backup. The command is automatically routed to the leader and from there to the other cluster members.

Using `cat` (UNIX)

```
cat data/databases/movies1/tools/metadata_script.cypher | bin/cypher-shell -u neo4j -p password -a
localhost:7688 -d system --param "database => 'movies1'"
```

Using `type` (Windows)

```
type data\databases\movies1\tools\metadata_script.cypher | bin\cypher-shell.bat -u neo4j -p password -a
localhost:7688 -d system --param "database => 'movies1'"
```

Follow the steps from 1 to 6 of section Prepare to back up your database to verify that all data and metadata of the database backup have been successfully restored on all cluster members.

## .B.3. Neo4j Admin import

This tutorial provides detailed examples to illustrate the capabilities of importing data from CSV files with the command `neo4j-admin import`.

The `neo4j-admin import` is a command for loading large amounts of data from CSV files into an **unused non-existing** database. Importing data from CSV files with `neo4j-admin import` can only be done once into an **unused** database, it is used for initial graph population only. The `neo4j-admin import` command can be used on the local Neo4j instance even if the instance is running or not.



**Important:**

The `neo4j-admin import` command does not create a database, the command only imports data and make it available for the database. The database must not exist before the `neo4j-admin import` command has been executed, and the database should be created afterwards. 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 import` command.

To create a cluster based on imported data, see Seed a cluster using the import tool.



**Tip:**

**Handy tips:**

- The details of CSV file header format can be found at CSV header format.
- To show available databases, use the Cypher query `SHOW DATABASES` against the `system` database.
- To remove a database, use the Cypher query `DROP DATABASE database_name` against the `system` database.
- To create a database, use the Cypher query `CREATE DATABASE database_name` against the `system` database.

## 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 belongs 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 a shorthand of the actors 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 refering to a node.
2. `:END_ID` — ID refering to a node.
3. `:TYPE` — The relationship type.

In order 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 is defined with the `--nodes` option.
- Paths to relationship data is defined with the `--relationships` option.

The call to `neo4j-admin import` would look like this:

*shell*

```
bin/neo4j-admin import --database=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 is `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 CSV file header format can be found at CSV header format.

### The data

The following CSV files have:

- `--delimiter=";"`
- `--array-delimiter="|"`
- `--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 import` would look like this:

*shell*

```
bin/neo4j-admin import --database=neo4j --delimiter=";" --array-delimiter="|" --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 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 import --database=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 nodes or relationships files. Files within such an input group can be specified with multiple match strings, delimited by `,`, where each match 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 import` would look like this:

*shell*

```
bin/neo4j-admin import --database=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 in order to simplify using the command line 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 numbers inside the file names and will sort them accordingly, without the need for padding with zeros.

*Example 143. 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 import` can be simplified to:

*shell*

```
bin/neo4j-admin import --database=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 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 144. 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 import` would look like this:

*shell*

```
bin/neo4j-admin import --database=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 import`.

*Example 145. 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 import` would look like this:

*shell*

```
bin/neo4j-admin import --database=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 are 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 import` would look like this:

*shell*

```
bin/neo4j-admin import --database=neo4j --nodes=import/movies6.csv --nodes=import/actors6.csv
--relationships=import/roles6.csv
```

## ID space

The import tool makes the assumption that identifiers are unique across node files. This may not be the case for data sets which 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 only unique within files, using ID spaces is a way to ensure uniqueness across all nodes files. See Using ID spaces.

Each node processed by `neo4j-admin 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.

*Example 146. ID space*

To define a ID space `Movie-ID` for `movieId:ID` the syntax will be `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 together.

*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 import` would look like this:

*shell*

```
bin/neo4j-admin import --database=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 `--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 import` would look like this:

*shell*

```
bin/neo4j-admin import --database=neo4j --nodes=import/movies8a.csv --nodes=import/actors8a.csv
--relationships=import/roles8a.csv
```

Since there was a bad relationship in the input data, the import process will fail completely.

Let's see what happens if you append the `--skip-bad-relationships` flag:

*shell*

```
bin/neo4j-admin import --database=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 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 import` would look like this:

*shell*

```
bin/neo4j-admin import --database=neo4j --nodes=import/actors8b.csv
```

Since there was a bad node in the input data, the import process will fail completely.

Let's see what happens if you append the `--skip-duplicate-nodes` flag:

*shell*

```
bin/neo4j-admin import --database=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
```

## .B.4. Set up and use Fabric

Neo4j Fabric is a tool for storing and retrieving data in multiple databases, located in one or many Neo4j DBMS(s), with a single Cypher query.

In this tutorial, you will learn how to:

- Model your data for Fabric

- Configure Fabric with three databases
- Import data in your databases
- Retrieve data with a single Cypher query

> **Tip:**
>
> For more information on how to manage multiple active databases in Neo4j, see Manage databases.
> For more details on Fabric, see Fabric.

## Model your data for Fabric

### Northwind data



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.

> **Tip:**
>
> For more information on how Northwind (a relational dataset) is modeled into a graph, run `:guide northwind-graph` in Neo4j Browser to play the built-in guide Northwind Graph. See the Neo4j Browser documentation.

### The model

The Northwind graph model consists of the following data:

- Node labels
  - `:Product`
  - `:Category`
  - `:Supplier`
  - `:Order`
  - `:Customer`
- Relationship types
  - `:SUPPLIES`
  - `:PART_OF`

- `:ORDERS`
- `:PURCHASED`

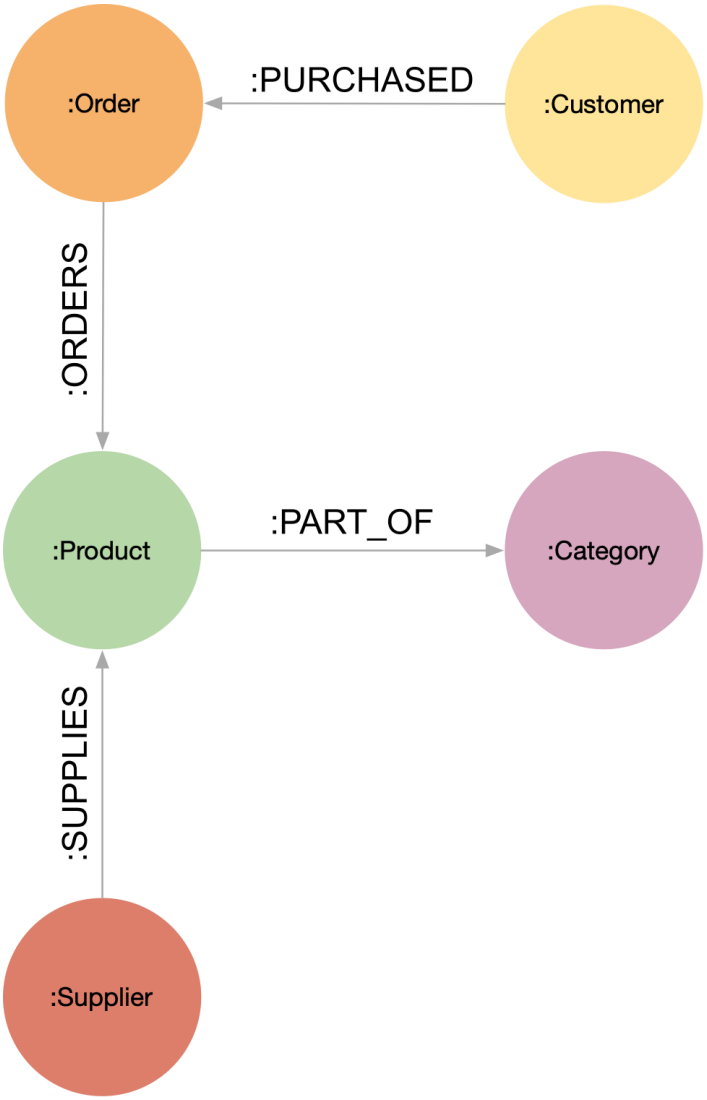



*Figure 16. The Northwind data model*

## Remodeling the Northwind dataset

In this scenario, you imagine 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`.

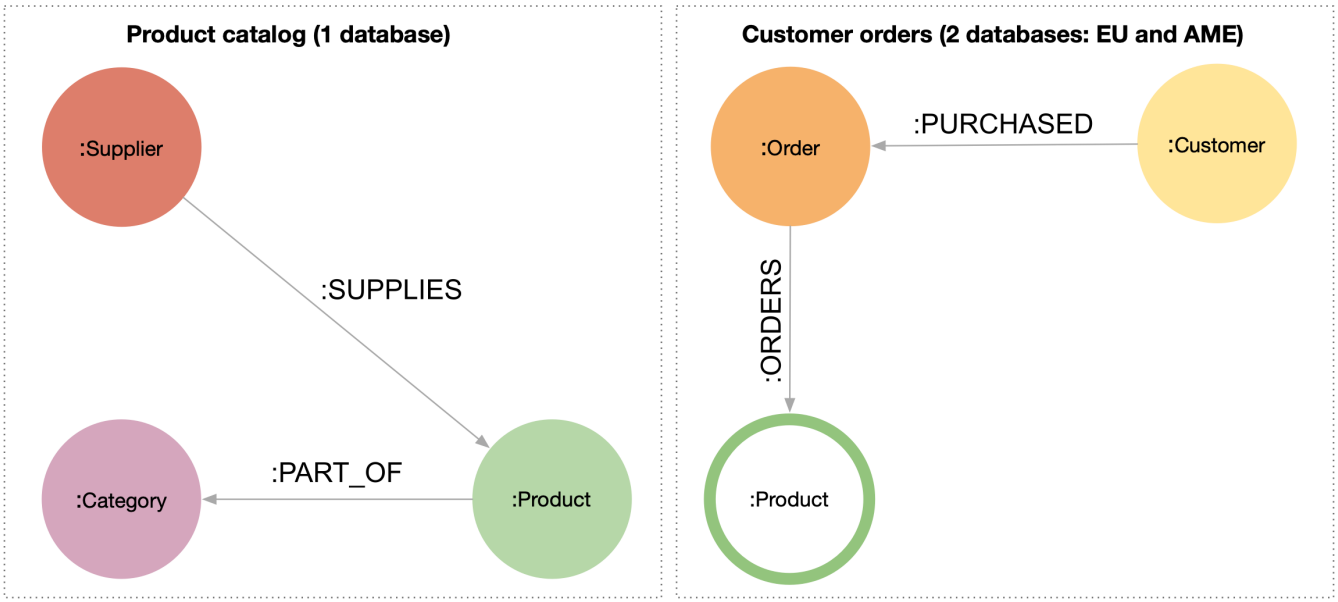



*Figure 17. The new data model*

### Data Federation

This way, the Product and Customer data are in two **disjointed 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.

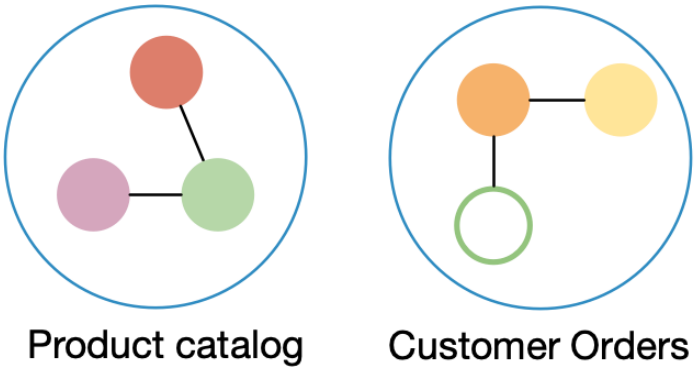



*Figure 18. Data Federation*

### 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*.

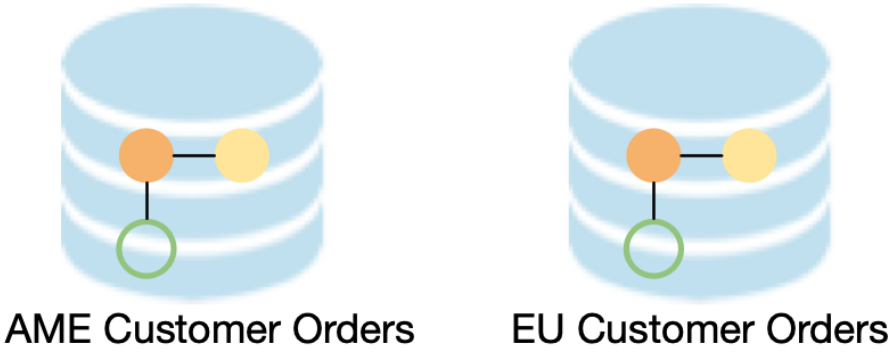



*Figure 19. Data Sharding*

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.

## Configure Fabric with three databases

Now that you have a new multi-database model defined, you can start to configure the Fabric infrastructure.

> **Note:**
>
> This tutorial uses the Linux or macOS tarball installation. It assumes that your current work directory is the *<neo4j-home>* directory of the tarball installation.

### Create three databases

You need three databases: `db0` for the Product catalog, `db1` for the EU customer data, and `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. For more information about the Cypher Shell command-line interface (CLI) and how to use it, see Cypher Shell.

```
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.
```

4. Run the command `SHOW DATABASES` to list all available databases.

```
SHOW DATABASES;
```

```
+-------------------------------------------------------------------------------------------------------
--------------------------+
| name     | aliases | access       | address          | role         | requestedStatus |
currentStatus | error | default | home  |
+-------------------------------------------------------------------------------------------------------
--------------------------+
| "neo4j"  | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""    | TRUE    | TRUE  |
| "system" | []      | "read-write" | "localhost:7687" | "standalone" | "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 14 ms, consumed after another 0 ms
```

```
CREATE DATABASE db2;
```

```
0 rows available after 10 ms, consumed after another 0 ms
```

6. Again run the command `SHOW DATABASES` to verify that the new databases have been created.

```
SHOW DATABASES;
```

```
+-------------------------------------------------------------------------------------------------------
--------------------------+
| name     | aliases | access       | address          | role         | requestedStatus |
currentStatus | error | default | home  |
+-------------------------------------------------------------------------------------------------------
--------------------------+
```

```
| "db0"    | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""    | FALSE   | FALSE |
| "db1"    | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""    | FALSE   | FALSE |
| "db2"    | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""    | FALSE   | FALSE |
| "neo4j"  | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""    | TRUE    | TRUE  |
| "system" | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""    | FALSE   | FALSE |
+-------------------------------------------------------------------------------------------------------
---------------------------+

5 rows available after 8 ms, consumed after another 7 ms
```

7. Exit the Cypher Shell command-line tool.

```
:exit
```

## Configure Fabric

You set up Fabric by configuring the fabric database and the graph names and IDs in the *neo4j.conf* file. In this example, the Fabric database is called `fabricnw`.

1. Navigate to the *<neo4j-home>/conf/* folder and open the *neo4j.conf* file.
2. Add the following lines and save it.

```
#*****************************************************************
# Fabric tutorial
#*****************************************************************

fabric.database.name=fabricnw

fabric.graph.0.uri=neo4j://localhost:7687
fabric.graph.0.name=product
fabric.graph.0.database=db0

fabric.graph.1.uri=neo4j://localhost:7687
fabric.graph.1.name=customerEU
fabric.graph.1.database=db1

fabric.graph.2.uri=neo4j://localhost:7687
fabric.graph.2.name=customerAME
fabric.graph.2.database=db2
```

3. Navigate back to the *<neo4j-home>* folder and restart the Neo4j DBMS.

```
bin/neo4j restart
```

4. Connect to the Neo4j DBMS using `cypher-shell` and your credentials.

```
bin/cypher-shell -u neo4j -p your-password
```

5. Run the command `SHOW DATABASES` to verify that the Fabric database has been configured and is `online`.

```
SHOW DATABASES;
```

```
+-----------------------------------------------------------------------------------------------------------
-----------------------------+
| name       | aliases | access       | address          | role         | requestedStatus |
currentStatus | error | default | home  |
+-----------------------------------------------------------------------------------------------------------
-----------------------------+
| "db0"      | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""    | FALSE   | FALSE |
| "db1"      | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""    | FALSE   | FALSE |
| "db2"      | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""    | FALSE   | FALSE |
| "fabricnw" | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""    | FALSE   | FALSE |
| "neo4j"    | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""    | TRUE    | TRUE  |
| "system"   | []      | "read-write" | "localhost:7687" | "standalone" | "online"        | "online"
| ""    | FALSE   | FALSE |
+-----------------------------------------------------------------------------------------------------------
-----------------------------+

6 rows available after 242 ms, consumed after another 18 ms
```

## Import data in your databases

You can use the command `LOAD CSV WITH HEADERS FROM` to import data in 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 "http://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 "http://data.neo4j.com/northwind/categories.csv" AS row
CREATE (n:Category)
SET n = row;

LOAD CSV WITH HEADERS FROM "http://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 "http://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 "http://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 "http://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 "http://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 "http://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 "http://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 "http://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 "http://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
```

## Retrieve data with a single Cypher query

Fabric allows you to retrieve data from all your databases with a single Cypher query.

As the databases `db0`, `db1`, `db2` in this tutorial are part of the same Neo4j DBMS, you can also access them directly, using their database names. This is especially useful when you want to set up Fabric locally for development or testing purposes. In this case, you only have to add `fabric.database.name=fabricnw` to the *neo4j.conf* file, and use queries as the following one.

```
:use fabricnw
```

```
USE db1
MATCH (c:Customer)
WHERE c.customerID STARTS WITH 'A'
RETURN c.customerID AS name, c.country AS country
  UNION
USE db2
MATCH (c:Customer)
WHERE c.customerID STARTS WITH 'A'
RETURN c.customerID AS name, c.country AS country
LIMIT 5;
```

```
+---------------------+
| name    | country   |
+---------------------+
| "ALFKI" | "Germany" |
| "AROUT" | "UK"      |
| "ANATR" | "Mexico"  |
| "ANTON" | "Mexico"  |
+---------------------+

4 rows available after 404 ms, consumed after another 1 ms
```

However, if your databases `db0`, `db1`, `db2` are located in other Neo4j DBMSs, on completely different servers for example, then you must update the URI settings to connect to them.

In this tutorial, you will try the Fabric capabilities as if the data is deployed on different servers.

### Query a single database

You can retrieve data from a single database by using the cypher clause `USE` and the name of the Fabric graph. When querying a single database, you do not have to change the active database to Fabric.

```
USE fabricnw.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 Fabric to query both shards and get customers whose name starts with A.

When you want to retrieve data from multiple databases, you have to change the active database to `fabricnw`.

```
:use fabricnw
```

```
USE fabricnw.customerAME
MATCH (c:Customer)
WHERE c.customerID STARTS WITH 'A'
RETURN c.customerID AS name, c.country AS country
  UNION
USE fabricnw.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 Fabric idiom:

```
UNWIND [1,2]AS gid
CALL {
    USE fabricnw.graph(gid)
    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

Finally, a more complex query that uses all 3 databases to find all customers who have bought discontinued products in the Meat/Poultry category.

```
CALL {
USE fabricnw.product
MATCH (p:Product{discontinued:true})-[:PART_OF]->(c:Category{categoryName:'Meat/Poultry'})
              RETURN COLLECT(p.productID) AS pids
}
WITH *, [g IN fabricnw.graphIds() WHERE g<>0] AS gids
UNWIND gids AS gid
CALL {
    USE fabricnw.graph(gid)
    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
```



First, `fabricnw` calls 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.

## The end

You have just learned how to store and retrieve data from multiple databases using a single Cypher query. For more details on the Neo4j Fabric, see Fabric.

# .B.5. Neo4j Single Sign-On (SSO) configuration

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/browser/* (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:

<code>http://localhost:7474/browser/?idp_id={provider}&auth_flow_step=redirect_uri</code>

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.



**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.



**Note:**

Currently it is not possible to login to Cypher Shell using SSO authentication and authorization.

## Okta

This example shows how to configure Okta for authentication and authorization using access tokens.

