



The Neo4j Operations Manual

v4.2

Table of Contents

1. Introduction	2
1.1. Community Edition	2
1.2. Enterprise Edition	2
1.3. Versioning	2
1.4. Cypher	2
1.5. Interaction	3
1.5.1. The official Neo4j Drivers	3
1.5.2. Other tools	3
1.6. Neo4j feature details	3
1.6.1. Neo4j key features	3
1.6.2. Performance and scalability	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	7
2.2. Neo4j Desktop	7
2.3. Linux installation	8
2.3.1. Debian	8
Installation	8
File locations	12
Operation	12
2.3.2. Deploy Neo4j using the Neo4j RPM package	12
Install on Red Hat, CentOS, Fedora or Amazon Linux	12
Install on SUSE	14
Offline installation	14
2.3.3. Linux tarball installation	15
Unix console application	15
Linux service	16
Setting the number of open files	16
2.3.4. Neo4j system service	17
Configuration	17
Starting the service automatically on system start	17
Controlling the service	17
Log	18
2.4. macOS installation	18
2.4.1. Unix console application	18
2.4.2. macOS service	19
2.4.3. macOS file descriptor limits	19
2.5. Windows installation	19
2.5.1. Windows console application	19
2.5.2. Windows service	20
Java options	20
2.5.3. Windows PowerShell module	20

System requirements	21
Managing Neo4j on Windows	21
How do I import the module?	21
How do I get help about the module?	21
Example usage	22
Common PowerShell parameters	22
3. Cloud deployments	23
3.1. Neo4j cloud VMs	23
3.1.1. Basics and file Locations	23
3.1.2. VM configuration	23
3.1.3. Configuration via VM tags	23
3.1.4. Interacting with the Neo4j Service	24
3.2. Neo4j on Amazon EC2	24
3.2.1. Neo4j deployment automation on AWS	24
Prerequisites	24
CloudFormation	24
Creating a CloudFormation stack	25
Deploying Neo4j