1. Configure the client with the appropriate redirect URI. You can skip the group assignments in this step:

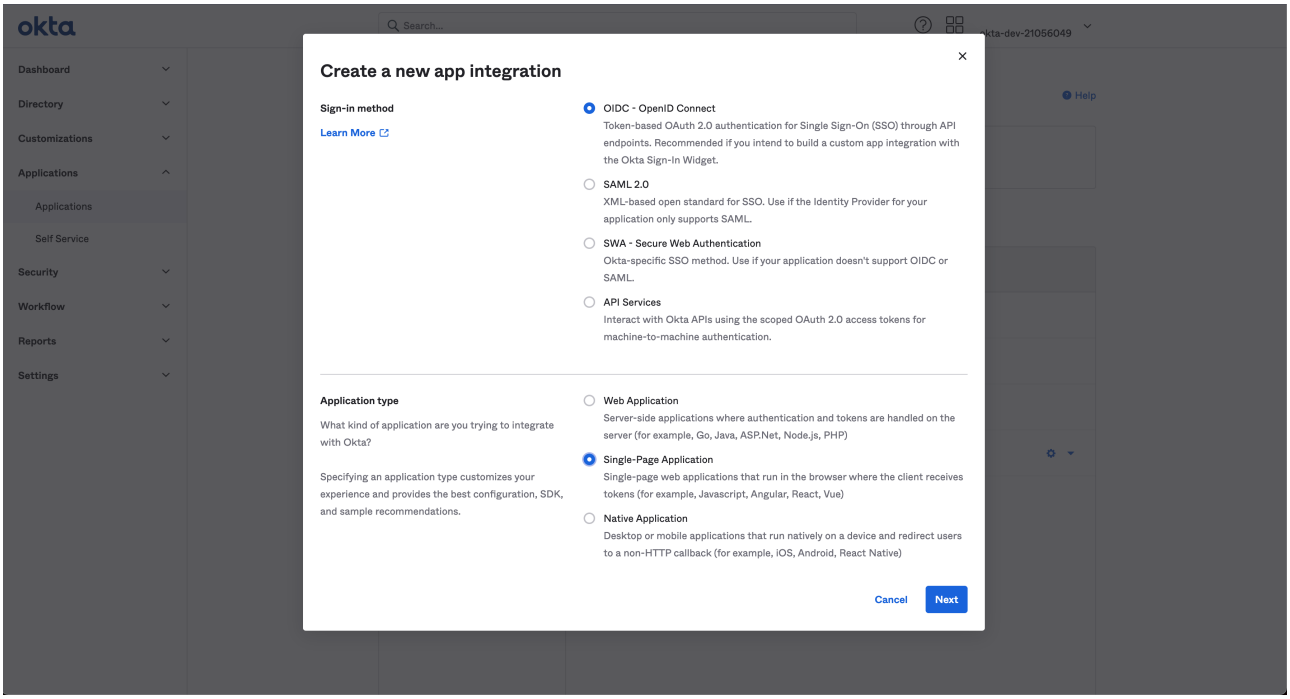



*Figure 20. Okta OIDC client creation*

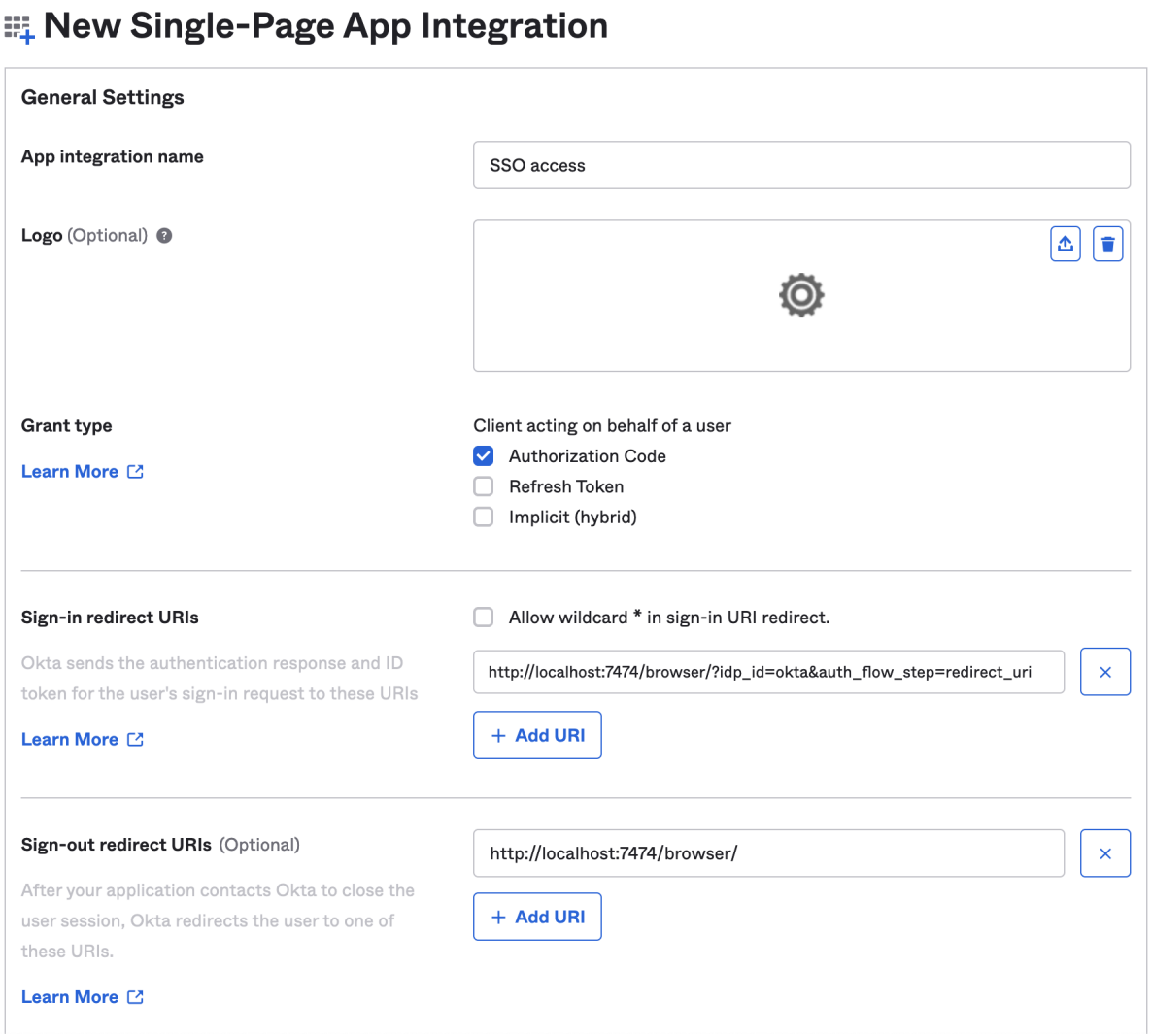



*Figure 21. Okta OIDC client configuration*

2. Take note of the Client ID and the Okta domain. You will need them later when configuring the Okta parameters and the Well-known OpenID Connect endpoint in the *neo4j.conf* file:

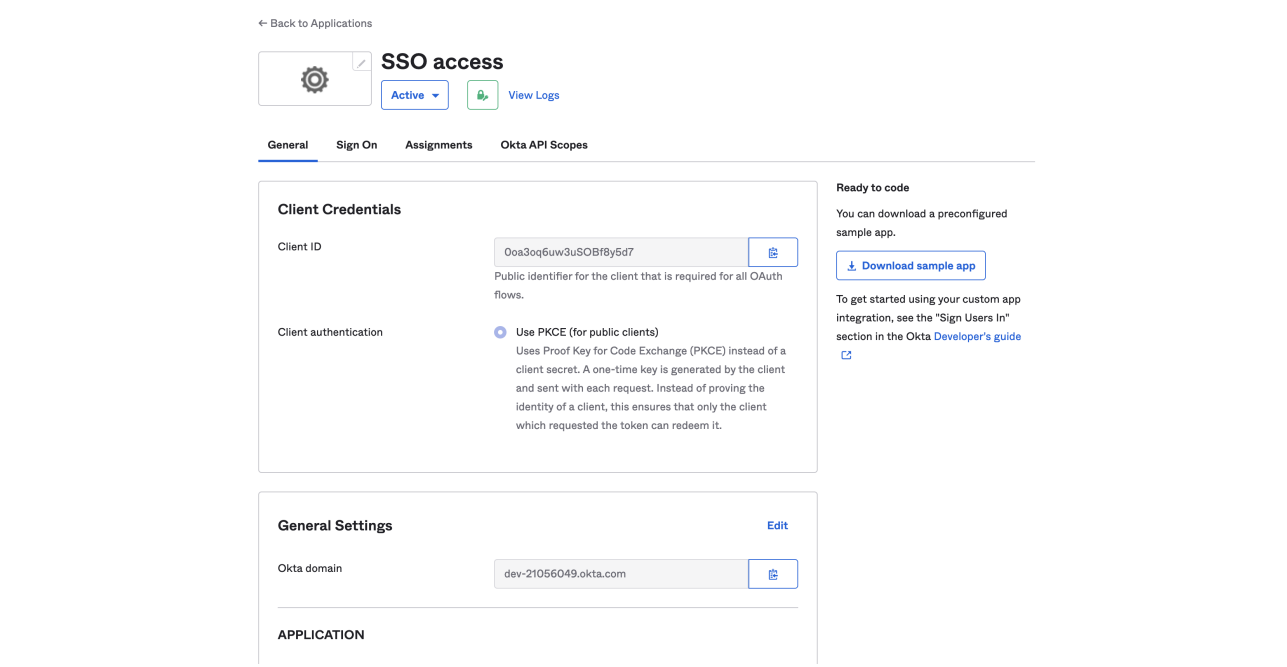



*Figure 22. Okta OIDC client configuration*

3. Create groups in Okta, assign users to them (the user can be added to a group either on user creation or editing the group), and map them in the `neo4j.conf` to native groups:

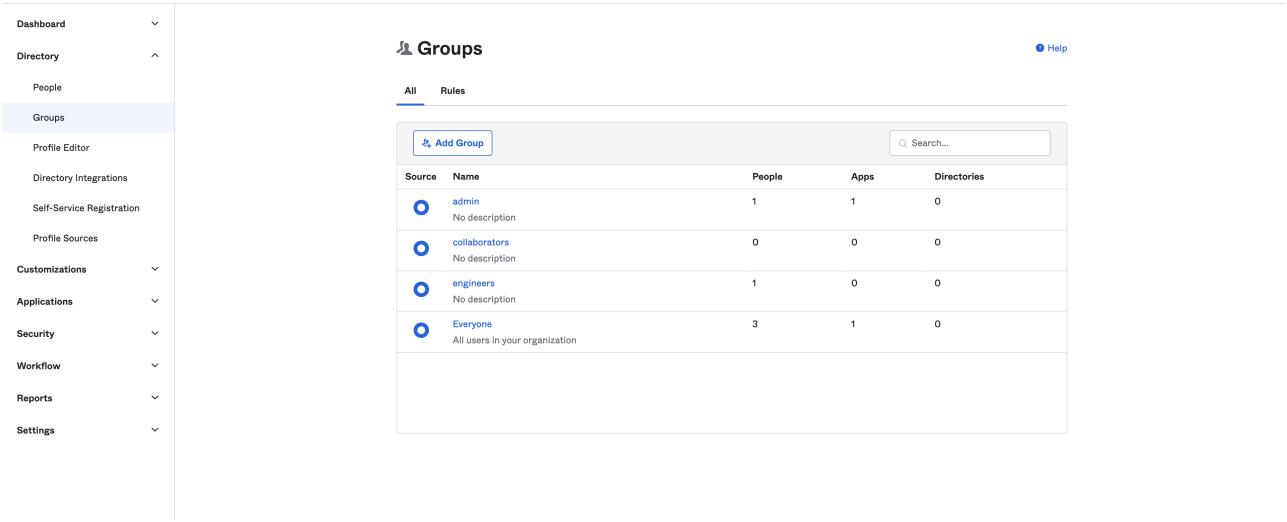



*Figure 23. Okta OIDC server groups*

4. Configure the default authorization server (the one that shows `api://default` as audience) to return the `groups` claim in access tokens:

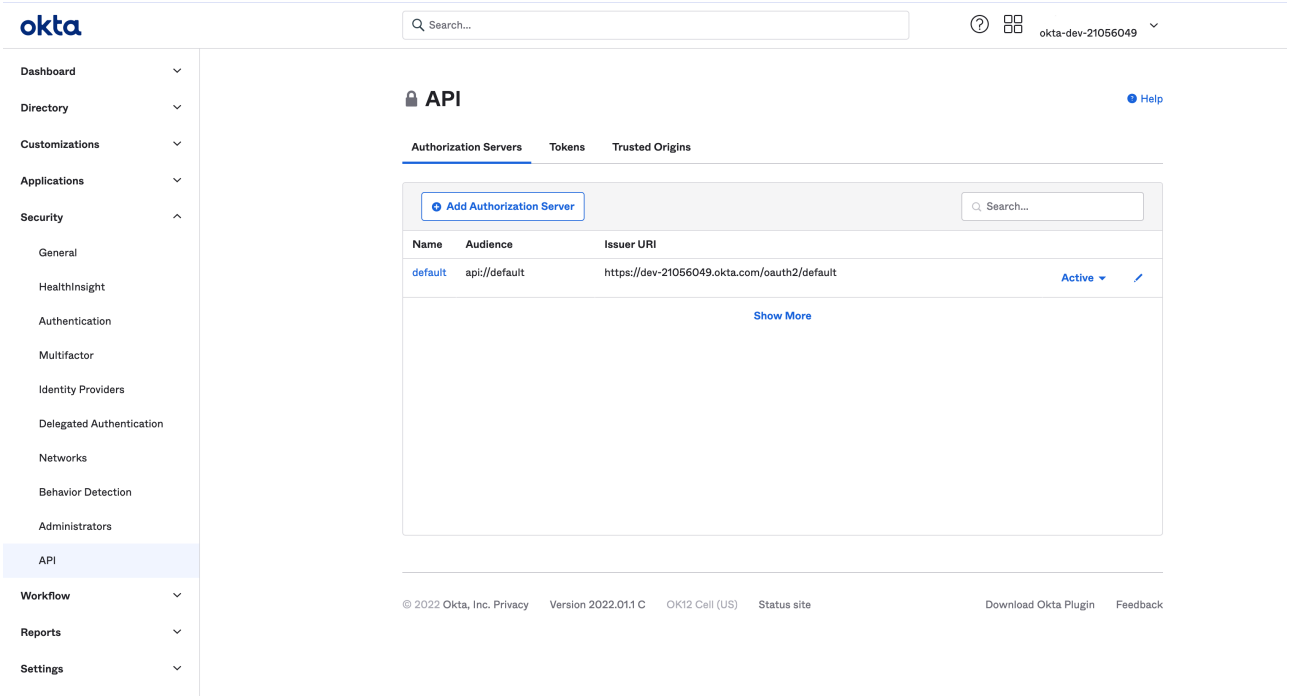



*Figure 24. Okta OIDC authorization server*

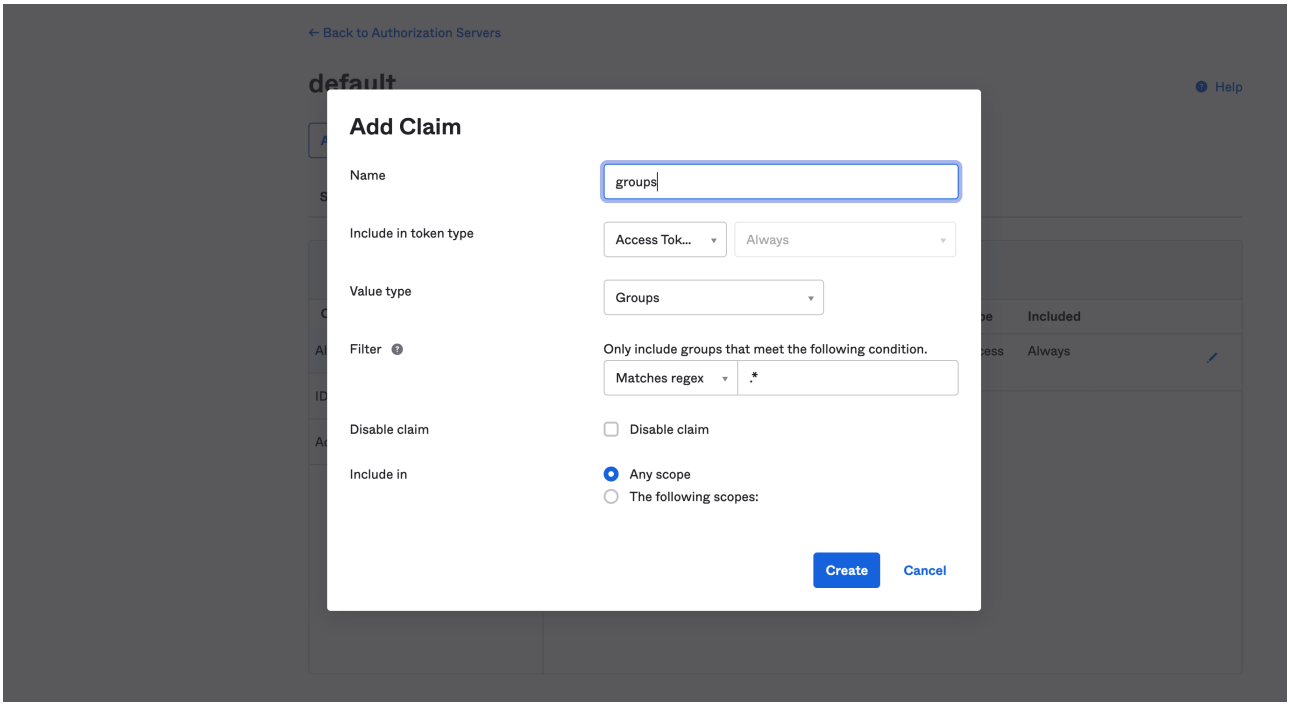



*Figure 25. Okta OIDC server claims*

5. 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-21056049.okta.com/oauth2/default/.well-known/openid-configuration
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=0oa3oq6uw3uSOBf8y5d7;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.

6. Log in with your Okta SSO credentials using the email of an `engineer` role user that results in an `admin` role in the database:

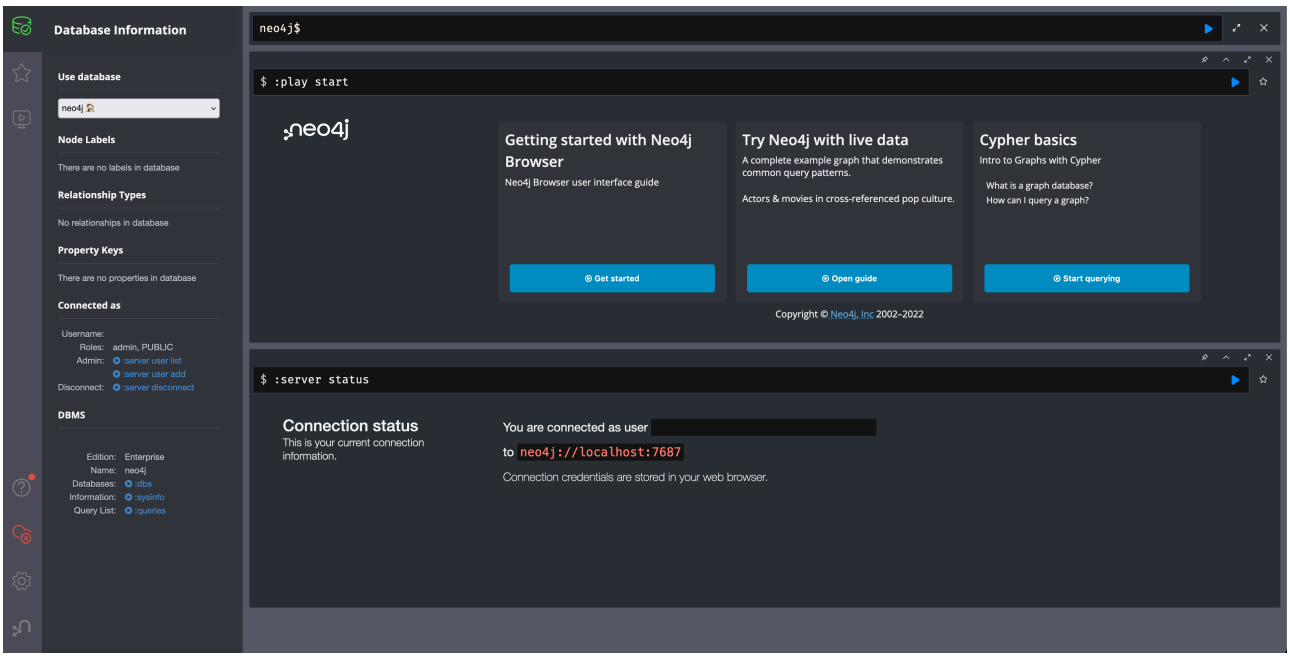



*Figure 26. Okta OIDC successful login*

## Microsoft Entra ID (formerly Azure Active Directory)

### Access token

This example shows how to configure Entra ID for authentication and authorization using an access token.

1. Set parameters to be `access_token`:

```
dbms.security.oidc.azure.config=principal=unique_name;code_challenge_method=S256;token_type_principal=
access_token;token_type_authentication=access_token
```

2. Add the following parameter:

```
dbms.security.oidc.azure.token_endpoint=https://login.microsoftonline.com/54e85725-ed2a-49a4-a19e-
11c8d29f9a0f/oauth2/v2.0/token
```

> **Note:**
>
> The GUID is the directory (tenant) ID. You can find it on the app registration page:

3. Include the issuer:

```
dbms.security.oidc.azure.issuer=https://sts.windows.net/54e85725-ed2a-49a4-a19e-11c8d29f9a0f/
```

> **Note:**
>
> As previously mentioned, the GUID here is also the directory (tenant) ID. Make sure you add the trailing slash ( `/` ) at the end or this operation might fail.

4. Go to the "Expose an API" tab and click "Add a Scope" to include the following statement:

```
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
```

5. Add the value in the scope column to the scopes in the configuration. Note that the audience parameter for access tokens are typically set with` api://` at the front.

### 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:

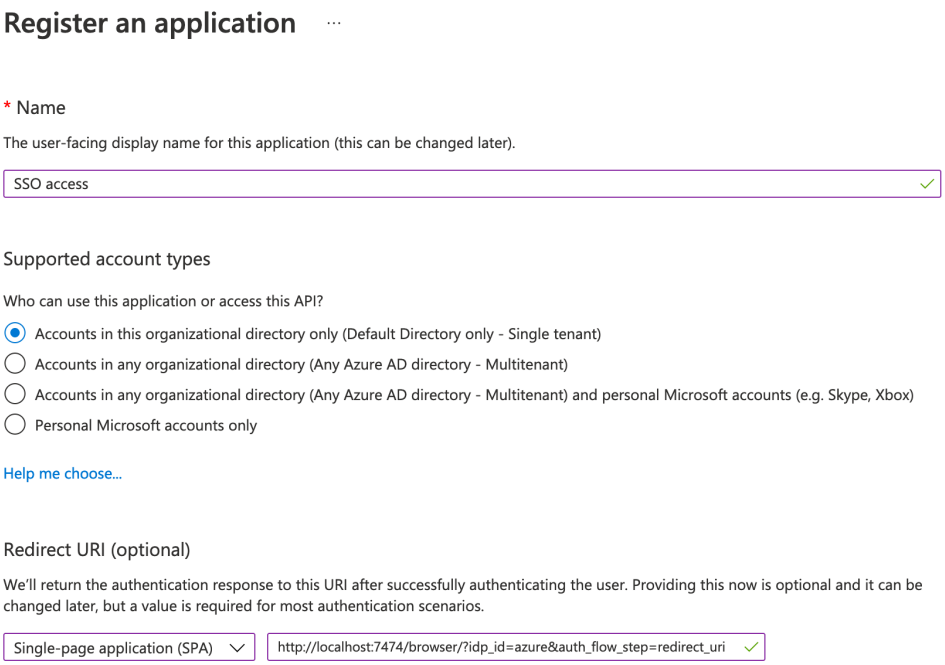



*Figure 27. Entra OIDC client creation*

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. After the successful app creation, on the app's **Overview** page, find the Application (client) ID value. Use it to configure the following properties in the *neo4j.conf* file.

```
dbms.security.oidc.azure.audience=c2830ff5-86d9-4e38-8a2b-9efad6f3d06d
dbms.security.oidc.azure.params=client_id=c2830ff5-86d9-4e38-8a2b-
9efad6f3d06d;response_type=code;scope=openid profile email
```

2. Navigate to **Endpoints**, to find the OpenID Connect metadata document. Use it to configure the `well_known_discovery_uri` in the *neo4j.conf* file.

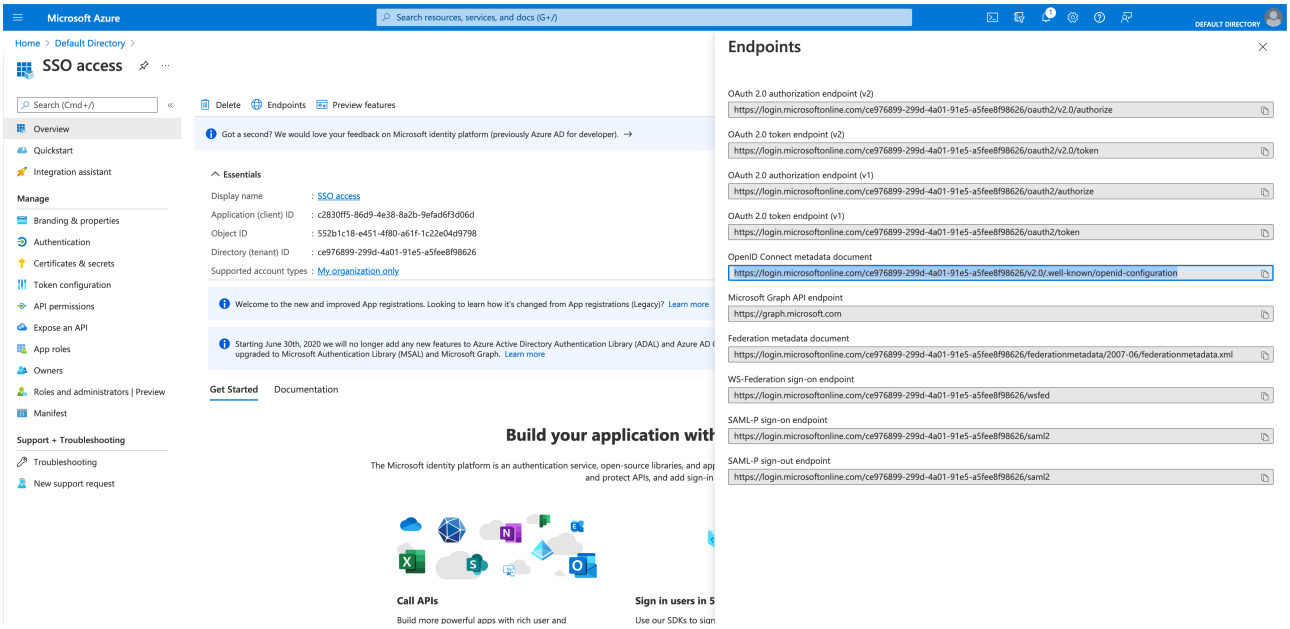



*Figure 28. Entra OIDC client config*

```
dbms.security.oidc.azure.well_known_discovery_uri=https://login.microsoftonline.com/ce976899-299d-4a01-91e5-a5fee8f98626/v2.0/.well-known/openid-configuration
```

3. Configure Neo4j to use Entra ID authentication by configuring the following settings 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
```

4. 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 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 Neo4j settings.

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. Configure the server to return the Group Object IDs in the JWT identity tokens. To do this, set `groupMembershipClaims` to `SecurityGroup` in the Manifest of the registered application:

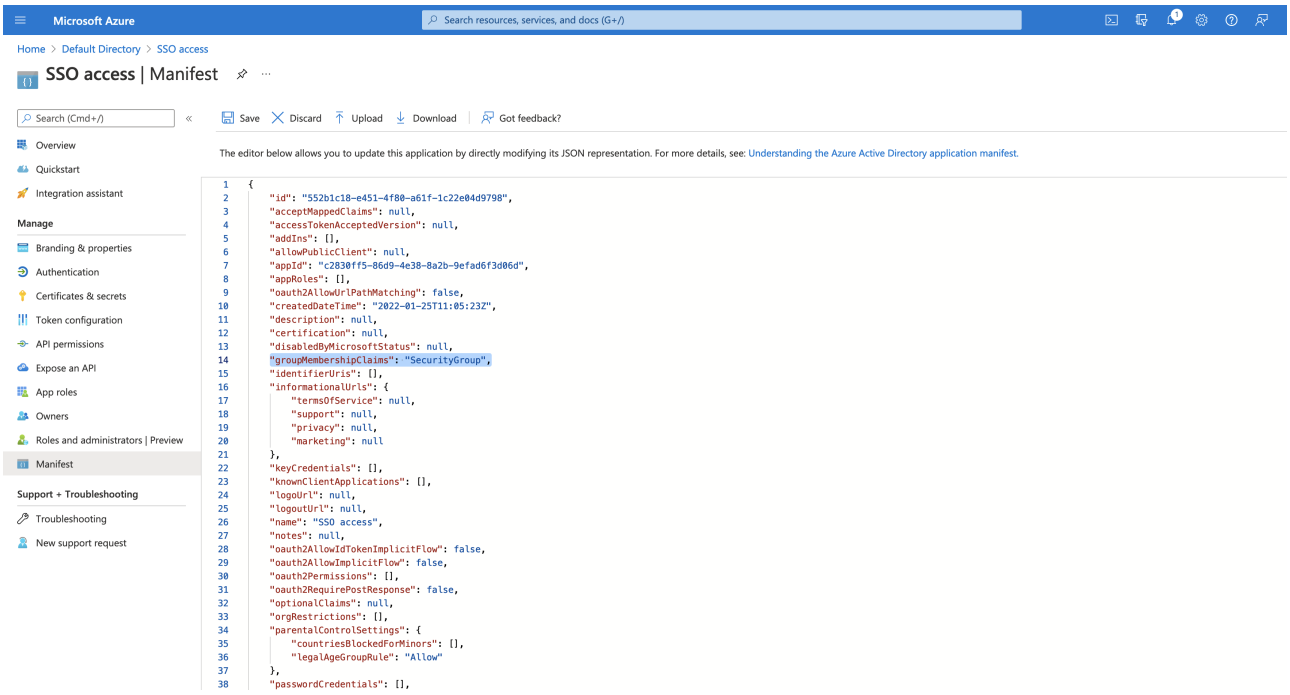



*Figure 29. Entra OIDC server claims*

2. Create groups in the Entra AD console and assign users to them. Take note of the Object Id column. In the next step, you must map these to user roles in the Neo4j settings.

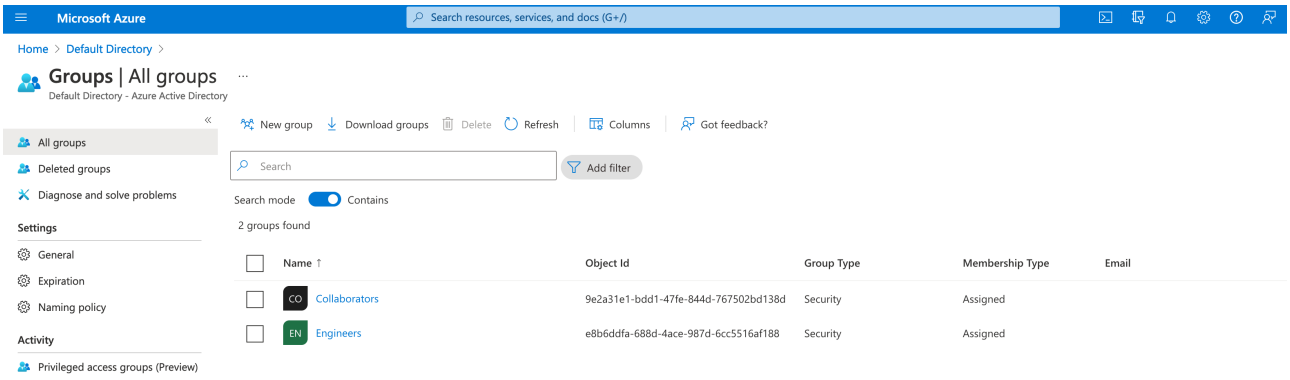



*Figure 30. Entra OIDC server groups*

3. 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
```

4. Configure Neo4j to use the `groups` field from the JWT token.

```
dbms.security.oidc.azure.claims.groups=groups
```

Using Entra ID App Roles

1. On the app's home page, navigate to **App roles** and add the Neo4j roles to the Microsoft Entra ID.

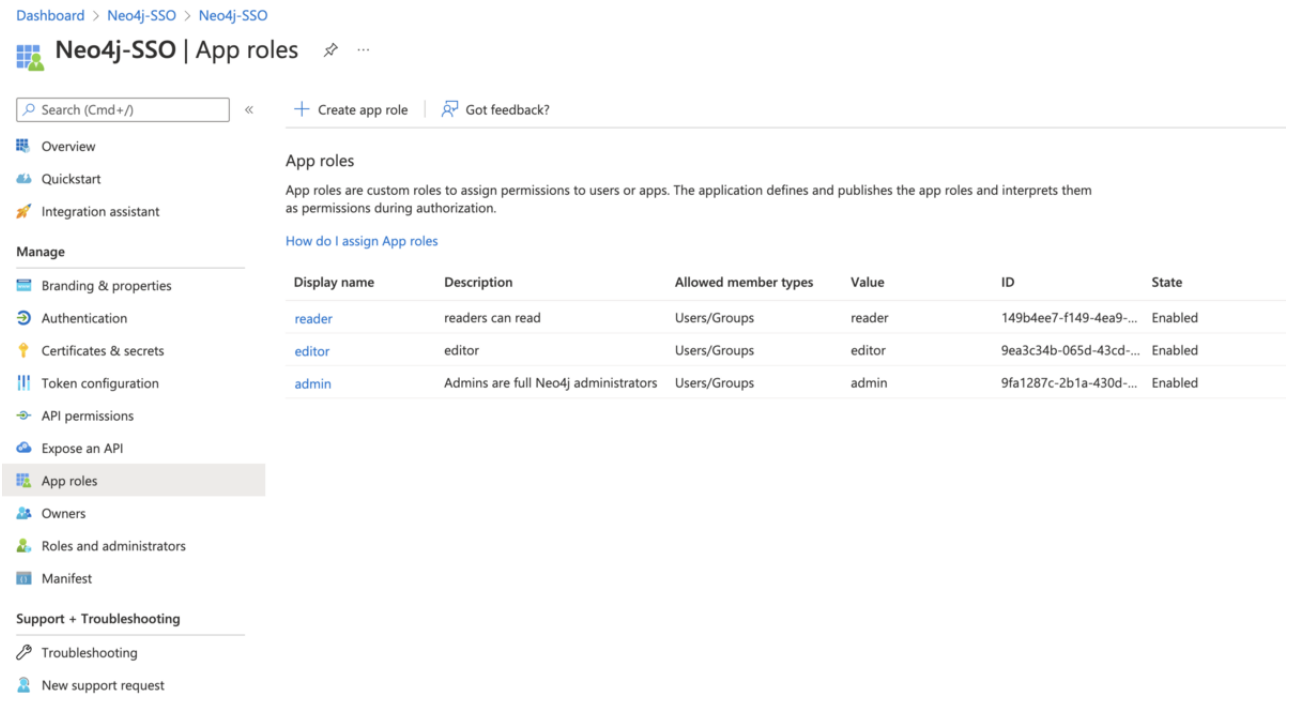



*Figure 31. Entra OIDC app roles config*

2. The **Value** column in the App roles config must either correspond to Neo4j Roles or be mapped 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
```

3. Configure Neo4j to use the `roles` field from the JWT token.

```
dbms.security.oidc.azure.claims.groups=roles
```

> **Note:**
>
> Remember to always set an email in contact info, otherwise login may not work properly at this stage.

## Google

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:

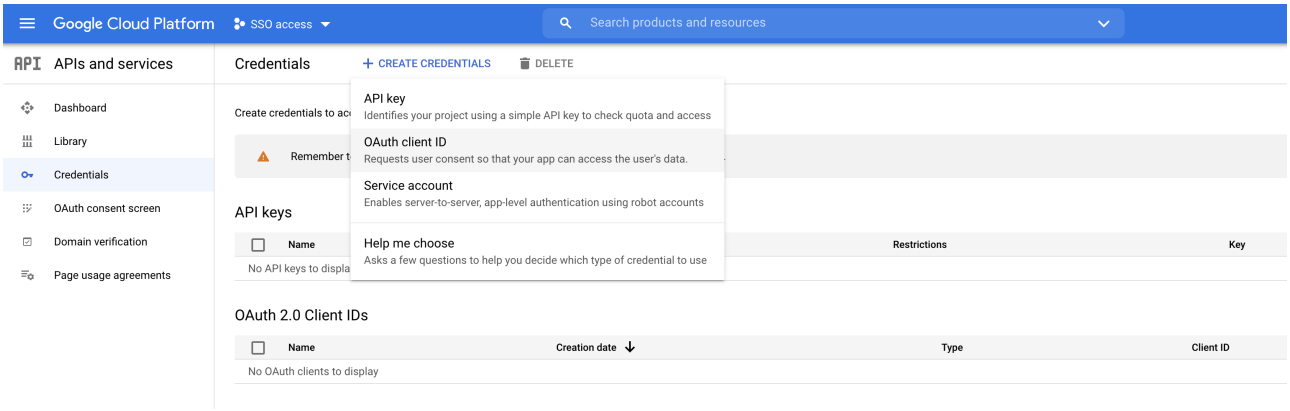



*Figure 32. Google OIDC client creation*

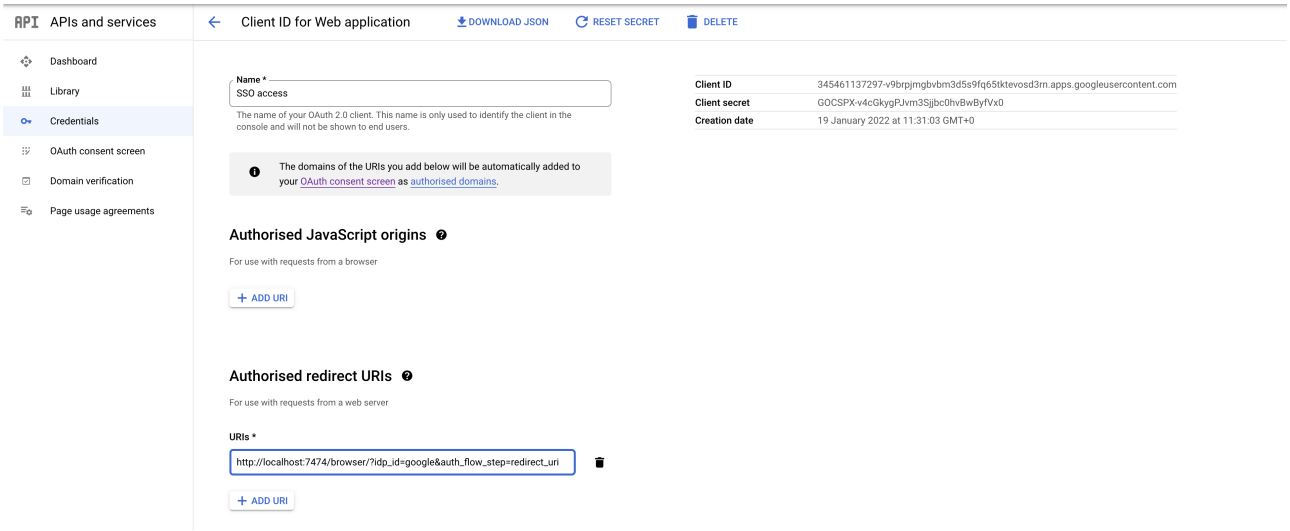



*Figure 33. Google OIDC client configuration*

> **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. The role assigned to the email used to log in with SSO, in this case, `alice@neo4j-test.com`, must have `GRANT ROLE` permissions in the database (`native` authentication temporarily enabled):

```
CREATE USER `alice@neo4j-test.com` SET PASSWORD 'secretpassword';
GRANT ROLE admin to `alice@neo4j-test.com`;
```

3. Configure Neo4j to use Google authentication by configuring the following settings 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
```

4. Log in with your Google SSO credentials using the email address and get the `admin` role when doing so:

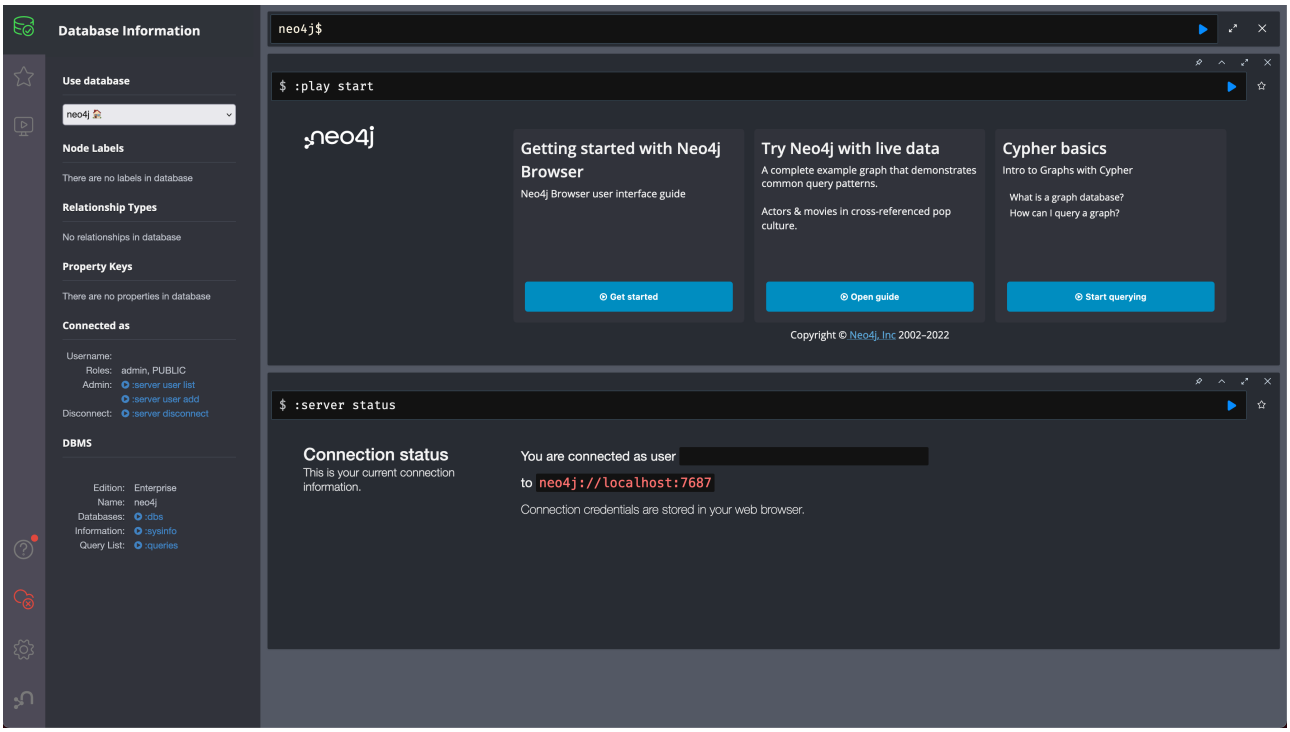



*Figure 34. Entra OIDC successful login*

> **Note:**
>
> The native authentication is disabled to prevent someone from logging in to *alice@neo4j-test.com* with the set password.

## 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.

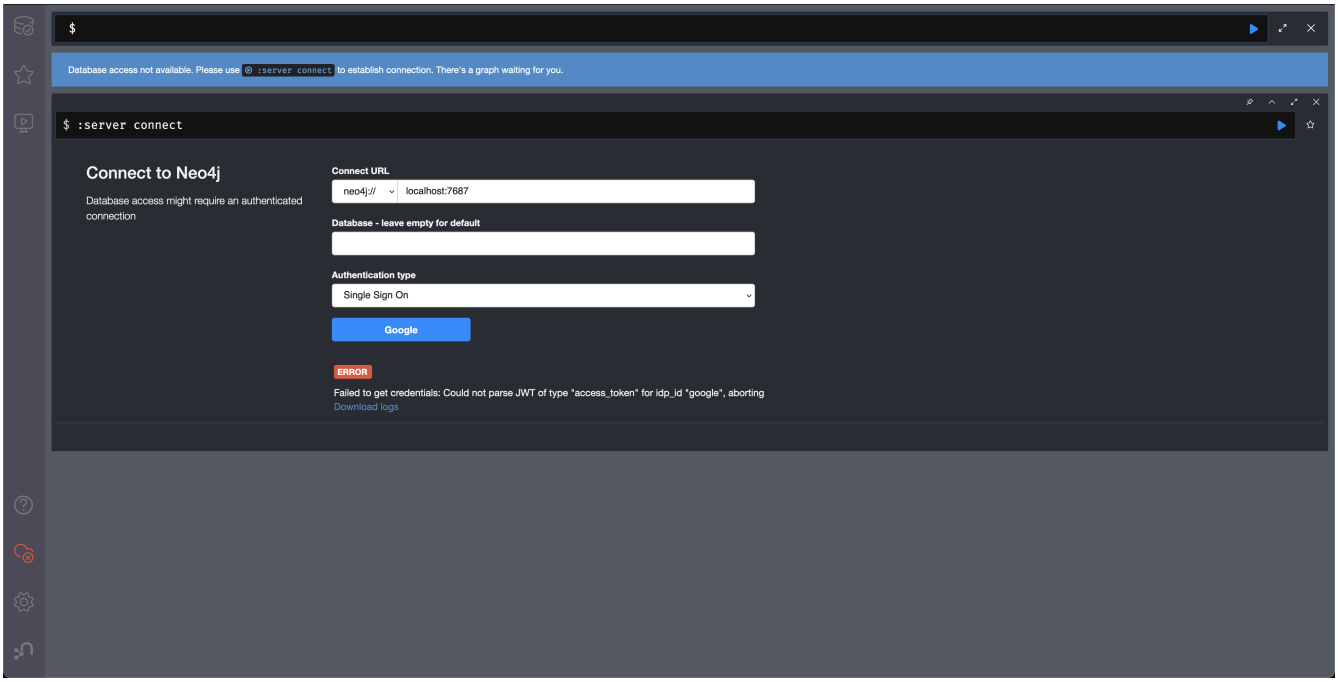



*Figure 35. Failed to parse JWT of type access_token*

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`.

**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:

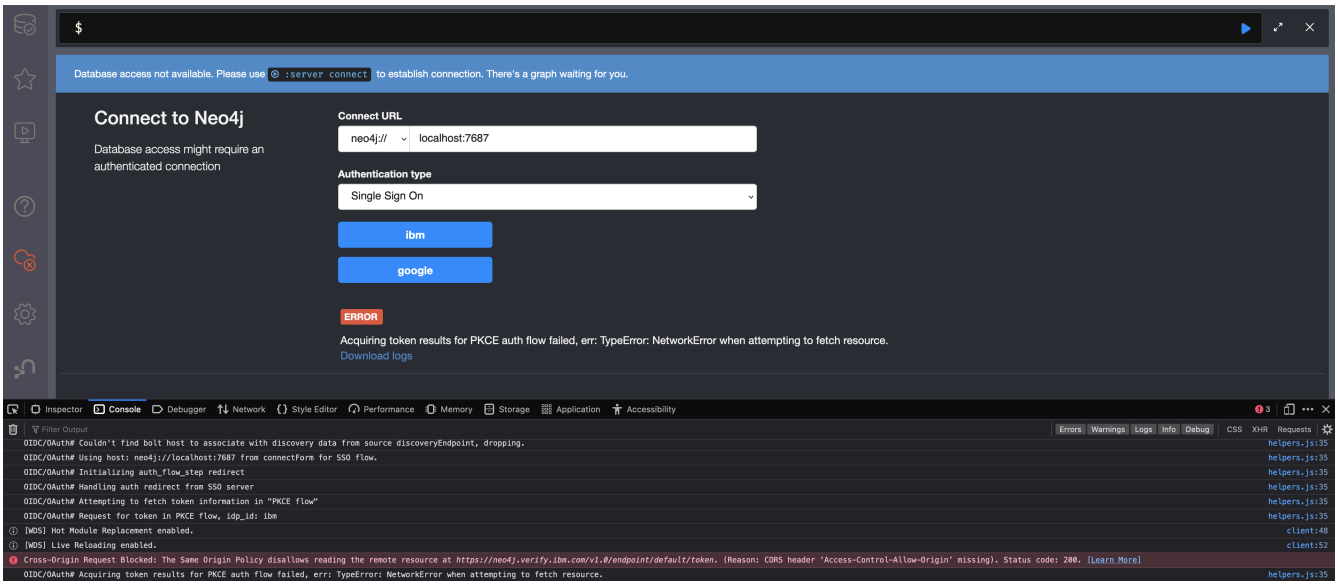



*Figure 36. CORS error*

The solution involves adding the redirect domain to the list of allowed domains in the provider (in this case, `localhost:8080`):

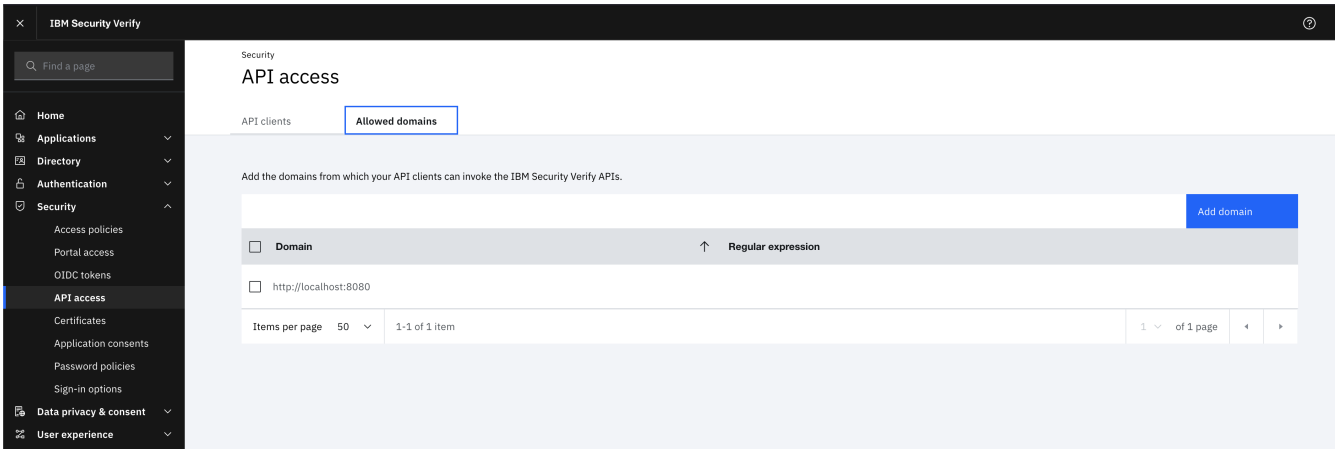



*Figure 37. CORS error solution allowing the redirect domain on the provider*

# Appendix C: Advanced Causal Clustering

This section includes information about advanced deployments and configuration options for multi-data center operations.

- Causal Clustering lifecycle — A walk-through of the lifecycle of a cluster.
- Multi-data center — Overview of the multi-data center section.

    - Licensing for multi-data center operations — Information about licensing for multi-data center operations.
    - Multi-data center design — Patterns for multi-data center deployments.
    - Multi-data center operations — Configuration options for multi-data center deployments.
    - Multi-data center load balancing — Configuration options for making client applications aware of multi-data center topologies.
    - Data center disaster recovery — How to recover a cluster to full working capability after data center loss.
- Embedded usage — How to embed a Neo4j Causal Cluster in your application.

For details on the configuration and operation of a Neo4j Causal Cluster, see Clustering.

For descriptions of settings related to running a Neo4j Causal Cluster, see Settings reference.

## .C.1. Causal Clustering lifecycle

This section includes:

- Introduction
- Discovery protocol
- Core membership
- Read Replica membership
- Transacting via the Raft protocol
- Catchup protocol
- Read Replica shutdown
- Core shutdown

### Introduction

In this section we will develop some deeper knowledge of how the cluster operates. By developing our understanding of how the cluster works we will be better equipped to design, deploy, and troubleshoot our production systems.

Our in-depth tour will follow the lifecycle of a cluster. We will boot a Core cluster and pick up key architectural foundations as the cluster forms and transacts. We will then add in Read Replicas and show how they bootstrap join the cluster and then catchup and remain caught up with the Core Servers. We will then see how backup is used in live cluster environments before shutting down Read Replicas and Core Servers.

### Discovery protocol

The discovery protocol is the first step in forming a Causal Cluster. It takes in some information about existing Core cluster servers, and uses this to initiate a network join protocol.

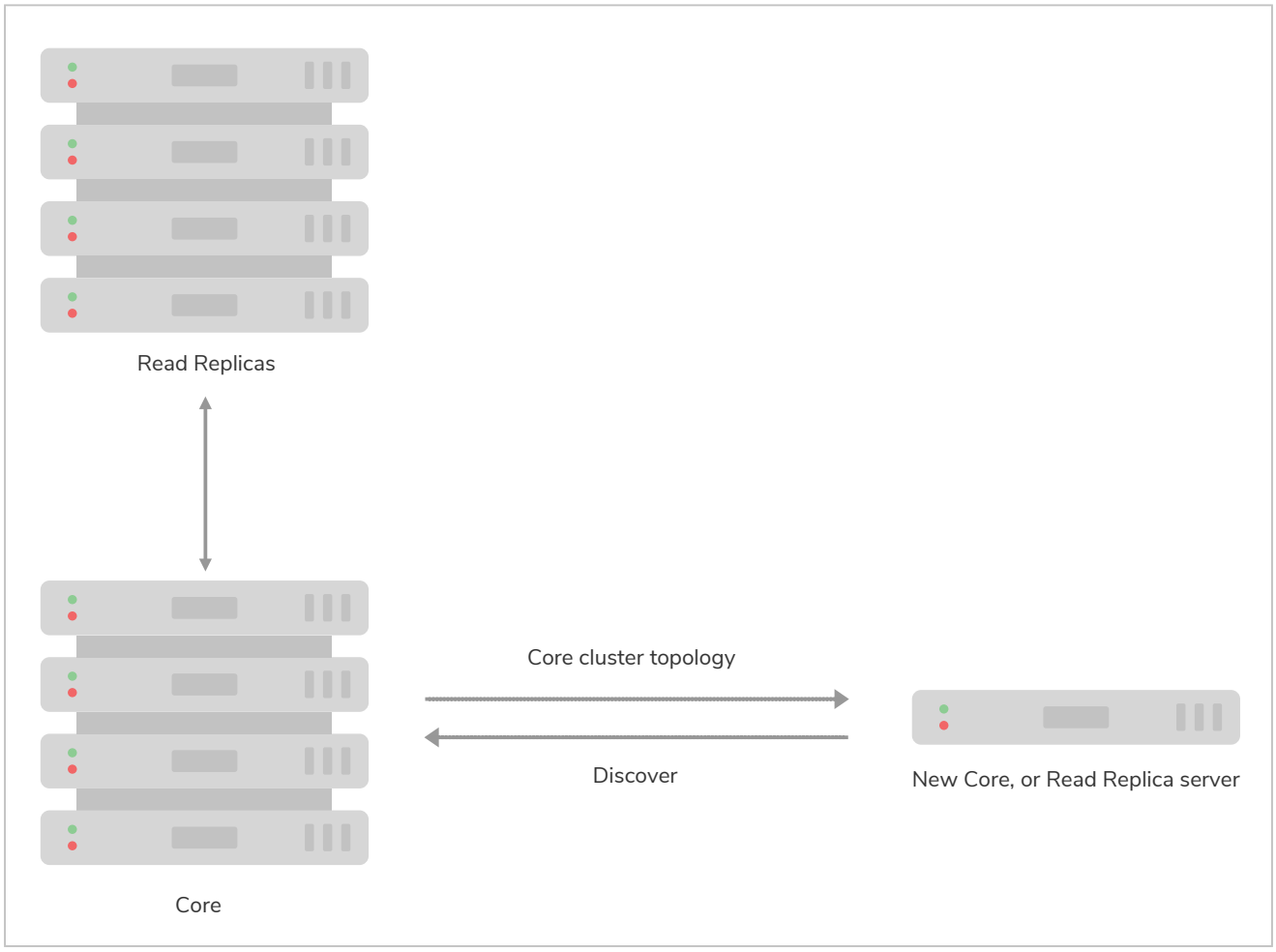



*Figure 38. Causal Cluster discovery protocol: Core-to-Core or Read Replica-to-Core only.*

Using this information, the server will either join an existing cluster or form one of its own.

> **Note:**
>
> The discovery protocol targets Core Servers only regardless of whether it is a Core Server or Read Replica performing discovery. It is because we expect Read Replicas to be both numerous and, relatively speaking, transient whereas Core Servers will likely be fewer in number and relatively stable over time.

The discovery protocol takes information from `causal_clustering.initial_discovery_members` in *neo4j.conf*, which lists which IP addresses and ports that form the cluster on startup. Detailed information about discovery and discovery configuration options is given in the Initial discovery of cluster members section. When consuming this information, the server will try to handshake with the other listed servers. On successful handshake with another server (or servers), the current server will discover the whole current topology.

The discovery protocol continues to run throughout the lifetime of the Causal Cluster and is used to maintain the current state of available servers and to help clients route queries to an appropriate server via the client-side drivers.

## Core membership

If it is a Core Server that is performing discovery, once it has made a connection to the one of the existing Core Servers, it then joins the Raft protocol. Each database is replicated by a logically separate Raft group, so the process below is repeated for every one.

> **Note:**
>
> Raft is a distributed algorithm for maintaining a consistent log across multiple shared-nothing servers designed by Diego Ongaro for his 2014 Ph.D. thesis. See the Raft thesis for details.

Raft handles cluster membership by making it a normal part of keeping a distributed log in sync. Joining a cluster involves the insertion of a cluster membership entry into the Raft log which is then reliably replicated around the existing cluster. Once that entry is applied to enough members of the Raft consensus group (those machines running the specific instance of the algorithm), they update their view of the cluster to include the new server. Thus membership changes benefit from the same safety properties as other data transacted via Raft (see Transacting via the Raft protocol for more information).

The new Core Server must also catch up its own Raft logs with respect to the other Core Servers as it initializes its internal Raft instance. This is the normal case when a cluster is first booted and has performed few operations. There will be a delay before the new Core Server becomes available if it also needs to catch up (as per Catchup protocol) graph data from other servers. This is the normal case for a long lived cluster where the servers holds a great deal of graph data.

Where a joining Neo4j instance has databases whose names match databases which already exist in the cluster, the database stores on the joining instance must be the same as their counterparts on cluster members (although they are allowed to be in previous states). For example, if a cluster contains a database named *products*, a new instance may join with a backup of *products*, but not a database named *products* with different contents. A new instance may also join a cluster if it does not contain any matching databases.

The described catchup process is repeated for each database which exists in the cluster.

## Read Replica membership

When a Read Replica performs discovery, once it has made a connection to any of the available Core clusters it proceeds to add itself into a shared whiteboard.

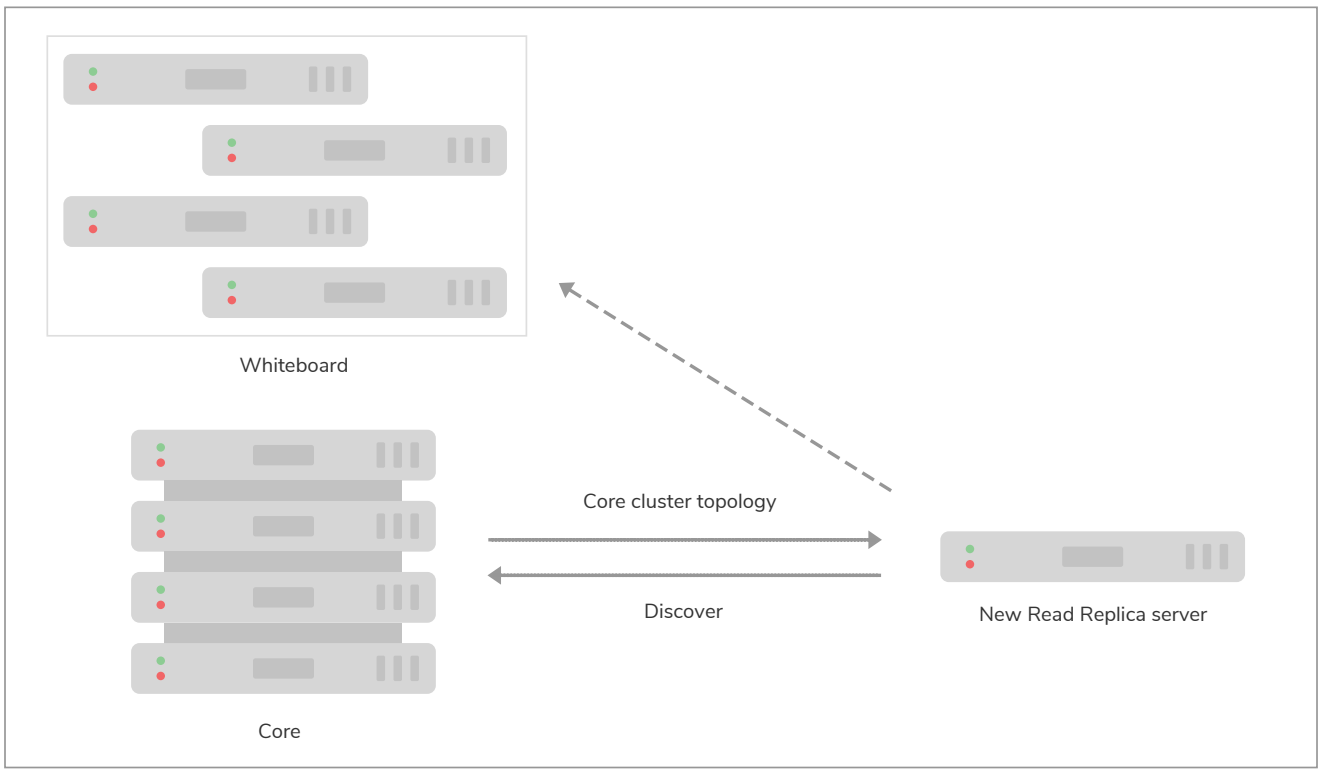



*Figure 39. All Read Replicas registered with shared whiteboard.*

This whiteboard provides a view of all live Read Replicas and is used both for routing requests from database drivers that support end-user applications and for monitoring the state of the cluster.

The Read Replicas are not involved in the Raft protocol, nor are they able to influence cluster topology. Hence a shared whiteboard outside of Raft comfortably scales to very large numbers of Read Replicas.

The whiteboard is kept up to date as Read Replicas join and leave the cluster, even if they fail abruptly rather than leaving gracefully.

## Transacting via the Raft protocol

Once bootstrapped, each Core Server spends its time processing database transactions. Updates are reliably replicated around Core Servers via the Raft protocol. Updates appear in the form of a (committed) Raft log entry containing transaction commands which is subsequently applied to update the database.

> **Note:**
>
> One of Raft's primary design goals is to be easily understandable so that there are fewer places for tricky bugs to hide in implementations. As a side-effect, it is also possible for database operators to reason about their Core Servers in their Causal Clusters.

The Raft Leader for the current term (a logical clock) appends the transaction (an 'entry' in Raft terminology) to the head of its local log and asks the other instances to do the same. When the Leader can see that a majority instances have appended the entry, it can be considered committed into the Raft log. The client application can now be informed that the transaction has safely committed since there is sufficient redundancy in the system to tolerate any (non-pathological) faults.



Note:

The Raft protocol describes three roles that an instance can be playing: *Leader*, *Follower*, and *Candidate*. These are transient roles and any Core Server can expect to play them throughout the lifetime of a cluster. While it is interesting from a computing science point of view to understand those states, operators should not be overly concerned: they are an implementation detail.



Note:

As each database operates within a logically separate Raft group, a core server can have multiple roles: one for each database. For example, it may be the *Leader* for database `system` and at the same time be a *Follower* for database `neo4j`.

For safety, within any Raft protocol instance there is only one Leader able to make forward progress in any given term. The Leader bears the responsibility for imposing order on Raft log entries and driving the log forward with respect to the *Followers*.

Followers maintain their logs with respect to the current Leader's log. Should any participant in the cluster suspect that the Leader has failed (not receiving new entries or heartbeats), then they can instigate a leadership election by entering the *Candidate* state. In Neo4j Core Servers this failure detection window is set by default above 20s to enable more stable leaders.

Whichever instance is in the best state (including the existing Leader, if it remains available) can emerge from the election as Leader. The "best state" for a Leader is decided by highest term, then by longest log, then by highest committed entry.

The ability to fail over roles without losing data allows forward progress even in the event of faults. Even where Raft instances fail, the protocol can rapidly piece together which of the remaining instances is best placed to take over from the failed instance (or instances) **without data loss**. This is the essence of a *non-blocking* consensus protocol which allows Neo4j Causal Clustering to provide continuous availability to applications.

## Catchup protocol

Read Replicas spend their time concurrently processing graph queries and applying a stream of transactions from the Core Servers to update their local graph store.

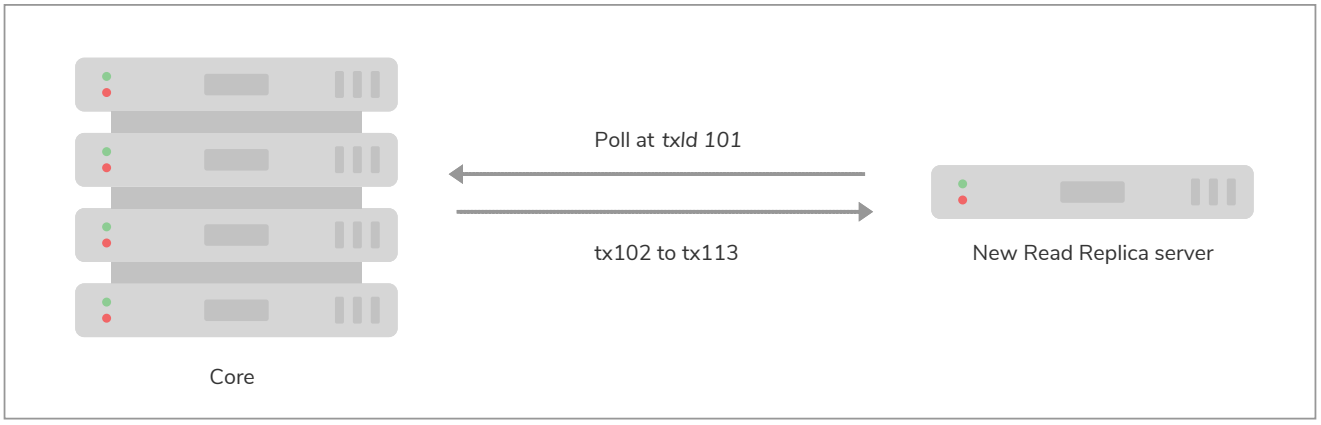



*Figure 40. Transactions shipped from Core to Read Replica.*

Updates from Core Servers to Read Replicas are propagated by transaction shipping. Transaction shipping is instigated by Read Replicas frequently *polling* any of the Core Servers specifying the ID of the last transaction they received and processed. The frequency of polling is an operational choice.

> **Note:**
>
> Neo4j transaction IDs are strictly monotonic integer values (they always increase). This makes it possible to determine whether or not a transaction has been applied to a Read Replica by comparing its last processed transaction ID with that of a Core Server.

If there is a large difference between an Read Replica's transaction history and that of a Core Server, polling may not result in any transactions being shipped. This is quite expected, for example when a new Read Replica is introduced to a long-running cluster or where a Read Replica has been down for some significant period of time. In such cases the catchup protocol will realize the gap between the Core Servers and Read Replica is too large to fill via transaction shipping and will fall back to copying the database store directly from Core Server to Read Replica. Since we are working with a live system, at the end of the database store copy the Core Server's database is likely to have changed. The Read Replica completes the catchup by asking for any transactions missed during the copy operation before becoming available.

> **Note:**
>
> A very slow database store copy could conceivably leave the Read Replica too far behind to catch up via transaction log shipping as the Core Server has substantially moved on. In such cases the Read Replica server repeats the catchup protocol. In pathological cases the operator can intervene to snapshot, restore, or file copy recent store files from a fast backup.

## Read Replica shutdown

On clean shutdown, a Read Replica will invoke the discovery protocol to remove itself from the shared whiteboard overview of the cluster. It will also ensure that the database is cleanly shutdown and consistent, immediately ready for future use.

On an unclean shutdown such as a power outage, the Core Servers maintaining the overview of the cluster will notice that the Read Replica's connection has been abruptly been cut. The discovery machinery

will initially hide the Read Replica's whiteboard entry, and if the Read Replica does not reappear quickly its modest memory use in the shared whiteboard will be reclaimed.

On unclean shutdown it is possible the Read Replica will not have entirely consistent store files or transaction logs. On subsequent reboot the Read Replica will rollback any partially applied transactions such that the database is in a consistent state.

### Core shutdown

A shutdown of a Core Server, like Core Server booting, is handled via the Raft protocol. When a member is shutdown, either cleanly or by force, it will eventually be voted out from the Raft group. All remaining instances accept that the cluster has grown smaller, and is therefore less fault tolerant. For any databases where the leaver was playing the *Leader* role, each of those leaderships will be transferred to other Core Servers. Once the new Leader is established, the Core cluster continues albeit with less redundancy.

If more members than the current fault tolerance leaves the cluster within a very short time period, the cluster cannot proceed and will lose quorum. However, if members are gradually lost, the cluster may have time to reduce the size of the cluster. A Core cluster of 5 members reduced to 3 can still continue operate normally with a fault tolerance reduced from 2 to 0. After the Raft protocol votes out the lost members which reduces the cluster size to 3, our fault tolerance has been increased from 0 to 1, and can lose yet another member and keep operating. This is because the Raft protocol has had time to vote out the lost members, and changed the cluster size of 5 (fault tolerance of 2) to 3 (fault tolerance of 1).



**Note:**

Raft may only reduce a cluster size to the configured causal_clustering.minimum_core_cluster_size_at_runtime. Once the cluster has reached this size, it will stop voting out members.

## .C.2. Multi-data center

Some use cases present high needs for availability, redundancy, locality of client applications, or simply scale. In these cases it is important that the cluster is aware of its physical topology so that it can optimize for workload. This makes configuring a single cluster to span multiple data centers a necessary proposition.

The following sections are dedicated to describing the different aspects of multi-data center operations of a Causal Cluster.

- Licensing for multi-data center operations
- Multi-data center design
  - Introduction
  - Core Server deployment scenarios
  - Allowing Read Replicas to catch up from other Read Replicas
- Multi-data center operations
  - Enable multi-data center operations

    - Server groups
    - Strategy plugins
- Multi-data center load balancing
    - Introduction
    - Prerequisite configuration
    - The load balancing framework
    - Load balancing examples
- Data center disaster recovery
    - Data center loss scenario
    - Procedure for recovering from data center loss


### Licensing for multi-data center operations

Multi-data center functionality is intended for very demanding users of Neo4j who typically operate under a commercial database license. As a result, multi-data center functionality is licensed separately from the single-data center Causal Clustering features.

In order to confirm that you are operating under a suitable license, you must explicitly set the following in *neo4j.conf*:

```
causal_clustering.multi_dc_license=true
```

Without this configuration, all of the multi-data center features will remain disabled.

## .C.3. Multi-data center design

This section describes the following:


- Introduction
- Core Server deployment scenarios
- Allowing Read Replicas to catch up from other Read Replicas
    - Hierarchical Read Replica deployment
    - Catch up (mostly) from peer Read Replicas
    - Maintaining causal consistency in scale-out topologies


### Introduction

This section is based on a series of examples to illustrate the different considerations we should take into account when designing our Causal Cluster for a multi-data center environment. We'll come to understand the weaknesses and benefits of common multi-data center deployment scenarios. Each scenario is presented at a high architectural level for clarity. In subsequent sections we will go into more detail on how such deployments are configured.

## Core Server deployment scenarios

We will start with the conceptually simplest multi-data center scenario where we deploy the same number and kind of instances into each DC. This is a *homogeneous* deployment because each data center is identical to the other.

*Example 147. Homogeneous three data center deployment*

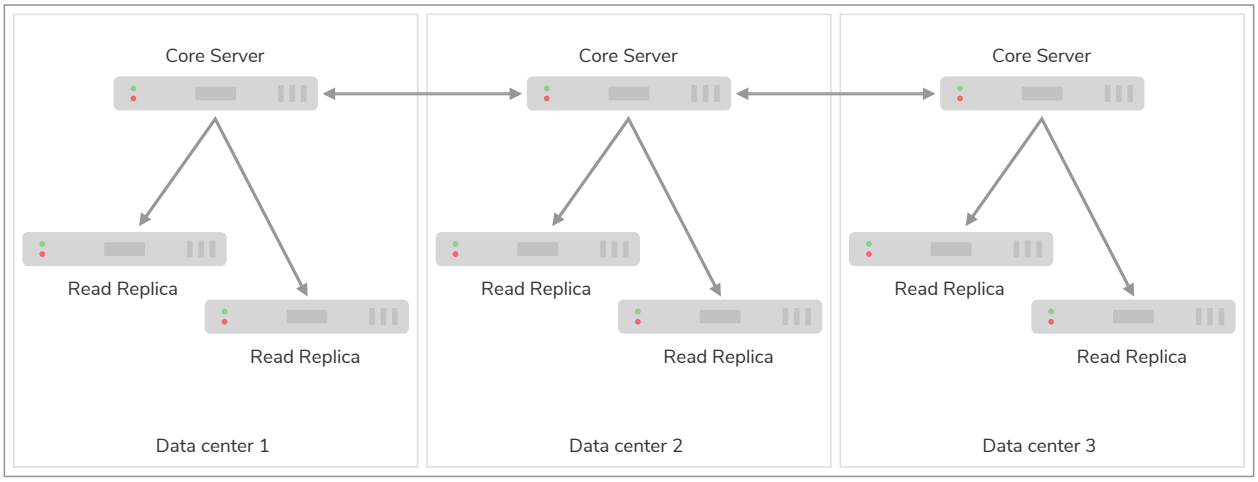



*Figure 41. Homogeneous deployment across three data centers with one Core instance in each*

In diagram above we have three data centers, each identically equipped with a single Core Server and a small number of Read Replicas.

Since Raft only requires a majority of the instances to acknowledge a write before it is safely committed, the latency of the commit path for this pattern involves only the two fastest data centers. As such the cost of committing to this setup is two WAN messages: one to send the transaction and one ACK message. In a non-failure case the other data center will not be far behind and will apply the transaction as well.

Within each of the data centers we can increase machine-level redundancy by adding more Core instances. For example we could add two more machines in each data center so that we can tolerate the spontaneous loss of up to four machines anywhere in the cluster or a single data center as a whole.

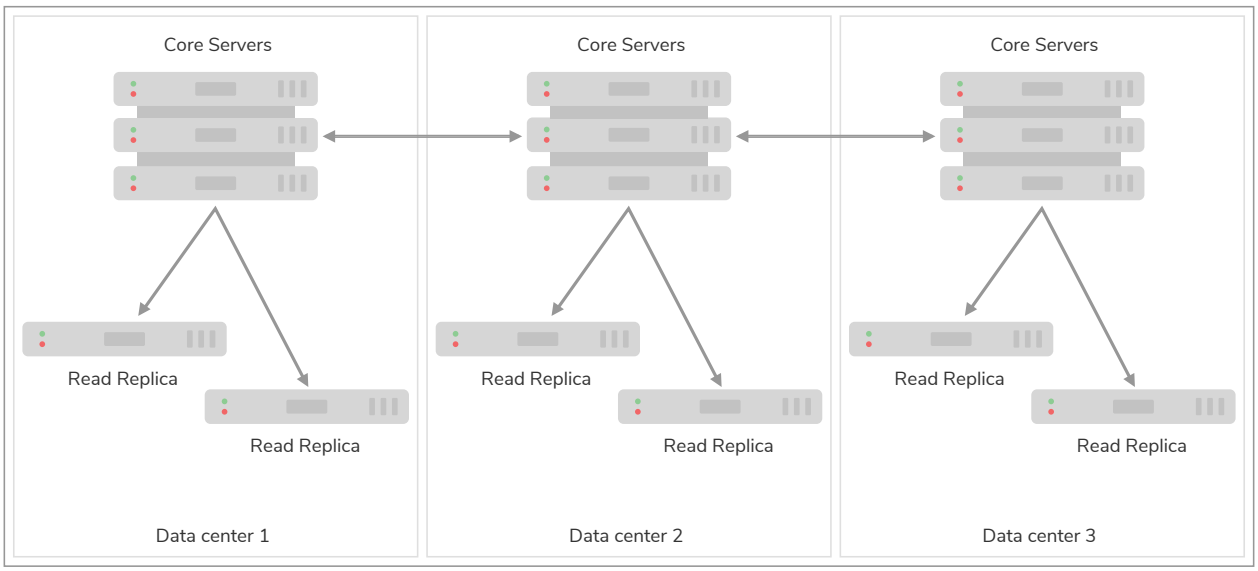



*Figure 42. Homogeneous deployment across three data centers with three Core instances in each*

To recap the strengths and weaknesses of this deployment pattern:

- We can lose an entire data center without losing availability and, depending on the number of machines in each data center, we may still be able to tolerate the loss of individual servers regardless of which data center they are in.
- The commit path for transactions is short, just two WAN messages exchanged.
- While the loss of majority data centers will need to be recovered, the operational procedure is identical irrespective of which of the data centers are lost.

As will be shown in the section on multi-data center configuration the Read Replicas can be biased to catchup from their data center-local Core Servers to minimize catchup latency. Data center-local client applications would also likely be routed to those same Read Replicas both for topological locality and scaling. More details are available in the section on multi-data center load balancing.

In the two data center case, our first instinct is to balance the available servers for operational consistency. An example of a homogeneous deployment across two data centers with two Core instances in each is illustrated in the diagram below:

*Example 148. Homogeneous two data center deployment*

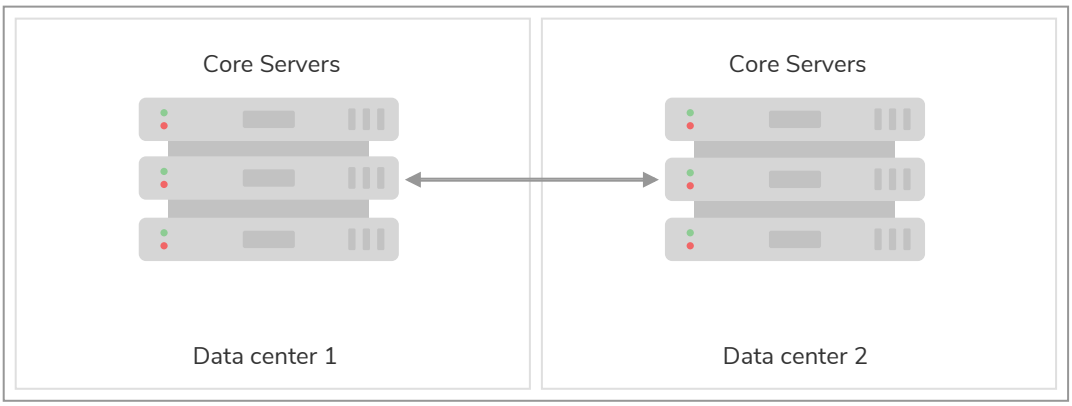



*Figure 43. Homogeneous deployment across two data centers*

The problem with this configuration is that while architecturally simple, it does not play to the strengths of the Raft protocol which is based on majority consensus. In the non-failure case, we incur two WAN messages to commit any transaction because a majority commit implies at least one response from the non-local data center instances. Worse, if we lose either data center the cluster will become read-only because it is impossible to achieve a majority.

As seen in the example above, the homogeneous deployment over two data centers does not take full advantage of the strengths of Causal Clustering. However it guarantees that the full Raft log will be present in either data center in the case of total data center loss.

The opposite of spreading Core Servers around our data centers, is to have them all hosted in a single one. This may be for technical or governance reasons, but either way has the advantage of LAN commit latencies for writes.

While our Core Servers are colocated, we spread out our Read Replicas close to the client applications to enable fan-out scaling.

*Example 149. Core Servers and Read Replicas segregated by data center*

The diagram below shows an example of a heterogeneous deployment directing writes to one data center, and reads to all. This pattern provides high survivability for data because of geo-replication. It also provides locality for client applications. However, if the Core Server data center is lost, we must immediately instigate recovery and turn one of the remaining Read Replica data centers into a new Core cluster.

It is possible that none of the Read Replicas have received all of the confirmed transactions prior to losing Data Center 1. While this is a convenient pattern for geo-replication, its semantics are best-effort. Cluster designers must take this aspect under consideration when deciding on recovery strategy.

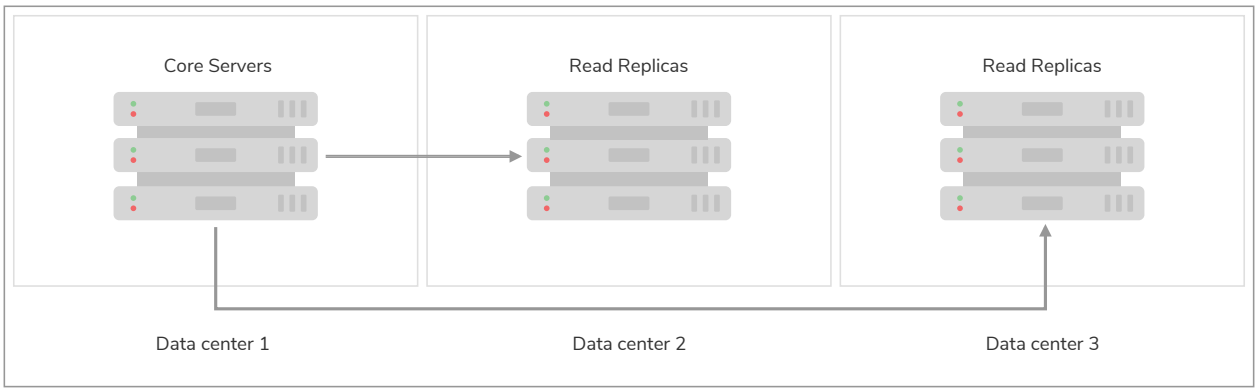



*Figure 44. Heterogeneous deployment separating Read Replicas from the Core cluster*

An operational tweak to this approach would be to host a Core Server in Data Center 2 and 3 as the starting point for recovery. During normal operations, these extra Core Servers should be configured with `causal_clustering.refuse_to_be_leader=true`. Should we lose Data Center 1, then we can use one of these Core Servers to quickly bootstrap a new Core cluster and return to full service rapidly.

To recap the strengths of this deployment pattern:

- Core Servers commit at LAN latencies if using the setup with Core Servers exclusively in one data center.
- Read Replicas provide scale and locality for client applications.
- Geo-replication provides high survivability for data.

## Allowing Read Replicas to catch up from other Read Replicas

With an understanding of the basic multi-data center patterns at our disposal, we can refine our deployment models to embrace local catchup within data centers. This means that any server, including Read Replicas, can act as a source of transactions for Read Replica server. When catching up from data center-local instances we aim to amortize the cost of WAN traffic catchup across many local replications.

Allowing Read Replicas to choose a data center-local Core Server or even another Read Replica gives us a great deal of design freedom, and importantly allows us to scale to truly huge numbers of Read Replicas. Using this feature we might choose to fan-out Read Replicas so that the catchup load on the Core Servers grows (approximately) logarithmically rather than linearly.

### Hierarchical Read Replica deployment

The primary motivation for Read Replicas catching up from other Read Replicas is to allow for fan-out scale. To achieve a fan-out we arrange the Read Replicas in a hierarchy, with each layer of the hierarchy being broader than the one above.

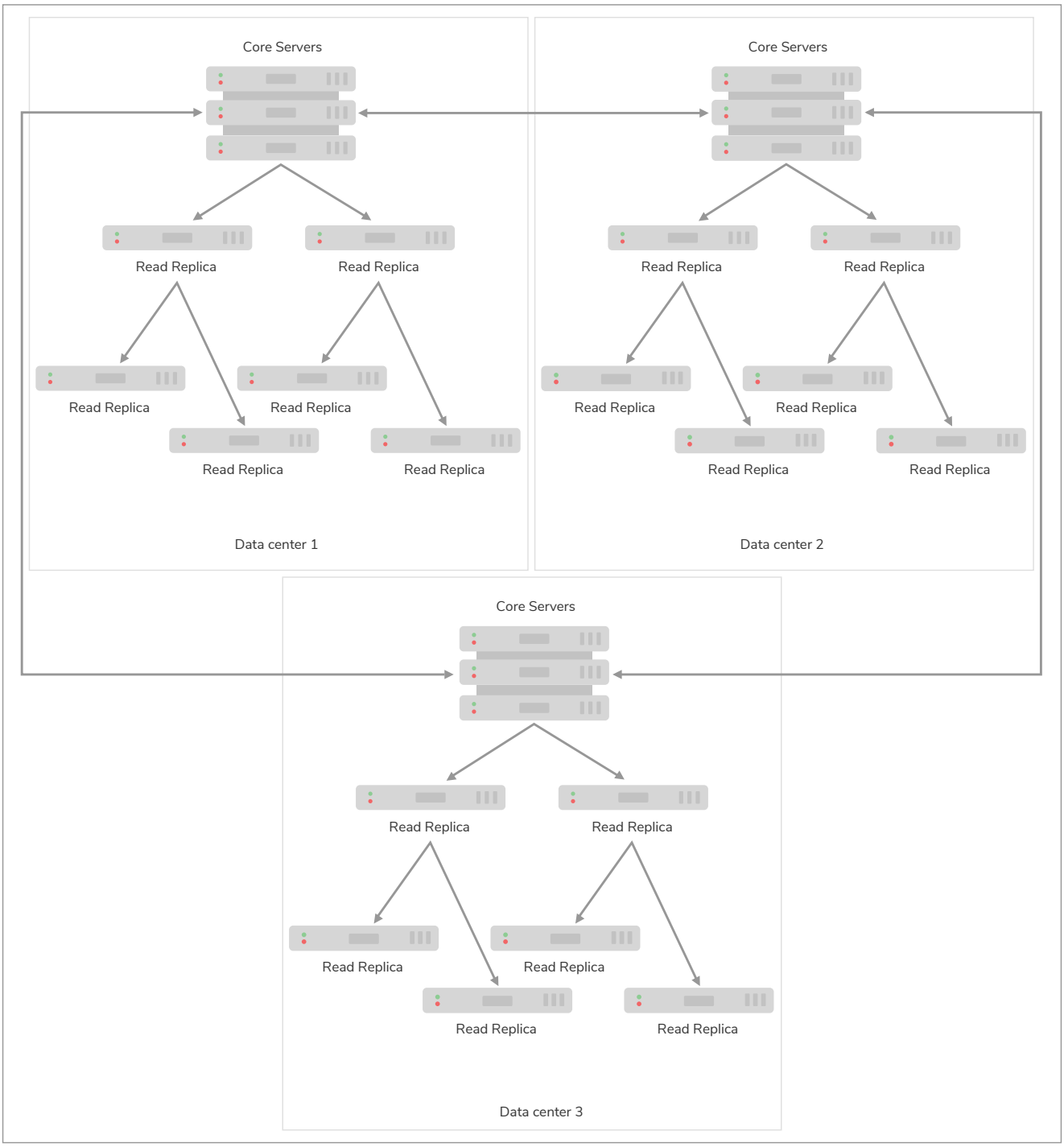



*Figure 45. Fan out from Core Servers for scale at log cost*

An illustrative hierarchy is presented in the diagram above. The Core Servers supply transactions to a relatively small number of Read Replicas at the first tier. This results in a relatively modest load on the Core Servers, freeing up resources to focus on the commit path. Those Read Replicas in the first tier in turn feed a larger number of Read Replicas in the second tier. This pattern can be reasonably extended to several tiers to provide enormous fan-out.

At each tier we expand the scalability of the Read Replicas, but we add another level of catchup latency. By careful measurement we can ascertain the appropriate depth and breadth of the hierarchy to match the application requirements.

We should also take care that each tier in the hierarchy has sufficient redundancy so that failures do not compromise transmission of data from the Core Servers. A strategy for keeping Read Replicas current in

the presence of failures is to occasionally have them subvert the hierarchy. That is, if a given Read Replica occasionally goes to its grandparents or even directly to the Core Servers then we can avoid pathologically high replication latencies under fault conditions.

### Catch up (mostly) from peer Read Replicas

Another strategy for Read Replica catchup is to treat them all as peers and have peer-to-peer catchup. This avoids the need to manage tiers of replicas to maintain availability since the Read Replicas catch up from one another in a mesh.

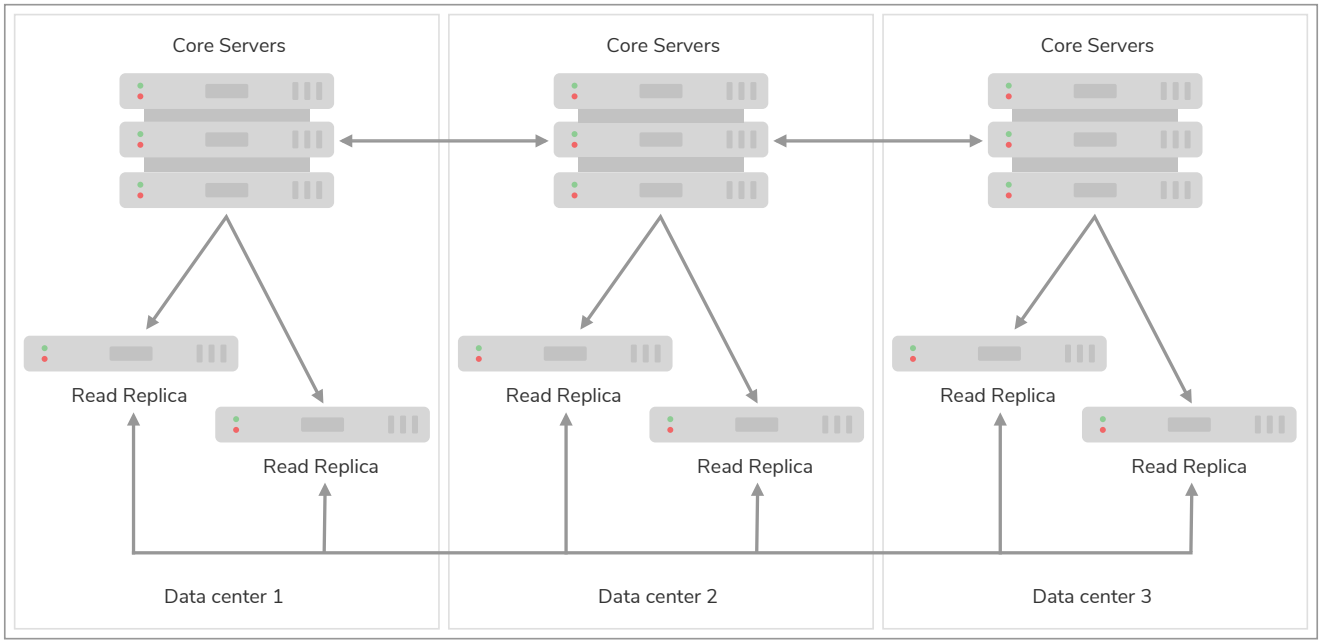



*Figure 46. Peer-to-peer Read Replica catchup*

Having a reduced load on the Core Servers allows us to scale out. For example if only one in ten catchup requests goes to the Core Servers, we could expand the number of Read Replicas by approximately a factor of 10.

To avoid groups of orphans in the mesh, Read Replicas will occasionally catch up directly from Core Servers. Having Read Replicas catch up with Core Servers ensures that no Read Replica is left behind indefinitely, placing an upper bound on replication latency. While this places some load on the Core Servers, it is far less than if all catch up attempts from Read Replicas were directed to a Core Server.

The upper bound on replication latency for this mode of operation is the number of catchup attempts served by Read Replicas before trying core. The average replication latency will be half the number of attempts to replicate. This is because on average half the Read Replicas will be ahead and half behind any given Read Replica.



**Note:**

Connecting to a random Core Server on failure to retrieve updates from other sources is the default behavior of Read Replicas.

### Maintaining causal consistency in scale-out topologies

Causal consistency is always maintained, even in extreme situations with chains of Read Replicas catching up from other upstream Read Replicas. The key trade-off to understand, as so often in distributed systems, is that of latency for scale.

In Fan out from Core Servers for scale at log cost", role="middle we see that number of hops required for a transaction to propagate to the lowest tier is 2: the highest latency in this topology. Equally we see how the bottommost tier has far more members than any other tier giving it scale advantages.

Correspondingly, in the middle tier we have better latency (one hop) but less scale. At the top most tier (Core Servers) we have very little latency (just the Raft commit path) but the fewest available servers. This means we should target queries at the most appropriate tier based on latency, scale, and locality.

**Summary on latency versus scalability:**

- Issuing read queries to a Core Server generally has the lowest latency in principle but may have the highest contention.
- Issuing read queries to a Read Replica topologically closest to Core Servers typically has higher latency but also higher scalability.
- Issuing read queries to a Read Replica topologically further from Core Servers typically has the highest latency but also the highest scalability.

In large systems like the scale-out hierarchy above, we are conventionally used to having relaxed or *eventual* consistency semantics. With Neo4j multi-data center setups, that is also possible. Where we don't care about causality we can read from any Read Replica and accept that we might see older values. However the causal consistency semantics are maintained.

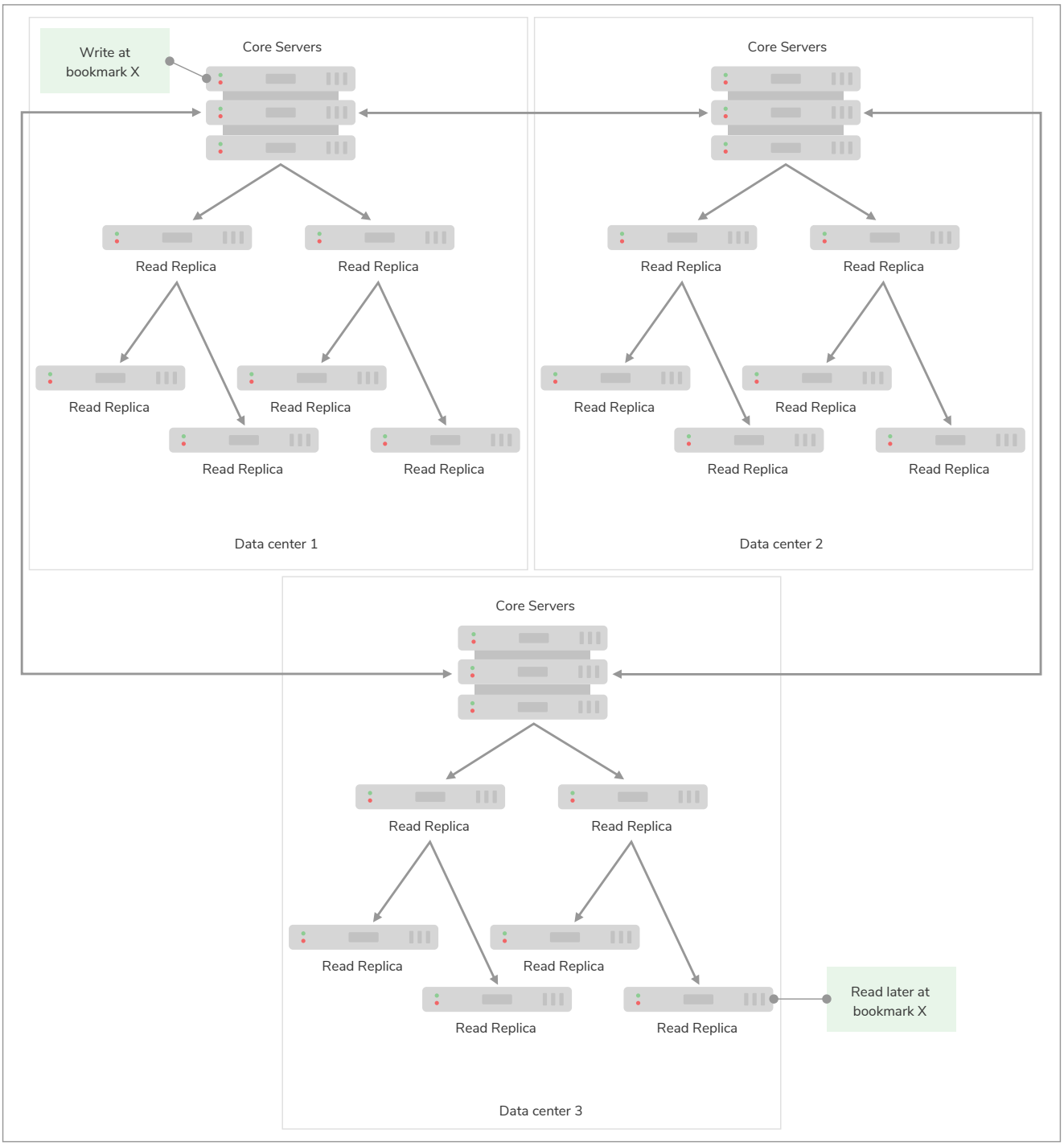



*Figure 47. Each tier in the Read Replicas is further behind the source of truth, but offers greater scale-out*

As we can see in diagram above, even if the client binds to a Read Replica that is multiple hops/data centers away from the source of truth, causal consistency is maintained. While the query may be suspended while the necessary transaction propagates to the Read Replica, the benefit is that there will be more Read Replicas available and so overall client throughput is higher than with a single-tier configuration.

## .C.4. Multi-data center operations

This section describes the following:

- Enable multi-data center operations

- Server groups
- Strategy plugins
  - Configuring upstream selection strategy using pre-defined strategies
  - Configuring user-defined strategies
  - Building upstream strategy plugins using Java
  - Favoring data centers


## Enable multi-data center operations

Before doing anything else, we must enable the multi-data center functionality. This is described in Licensing for multi-data center operations.

> **Note:**
>
> **Licensing for multi-data center**
>
> The multi-data center functionality is separately licensed and must be specifically enabled.

## Server groups

In order to optimize the use of our Causal Cluster servers according to our specific requirements, we sort them into *Server Groups*. Server Group membership can map to data centers, availability zones, or any other significant topological elements from the operator's domain. Server Groups can also overlap.

Server Groups are defined as a key that maps onto a set of servers in a Causal Cluster. Server Group membership is defined on each server using the `causal_clustering.server_groups` parameter in *neo4j.conf*. Each server in a Causal Cluster can belong to zero or more server groups.

*Example 150. Definition of Server Group membership*

The membership of a server group or groups can be set in *neo4j.conf* as in the following examples:

```
# Add the current instance to the groups `us` and `us-east`
causal_clustering.server_groups=us,us-east
```

```
# Add the current instance into the group `london`
causal_clustering.server_groups=london
```

```
# Add the current instance into the group `eu`
causal_clustering.server_groups=eu
```

We must be aware that membership of each server group is explicit. For example, a server in the `gb-london` group is not automatically part of some `gb` or `eu` group unless that server is explicitly added to those groups. That is, any (implied) relationship between groups is reified only when those groups are used as the basis for requesting data from upstream systems.

Server Groups are not mandatory, but unless they are present, we cannot set up specific upstream transaction dependencies for servers. In the absence of any specified server groups, the cluster defaults to its most pessimistic fall-back behavior: each Read Replica will catch up from a random Core Server.

## Strategy plugins

*Strategy plugins* are sets of rules that define how Read Replicas contact servers in the cluster in order to synchronize transaction logs. Neo4j comes with a set of pre-defined strategies, and also provides a Domain Specific Language, *DSL*, to flexibly create user-defined strategies. Finally, Neo4j supports an API which advanced users may use to enhance upstream recommendations.

Once a strategy plugin resolves a satisfactory upstream server, it is used for pulling transactions to update the local Read Replica 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 strategies

Neo4j ships with the following pre-defined strategy plugins. These provide coarse-grained algorithms for choosing an upstream instance:

| Plugin name | Resulting behavior |
| --- | --- |
| `connect-to-random-core-server` | Connect to any **Core Server** selecting at random from those currently available. |
| `typically-connect-to-random-read-replica` | Connect to any available **Read Replica**, but around 10% of the time connect to any random Core Server. |
| `connect-randomly-to-server-group` | Connect at random to any available **Read Replica** in any of the server groups specified in the comma-separated list `causal_clustering.connect-randomly-to-server-group`. |
| `leader-only` | Connect only to the current Raft leader of the Core Servers. |
| `connect-randomly-within-server-group` | Connect at random to any available **Read Replica** in any of the server groups to which this server belongs. Deprecated, please use `connect-randomly-to-server-group`. |

Pre-defined strategies are used by configuring the `causal_clustering.upstream_selection_strategy` option. Doing so allows us to specify an ordered preference of strategies to resolve an upstream provider of transaction data. We provide a comma-separated list of strategy plugin names with preferred strategies earlier in that list. The upstream strategy is chosen by asking each of the strategies in list-order whether they can provide an upstream server from which transactions can be pulled.

*Example 151. Define an upstream selection strategy*

Consider the following configuration example:

```
causal_clustering.upstream_selection_strategy=connect-randomly-to-server-group,typically-connect-to-
```

```
random-read-replica
```

With this configuration the instance will first try to connect to any other instance in the group(s) specified in `causal_clustering.connect-randomly-to-server-group`. Should we fail to find any live instances in those groups, then we will connect to a random Read Replica.

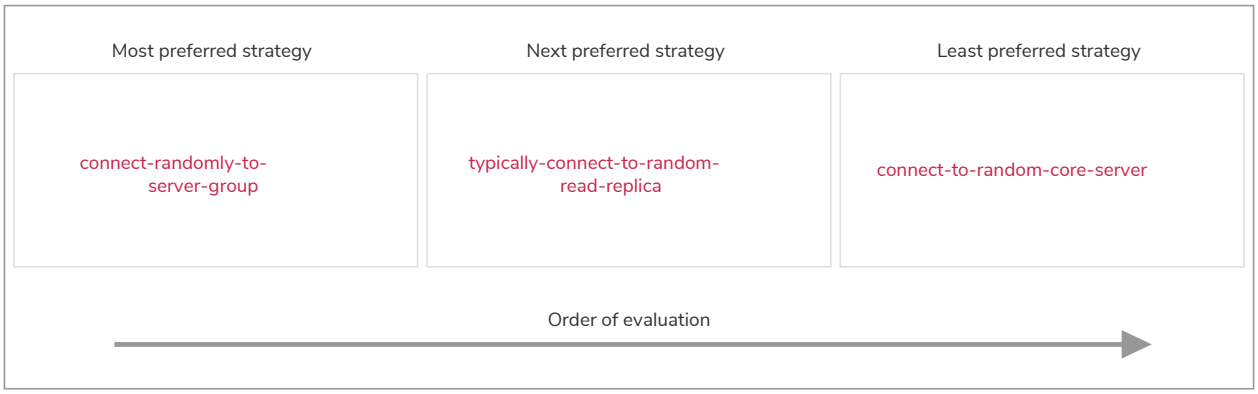



*Figure 48. The first satisfactory response from a strategy will be used.*

To ensure that downstream servers can still access live data in the event of upstream failures, the last resort of any instance is always to contact a random Core Server. This is equivalent to ending the `causal_clustering.upstream_selection_strategy` configuration with `connect-to-random-core-server`.

### Configuring user-defined strategies

Neo4j Causal Clusters support a small DSL for the configuration of client-cluster load balancing. This is described in detail in Policy definitions and Filters. The same DSL is used to describe preferences for how an instance binds to another instance to request transaction updates.

The DSL is made available by selecting the `user-defined` strategy as follows:

```
causal_clustering.upstream_selection_strategy=user-defined
```

Once the user-defined strategy has been specified, we can add configuration to the `causal_clustering.user_defined_upstream_strategy` setting based on the server groups that have been set for the cluster.

We will describe this functionality with two examples:

*Example 152. Defining a user-defined strategy*

For illustrative purposes we propose four regions: `north`, `south`, `east`, and `west` and within each region we have a number of data centers such as `north1` or `west2`. We configure our server groups so that each data center maps to its own server group. Additionally we will assume that each data center fails independently from the others and that a region can act as a supergroup of its constituent data centers. So an instance in the `north` region might have configuration like `causal_clustering.server_groups=north2,north` which puts it in two groups that match to our physical topology as shown in the diagram below.




*Figure 49. Mapping regions and data centers onto server groups*

Once we have our server groups, our next task is to define some upstream selection rules based on them. For our design purposes, let's say that any instance in one of the `north` region data centers prefers to catchup within the data center if it can, but will resort to any northern instance otherwise. To configure that behavior we add:

```
causal_clustering.user_defined_upstream_strategy=groups(north2); groups(north); halt()
```

The configuration is in precedence order from left to right. The `groups()` operator yields a server group from which to catch up. In this case only if there are no servers in the `north2` server group will we proceed to the `groups(north)` rule which yields any server in the `north` server group. Finally, if we cannot resolve any servers in any of the previous groups, then we will stop the rule chain via `halt()`.

Note that the use of `halt()` will end the rule chain explicitly. If we don't use `halt()` 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. But we can be more sophisticated if we so choose. For example we can place conditions on the server groups from which we catch up.

*Example 153. User-defined strategy with conditions*

In this example we wish to roughly qualify cluster health before choosing from where to catch up. For this we use the `min()` filter as follows:

```
causal_clustering.user_defined_upstream_strategy=groups(north2)->min(3), groups(north)->min(3);
all();
```

`groups(north2)->min(3)` states that we want to catch up from the `north2` server group if it has three available machines, which we here take as an indicator of good health. If `north2` can't meet that

requirement (is not healthy enough) then we try to catch up from any server across the `north` region provided there are at least three of them available as per `groups(north)->min(3)`. Finally, if we cannot catch up from a sufficiently healthy `north` region, then we'll (explicitly) fall back to the whole cluster with `all()`.

The `min()` filter is a simple but reasonable indicator of server group health.

### Building upstream strategy plugins using Java

Neo4j supports an API which advanced users may use to enhance upstream recommendations in arbitrary ways: load, subnet, machine size, or anything else accessible from the JVM. In such cases we are invited to build our own implementations of `org.neo4j.causalclustering.readreplica.UpstreamDatabaseSelectionStrategy` to suit our own needs, and register them with the strategy selection pipeline just like the pre-packaged plugins.

We have to override the `org.neo4j.causalclustering.readreplica.UpstreamDatabaseSelectionStrategy#upstreamDatabase()` method in our code. Overriding that class gives us access to the following items:

| Resource | Description |
| --- | --- |
| `org.neo4j.causalclustering.discovery.TopologyService` | This is a directory service which provides access to the addresses of all servers and server groups in the cluster. |
| `org.neo4j.kernel.configuration.Config` | This provides the configuration from *neo4j.conf* for the local instance. Configuration for our own plugin can reside here. |
| `org.neo4j.causalclustering.identity.MemberId` | This provides the unique cluster `MemberId` of the current instance. |

Once our code is written and tested, we have to prepare it for deployment. `UpstreamDatabaseSelectionStrategy` plugins are loaded via the Java Service Loader. This means when we package our code into a jar file, we'll have to create a file *META-INF.services/org.neo4j.causalclustering.readreplica.UpstreamDatabaseSelectionStrategy* in which we write the fully qualified class name(s) of the plugins, e.g. `org.example.myplugins.PreferServersWithHighIOPS`.

To deploy this jar into the Neo4j server we copy it into the *plugins* directory and restart the instance.

### Favoring data centers

In a multi-DC scenario, while it remains a rare occurrence, it is possible to bias where writes for the specified database should be directed. We can apply `causal_clustering.leadership_priority_group` to specify a group of servers which should have priority when selecting the leader for a given database. The priority group can be set on one or multiple databases and it means that the cluster will attempt to keep the leadership for the configured database on an instance tagged with the configured server group.

A database for which `leadership_priority_group` has been configured will be excluded from the automatic balancing of leaderships across a cluster. It is therefore recommended to not use this configuration unless it is necessary.

# .C.5. Multi-data center load balancing

This section describes the following:


- Introduction
- Prerequisite configuration
  - Enable multi-data center operations
  - Server groups
  - Cores for reading
- The load balancing framework
  - Policy definitions
  - Policy names
  - Filters
- Load balancing examples




**Note:**

**Enabling load balancing**

The load balancing functionality is part of the separately licensed multi-data center package and must be specifically enabled. See Licensing for multi-data center operations for details.

## Introduction

When deploying a multi-data center cluster we often wish to take advantage of locality to reduce latency and improve performance. For example, we would like our graph-intensive workloads to be executed in the local data center at LAN latencies rather than in a faraway data center at WAN latencies. Neo4j's enhanced load balancing for multi-data center scenarios facilitates precisely this and can also be used to define fall-back behaviors. This means that failures can be planned for upfront and persistent overload conditions be avoided.

The load balancing system 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 like for example a new server having been added for increased capacity. There are also failure situations where the driver will ask again immediately, for example when it cannot use any of its allocated servers.

This is mostly transparent from the perspective of a client. On the server side we configure the load balancing behaviors and expose them under a named *load balancing policy* which the driver can bind to. All server-side configuration is performed on the Core Servers.



**Note:**

**Use load balancing from Neo4j drivers**

This chapter describes how to configure a Causal Cluster to use custom load balancing policies Once enabled and configured, the custom load balancing feature is used by drivers to route traffic as intended. See the Neo4j Driver manuals for instructions on how to configure drivers to use custom load balancing.

## Prerequisite configuration

### Enable multi-data center operations

In order to configure a cluster for load balancing we must enable the multi-data center functionality. This is described in Licensing for multi-data center operations.

### Server groups

In common with server-to-server catchup, load balancing across multiple data centers is predicated on the *server group* concept. Servers can belong to one or more potentially overlapping server groups, and decisions about where to route requests from client to cluster member are parameterized based on that configuration. For details on server group configuration, refer to Server groups.

### Cores 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 Core Servers. The following configuration will allow the routing of read workload to Core Servers. Valid values are `true` and `false`.

```
causal_clustering.cluster_allow_reads_on_followers=true
```

## The load balancing framework

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:

```
causal_clustering.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.

It is of crucial importance to define the exact same policies on all core machines since this is to be regarded as cluster-wide configuration and failure to do so will lead to surprising behavior. Similarly, policies which are in active use should not be removed or renamed since it will break 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 which is not available, then it will not be able to use the cluster. A driver

which does not specify any name at all will get 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 will result in suboptimal routing choices or even prevent successful interactions with the cluster entirely.



Note:

The details of how to write a custom plugin is not documented here. Please get in contact with Neo4j Professional Services if you think that you need a custom plugin.

### 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 will be 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 compose 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 will be returned to the driver. However, if there are no servers left then the next rule will be considered. If no rule is able to produce a usable result then the driver will be signalled a failure.

### Policy names

The policies are configured under the namespace of the *server policies* plugin and named as desired. 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`.

```
causal_clustering.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 will be used by driver clients which 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.

- `groups(name1, name2, …)`
  - Only servers which are part of any of the specified groups will pass the filter.
  - The defined names must match those of the server groups.
- `min(count)`
  - Only the minimum amount of servers will be 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.
  - Will stop the processing of any more rules.

The groups filter is essentially an OR-filter, e.g. `groups(A,B)` which will pass any server in either A, B or both (the union of the server groups). An AND-filter can also be created by chaining two filters as in `groups(A) -> groups(B)`, which will only pass servers in both groups (the intersect of the server groups).

## Load balancing examples

In our discussion on multi-data center clusters we introduced a four region, multi-data center setup. We used the cardinal compass points for regions and numbered data centers within those regions. We'll use the same hypothetical setup here too.




*Figure 50. Mapping regions and data centers onto server groups*

We configure the behavior of the load balancer in the property

`causal_clustering.load_balancing.config.server_policies.<policy-name>`. The rules we specify will allow us to fine tune how the cluster routes requests under load.

In the examples we will 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 would be to insist on a particular data center to the exclusion of all others:

*Example 154. Specific data center only*

```
causal_clustering.load_balancing.config.server_policies.north1_only=\
groups(north1)->min(2); halt();
```

In this case we're stating that we are only interested in sending queries to servers in the `north1` server group, which maps onto a specific physical data center, provided there are two of them available. If we cannot provide at least two servers in `north1` then we should `halt()`, i.e. not try any other data center.

While the previous example demonstrates the basic form of our load balancing rules, we can be a little more expansive:

*Example 155. Specific data center preferably*

```
causal_clustering.load_balancing.config.server_policies.north1=\
groups(north1)->min(2);
```

In this case if at least two servers are available in the `north1` data center then we will load balance across them. Otherwise we will use any server in the whole cluster, falling back to the implicit, final `all()` rule.

The previous example considered only a single data center before resorting to the whole cluster. If we have a hierarchy or region concept exposed through our server groups we can make the fall back more graceful:

*Example 156. Gracefully falling back to neighbors*

```
causal_clustering.load_balancing.config.server_policies.north_app1=\
groups(north1,north2)->min(2);\
groups(north);\
all();
```

In this case we're saying that the cluster should load balance across the `north1` and `north2` data centers provided there are at least two machines available across them. Failing that, we'll resort to any instance in the `north` region, and if the whole of the north is offline we'll resort to any instances in the cluster.

# .C.6. Data center disaster recovery

This section describes the following:

- Data center loss scenario
- Procedure for recovering from data center loss

## Data center loss scenario

This section describes how to recover a multi-data center deployment which owing to external circumstances has reduced the cluster below half of its members. It is most easily typified by a 2x2 deployment with 2 data centers each containing two instances. This deployment topology can either arise because of other data center failures, or be a deliberate choice to ensure the geographic survival of data for catastrophe planning. However, by distributing an instance over three data centers instead, you could avoid having the cluster lose quorum through a single data center failure. For example, in a *1x1x1* deployment.

Under normal operation this provides a stable majority quorum where the fastest three out of four machines will execute users' transactions, as we see highlighted in Two Data Center Deployment with Four Core Instances", role="middle.

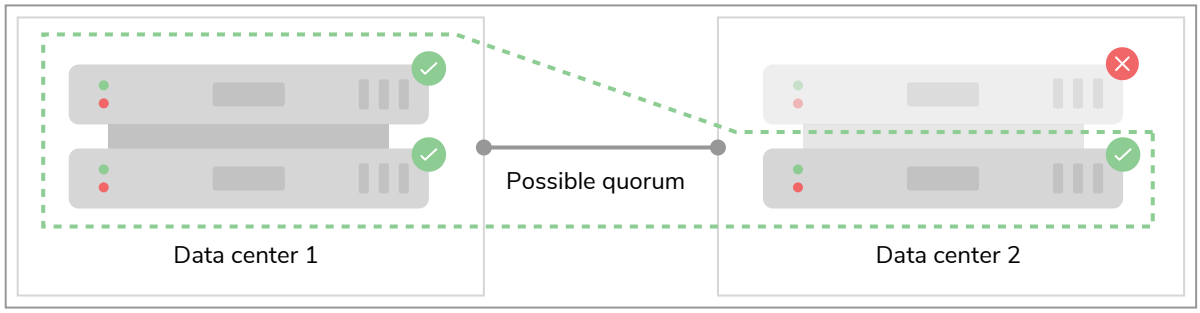



*Figure 51. Two Data Center Deployment with Four Core Instances*

However if an entire data center becomes offline because of some disaster, then a *majority quorum* cannot be formed in this case.

> **Note:**
>
> Neo4j Core clusters are based on the Raft consensus protocol for processing transactions. The Raft protocol requires a majority of cluster members to agree in order to ensure the safety of the cluster and data. As such, the loss of a majority quorum results in a read-only situation for the remaining cluster members.

When data center is lost abruptly in a disaster rather than having the instances cleanly shut down, the surviving members still believe that they are part of a larger cluster. This is different from even the case of rapid failures of individual instances in a live data center which can often be detected by the underlying cluster middleware, allowing the cluster to automatically reconfigure.

Conversely if we lose a data center, there is no opportunity for the cluster to automatically reconfigure. The loss appears instantaneous to other cluster members. However, because each remaining machine has only

a partial view of the state of the cluster (its own), it is not safe to allow any individual machine to make an arbitrary decision to reform the cluster.

In this case we are left with two surviving machines which cannot form a quorum and thus make progress.

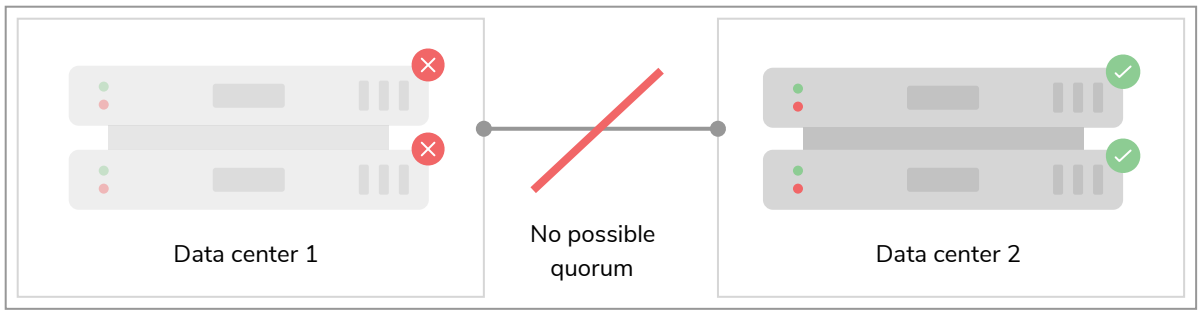



*Figure 52. Data Center Loss Requires Guided Recovery*

But, from a birds's eye view, it's clear we have surviving machines which are sufficient to allow a non-fault tolerant cluster to form under operator supervision.

> **Note:**
>
> Groups of individual cluster members (e.g. those in a single data center) may become isolated from the cluster during network partition for example. If they arbitrarily reformed a new, smaller cluster there is a risk of *split-brain*. That is from the clients' point of view there may be two or more smaller clusters that are available for reads and writes depending on the nature of the partition. Such situations lead to divergence that is tricky and laborious to reconcile and so best avoided.

To be safe, an operator or other out-of-band agent (e.g. scripts triggered by well-understood, trustworthy alerts) that has a trusted view on the whole of the system estate must make that decision. In the surviving data center, the cluster can be rebooted into a smaller configuration whilst retaining all data committed to that point. While end users may experience unavailability during the switch over, no committed data will be lost.

## Procedure for recovering from data center loss

The following procedure for performing recovery of a data center should not be done lightly. It assumes that we are completely confident that a disaster has occurred and our previously data center-spanning cluster has been reduced to a read-only cluster in a single data center, where there is no possible way to repair a connection to the lost instances. Further it assumes that the remaining cluster members are fit to provide a seed from which a new cluster can be created from a data quality point of view.

Having acknowledged the above, the procedure for returning the cluster to full availability following catastrophic loss of all but one data centers can be done using one of the following options, depending on your infrastructure.

Please note that the main difference between the options is that Option 2 will allow read-availability during recovery.

*Option 1.*

If you are unable to add instances to the current data-center, and can only use the current read-only cluster, the following steps are recommended:

1. Verify that a catastrophe has occurred, and that access to the surviving members of the cluster in the surviving data center is possible. Then for each instance:
    a. Stop the instance with `bin/neo4j stop` or shut down the service.
    b. Change the configuration in *neo4j.conf* such that the `causal_clustering.initial_discovery_members` property contains the DNS names or IP addresses of the other surviving instances.
    c. Optional: you may need to update `causal_clustering.minimum_core_cluster_size_at_formation`, depending on the current size of the cluster (in the current example, two cores).
    d. Unbind the instance using `neo4j-admin unbind`.
    e. Start the instance with `bin/neo4j start` or start the `neo4j` service.

*Option 2.*

If it is possible to create a new cluster while the previous read-only cluster is still running, then the following steps will enable you to keep read-availability during recovery:

1. Verify that a catastrophe has occurred, and that access to the surviving members of the cluster in the surviving data center is possible.
2. Perform an online backup of the currently running, read-only, cluster.
3. Seed a new cluster (in the current example, two new cores) using the backup from the read-only cluster, as described in Seed a cluster.
4. When the new cluster is up, load balance your workload over to the new cluster.
5. Shutdown the old, read-only, cluster.

Once your chosen recovery procedure is completed for each instance, they will form a cluster that is available for reads and writes. It recommended at this point that other cluster members are incorporated into the cluster to improve its load handling and fault tolerance. See Deploy a cluster for details of how to configure instances to join the cluster from scratch.

# .1. Embedded usage

For users coming to Causal Clustering from Neo4j HA embedded, there are a small number of changes required. The Neo4j routing driver is used for routing and load balancing queries in server deployments (other setups are possible with 3rd party load balancers).

The driver also handles bookmarks, which are essential for causal consistency, and as such is a fundamental part of the Causal Clustering architecture. In an embedded deployment the driver can be used either for routing queries externally from another application into the embedded cluster, or using an embedded driver internally within the cluster.

The workload must be comprised, in its entirety, of Cypher statements. If your workload depends on the Java Core API for writing, then you have to package those pieces as procedures which are (remotely) invoked using Cypher, via the driver. Read-only queries can still access the Core API directly.



For a detailed tutorial on how to embed Neo4j in your Java application, see Neo4j Java Reference → Including Neo4j in your project.

# Appendix D: Deprecated security procedures

This appendix describes deprecated procedures for security management:

- Enterprise Edition
- Community Edition



The procedures described in this appendix have been deprecated and will be removed in a future release.

It is strongly recommended to migrate to the security features as described in Cypher Manual → Access Control

See also a worked example in Fine-grained access control.

## .D.1. Enterprise Edition

A subset of this functionality is also available in Community Edition. The table below includes an indication of which functions this is valid for. Refer to Community Edition for a complete description.

In Neo4j, native user and role management are managed by using built-in procedures through Cypher. This section gives a list of all the security procedures for user management along with some simple examples. Use Neo4j Browser or Neo4j Cypher Shell to run the examples provided.

The following table lists the available procedures:

| Procedure name | Description | Executable by role(s) | Available in Community Edition |
|---|---|---|---|
| `dbms.security.activateUser` | Activate a suspended user | `admin` | |
| `dbms.security.addRoleToUser` | Assign a role to the user | `admin` | |
| `dbms.security.changePassword` | Change the current user's password | `reader,editor,publisher,architect,admin` | ✔ |
| `dbms.security.changeUserPassword` | Change the given user's password | `admin` | |
| `dbms.security.createRole` | Create a new role | `admin` | |
| `dbms.security.createUser` | Create a new user | `admin` | ✔ |
| `dbms.security.deleteRole` | Delete the specified role. Any role assignments will be removed | `admin` | |
| `dbms.security.deleteUser` | Delete the specified user | `admin` | ✔ |

| Procedure name | Description | Executable by role(s) | Available in Community Edition |
|---|---|---|---|
| dbms.security.listRoles | List all available roles | admin | |
| dbms.security.listRolesForUser | List all roles assigned to the specified user | admin | |
| dbms.security.listUsers | List all local users | admin | ✔ |
| dbms.security.listUsersForRole | List all users currently assigned the specified role | admin | |
| dbms.security.removeRoleFromUser | Unassign a role from the user | admin | |
| dbms.security.suspendUser | Suspend the specified user | admin | |

## Activate a suspended user

An administrator is able to activate a suspended user so that the user is once again able to access the data in their original capacity.

### Syntax:

```
CALL dbms.security.activateUser( username ,  requirePasswordChange )
```

### Arguments:

| Name | Type | Description |
|---|---|---|
| username | String | This is the username of the user to be activated. |
| requirePasswordChange | Boolean | This is optional, with a default of true. If this is true, (i) the user will be forced to change their password when they next log in, and (ii) until the user has changed their password, they will be forbidden from performing any other operation. |

### Exceptions:

| |
|---|
| The current user is not an administrator. |
| The username does not exist in the system. |
| The username matches that of the current user (i.e. activating the current user is not permitted). |

### Considerations:

| |
|---|
| This is an idempotent procedure. |

*Example 157. Activate a suspended user*

The following example activates a user with the username '**jackgreen**'. When the user '**jackgreen**' next logs in, he will be required to change his password.

```
CALL dbms.security.activateUser('jackgreen')
```

## Assign a role to the user

An administrator is able to assign a role to any user in the system, thus allowing the user to perform a series of actions upon the data.

**Syntax:**

```
CALL dbms.security.addRoleToUser( roleName ,  username )
```

**Arguments:**

| Name | Type | Description |
|---|---|---|
| `roleName` | String | This is the name of the role to be assigned to the user. |
| `username` | String | This is the username of the user who is to be assigned the role. |

**Exceptions:**

| |
|---|
| The current user is not an administrator. |
| The username does not exist in the system. |
| The username contains characters other than alphanumeric characters and the '_' character. |
| The role name does not exist in the system. |
| The role name contains characters other than alphanumeric characters and the '_' character. |

**Considerations:**

| |
|---|
| This is an idempotent procedure. |

*Example 158. Assign a role to the user*

The following example assigns the role `publisher` to the user with username '**johnsmith**'.

```
CALL dbms.security.addRoleToUser('publisher', 'johnsmith')
```

## Change the current user's password



The procedure `dbms.security.changePassword(newPassword, requirePasswordChange)` has been entirely removed since the corresponding Cypher administration command also requires the old password, and thus is more secure. Please use `ALTER CURRENT USER SET PASSWORD FROM 'oldPassword' TO 'newPassword'`, documented in the Cypher Manual, instead.

## Change the given user's password

An administrator is able to change the password of any user within the system. Alternatively, the current user may change their own password.

**Syntax:**

```
CALL dbms.security.changeUserPassword( username ,  newPassword ,  requirePasswordChange )
```

**Arguments:**

| Name | Type | Description |
|---|---|---|
| `username` | String | This is the username of the user whose password is to be changed. |
| `newPassword` | String | This is the new password for the user. |
| `requirePasswordChange` | Boolean | This is optional, with a default of `true`. If this is `true`, (i) the user will be forced to change their password when they next log in, and (ii) until the user has changed their password, they will be forbidden from performing any other operation. |

**Exceptions:**

| |
|---|
| The current user is not an administrator and the username does not match that of the current user. |
| The username does not exist in the system. |
| The password is the empty string. |
| The password is the same as the user's previous password. |

**Considerations:**

| |
|---|
| This procedure may be invoked by the current user to change their own password, irrespective of whether or not the current user is an administrator. |
| This procedure may be invoked by an administrator to change another user's password. |

In addition to changing the user's password, this will terminate with immediate effect all of the user's sessions and roll back any running transactions.

*Example 159. Change a given user's password*

The following example changes the password of the user with the username '**joebloggs**' to '**h6u4%kr**'. When the user '**joebloggs**' next logs in, he will be required to change his password.

```
CALL dbms.security.changeUserPassword('joebloggs', 'h6u4%kr')
```

## Create a new role

An administrator is able to create custom roles in the system.

**Syntax:**

```
CALL dbms.security.createRole( roleName )
```

**Arguments:**

| Name | Type | Description |
|---|---|---|
| `roleName` | String | This is the name of the role to be created. |

**Exceptions:**

| |
|---|
| The current user is not an administrator. |
| The role name already exists in the system. |
| The role name is empty. |
| The role name contains characters other than alphanumeric characters and the '_' character. |
| The role name matches one of the native roles: `reader`, `publisher`, `architect`, and `admin`. |

*Example 160. Create a new role*

The following example creates a new custom role.

```
CALL dbms.security.createRole('operator')
```

## Create a new user

An administrator is able to create a new user. This action ought to be followed by assigning a role to the user, which is described here.

Syntax:

```
CALL dbms.security.createUser( username ,  password ,  requirePasswordChange )
```

Arguments:

| Name | Type | Description |
| --- | --- | --- |
| `username` | String | This is the user's username. |
| `password` | String | This is the user's password. |
| `requirePasswordChange` | Boolean | This is optional, with a default of `true`. If this is `true`, (i) the user will be forced to change their password when they log in for the first time, and (ii) until the user has changed their password, they will be forbidden from performing any other operation. |

Exceptions:

| |
| --- |
| The current user is not an administrator. |
| The username either contains characters other than the ASCII characters between `!` and `~`, or contains `:` and `,`. |
| The username is already in use within the system. |
| The password is the empty string. |

*Example 161. Create a new user*

The following example creates a user with the username '**johnsmith**' and password '**h6u4%kr**'. When the user '**johnsmith**' logs in for the first time, he will be required to change his password.

```
CALL dbms.security.createUser('johnsmith', 'h6u4%kr')
```

## Delete the specified role

An administrator is able to delete roles from the system.

Syntax:

```
CALL dbms.security.deleteRole( roleName )
```

Arguments:

| Name | Type | Description |
| --- | --- | --- |
| `roleName` | String | This is the name of the role to be deleted. |

**Exceptions:**

| |
| --- |
| The current user is not an administrator. |
| The role name does not exist in the system. |
| The role name matches one of the native roles: `reader`, `publisher`, `architect`, and `admin`. |

**Considerations:**

| |
| --- |
| Any role assignments will be removed. |

*Example 162. Delete the specified role*

The following example deletes the custom role '**operator**' from the system.

```
CALL dbms.security.deleteRole('operator')
```

## Delete the specified user

An administrator is able to delete permanently a user from the system. It is not possible to undo this action, so, if in any doubt, consider suspending the user instead.

**Syntax:**

```
CALL dbms.security.deleteUser( username )
```

**Arguments:**

| Name | Type | Description |
| --- | --- | --- |
| `username` | String | This is the username of the user to be deleted. |

**Exceptions:**

| |
| --- |
| The current user is not an administrator. |
| The username does not exist in the system. |
| The username matches that of the current user (i.e. deleting the current user is not permitted). |

**Considerations:**

| |
|---|
| It is not necessary to remove any assigned roles from the user prior to deleting the user. |
| Deleting a user will terminate with immediate effect all of the user's sessions and roll back any running transactions. |
| As it is not possible for the current user to delete themselves, there will always be at least one administrator in the system. |

*Example 163. Delete the specified user*

The following example deletes a user with the username '**janebrown**'.

```
CALL dbms.security.deleteUser('janebrown')
```

## List all available roles

An administrator is able to view all assigned users for each role in the system.

**Syntax:**

```
CALL dbms.security.listRoles()
```

**Returns:**

| Name | Type | Description |
|---|---|---|
| `role` | String | This is the name of the role. |
| `users` | List<String> | This is a list of the usernames of all users who have been assigned the role. |

**Exceptions:**

| |
|---|
| The current user is not an administrator. |

*Example 164. List all available roles*

The following example shows, for each role in the system, the name of the role and the usernames of all assigned users.

```
CALL dbms.security.listRoles()
```

```
+------------------------------+
| role        | users          |
+------------------------------+
| "reader"    | ["bill"]       |
| "architect" | []             |
| "admin"     | ["neo4j"]      |
| "publisher" | ["john","bob"] |
+------------------------------+
4 rows
```

## List all roles assigned to the specified user

Any active user is able to view all of their assigned roles. An administrator is able to view all assigned roles for any user in the system.

**Syntax:**

```
CALL dbms.security.listRolesForUser( username )
```

**Arguments:**

| Name | Type | Description |
|---|---|---|
| `username` | String | This is the username of the user. |

**Returns:**

| Name | Type | Description |
|---|---|---|
| `value` | String | This returns all roles assigned to the requested user. |

**Exceptions:**

| |
|---|
| The current user is not an administrator and the username does not match that of the current user. |
| The username does not exist in the system. |

**Considerations:**

| |
|---|
| This procedure may be invoked by the current user to view their roles, irrespective of whether or not the current user is an administrator. |
| This procedure may be invoked by an administrator to view the roles for another user. |

*Example 165. List all roles assigned to the specified user*

The following example lists all the roles for the user with username '**johnsmith**', who has the roles `reader` and `publisher`.

```
CALL dbms.security.listRolesForUser('johnsmith')
```

```
+-------------+
| value       |
+-------------+
| "reader"    |
| "publisher" |
+-------------+
2 rows
```

## List all local users

An administrator is able to view the details of every user in the system.

**Syntax:**

```
CALL dbms.security.listUsers()
```

**Returns:**

| Name | Type | Description |
|---|---|---|
| `username` | String | This is the user's username. |
| `roles` | List<String> | This is a list of roles assigned to the user. |
| `flags` | List<String> | This is a series of flags indicating whether the user is suspended or needs to change their password. |

**Exceptions:**

| |
|---|
| The current user is not an administrator. |

*Example 166. List all local users*

The following example shows, for each user in the system, the username, the roles assigned to the user, and whether the user is suspended or needs to change their password.

```
CALL dbms.security.listUsers()
```

```
+-------------------------------------------------------------------------+
| username | roles                     | flags                            |
+-------------------------------------------------------------------------+
| "neo4j"  | ["admin"]                 | []                               |
| "anne"   | []                        | ["password_change_required"]     |
| "bill"   | ["reader"]                | ["is_suspended"]                 |
| "john"   | ["architect","publisher"] | []                               |
+-------------------------------------------------------------------------+
4 rows
```

## List all users currently assigned the specified role

An administrator is able to view all assigned users for a role.

**Syntax:**

```
CALL dbms.security.listUsersForRole( roleName )
```

**Arguments:**

| Name | Type | Description |
| --- | --- | --- |
| `roleName` | String | This is the name of the role. |

**Returns:**

| Name | Type | Description |
| --- | --- | --- |
| `value` | String | This returns all assigned users for the requested role. |

**Exceptions:**

| |
| --- |
| The current user is not an administrator. |
| The role name does not exist in the system. |

*Example 167. List all users currently assigned the specified role*

The following example lists all the assigned users - '**bill**' and '**anne**' - for the role `publisher`.

```
CALL dbms.security.listUsersForRole('publisher')
```

```
+---------+
| value   |
+---------+
| "bill" |
| "anne" |
+---------+
2 rows
```

## Unassign a role from the user

An administrator is able to remove a role from any user in the system, thus preventing the user from performing upon the data any actions prescribed by the role.

**Syntax:**

```
CALL dbms.security.removeRoleFromUser( roleName ,  username )
```

**Arguments:**

| Name | Type | Description |
| --- | --- | --- |
| `roleName` | String | This is the name of the role which is to be removed from the user. |
| `username` | String | This is the username of the user from which the role is to be removed. |

**Exceptions:**

| |
|---|
| The current user is not an administrator. |
| The username does not exist in the system. |
| The role name does not exist in the system. |
| The username is that of the current user and the role is `admin`. |

**Considerations:**

| |
|---|
| If the username is that of the current user and the role name provided is `admin`, an error will be thrown; i.e. the current user may not be demoted from being an administrator. |
| As it is not possible for the current user to remove the `admin` role from themselves, there will always be at least one administrator in the system. |
| This is an idempotent procedure. |

*Example 168. Unassign a role from the user*

The following example removes the role `publisher` from the user with username '**johnsmith**'.

```
CALL dbms.security.removeRoleFromUser('publisher', 'johnsmith')
```

## Suspend the specified user

An administrator is able to suspend a user from the system. The suspended user may be activated at a later stage.

**Syntax:**

```
CALL dbms.security.suspendUser( username )
```

**Arguments:**

| Name | Type | Description |
|---|---|---|
| `username` | String | This is the username of the user to be suspended. |

**Exceptions:**

| |
|---|
| The current user is not an administrator. |
| The username does not exist in the system. |
| The username matches that of the current user (i.e. suspending the current user is not permitted). |

Considerations:

| |
|---|
| Suspending a user will terminate with immediate effect all of the user's sessions and roll back any running transactions. |
| All of the suspended user's attributes — assigned roles and password — will remain intact. |
| A suspended user will not be able to log on to the system. |
| As it is not possible for the current user to suspend themselves, there will always be at least one active administrator in the system. |
| This is an idempotent procedure. |

*Example 169. Suspend the specified user*

The following example suspends a user with the username '**billjones**'.

```
CALL dbms.security.suspendUser('billjones')
```

## .D.2. Community Edition

User and password management for Community Edition is a subset of the functionality available in Enterprise Edition. The following is true for user management in Community Edition:

- It is possible to create multiple users.
- All users assume the privileges of an `admin` for the available functionality.

Users are managed by using built-in procedures through Cypher. This section gives a list of all the security procedures for user management along with some simple examples. Use Neo4j Browser or Neo4j Cypher Shell to run the examples provided. Unless stated otherwise, all arguments to the procedures described in this section must be supplied.

| Name | Description |
|---|---|
| dbms.security.changePassword | Change the current user's password |
| dbms.security.createUser | Add a user |
| dbms.security.deleteUser | Delete a user |
| dbms.security.listUsers | List all users |

### Change the current user's password



The procedure `dbms.security.changePassword(newPassword, requirePasswordChange)` has been entirely removed since the corresponding Cypher administration command also requires the old password, and thus is more secure. Please use `ALTER CURRENT USER SET PASSWORD FROM 'oldPassword' TO 'newPassword'`, documented in the Cypher Manual,

instead.

## Add a user

The current user is able to add a user to the system.

**Syntax:**

```
CALL dbms.security.createUser( username ,  password ,  requirePasswordChange )
```

**Arguments:**

| Name | Type | Description |
|---|---|---|
| `username` | String | This is the user's username. |
| `password` | String | This is the user's password. |
| `requirePasswordChange` | Boolean | This is optional, with a default of `true`. If this is `true`, (i) the user will be forced to change their password when they log in for the first time, and (ii) until the user has changed their password, they will be forbidden from performing any other operation. |

**Exceptions:**

| |
|---|
| The username either contains characters other than the ASCII characters between `!` and `~`, or contains `:` and `,`. |
| The username is already in use within the system. |
| The password is the empty string. |

*Example 170. Add a user*

The following example creates a user with the username '**johnsmith**' and password '**h6u4%kr**'. When the user '**johnsmith**' logs in for the first time, he will be required to change his password.

```
CALL dbms.security.createUser('johnsmith', 'h6u4%kr', true)
```

## Delete a user

The current user is able to delete permanently a user from the system.

**Syntax:**

```
CALL dbms.security.deleteUser( username )
```

Arguments:

| Name | Type | Description |
| --- | --- | --- |
| `username` | String | This is the username of the user to be deleted. |

Exceptions:

| |
| --- |
| The username does not exist in the system. |
| The username matches that of the current user (i.e. deleting the current user is not permitted). |

Considerations:

| |
| --- |
| Deleting a user will terminate with immediate effect all of the user's sessions and roll back any running transactions. |
| As it is not possible for the current user to delete themselves, there will always be at least one user in the system. |

*Example 171. Delete a user*

The following example deletes a user with the username '**janebrown**'.

```
CALL dbms.security.deleteUser('janebrown')
```

## List all native users

The current user is able to view the details of every user in the system.

Syntax:

`CALL dbms.security.listUsers()`

Returns:

| Name | Type | Description |
| --- | --- | --- |
| `username` | String | This is the user's username. |
| `flags` | List<String> | This is a flag indicating whether the user needs to change their password. |

*Example 172. List all users*

The following example shows the username for each user in the system, and whether the user needs to change their password.

```
CALL dbms.security.listUsers()
```

```
+----------------------------------------+
| username | flags                       |
+----------------------------------------+
| "neo4j"  | []                          |
| "anne"   | ["password_change_required"] |
| "bill"   | []                          |
+----------------------------------------+
3 rows
```

# Appendix E: Query routing decisions

## .E.1. Introduction

A query that arrives at a Neo4j server, over the bolt protocol from a driver, undergoes *query routing*. Query routing is the process of deciding with which Cypher executor (database) and at which physical location the query should be executed.

## .E.2. Routing decision tree

Before the query is executed, these are the decisions taken during query routing:

*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 the Fabric virtual database, then the context type is *Fabric*.
  - This also allows the query to target multiple databases.
- If the target database is a Fabric graph, then the context type is *External*.

*Step 4: Determine the location of execution*

- If context type is *Internal*, then ...
  - if the URI scheme is `bolt://` (routing disabled), then location is *Local*.
  - if transaction mode is `READ`, then location is *Local*.
  - if transaction mode is `WRITE`, then ...

        - if the local member is the leader for the database, then location is *Local*.
        - if the another member is the leader for the database, then ...
            - if server-side routing is enabled, then location is *Remote* (using the routing advertised address of that member).
            - if not, then fail.
    - If context type is *Fabric*, 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 member.
- 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.

## .E.3. Illustrated routing decision tree

*Figure 53. Illustrated routing decision tree*

# License

Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)

*You are free to*

*Share*

copy and redistribute the material in any medium or format

*Adapt*

remix, transform, and build upon the material

The licensor cannot revoke these freedoms as long as you follow the license terms.

*Under the following terms*

*Attribution*

You must give appropriate credit, provide a link to the license, and indicate if changes were made. You 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.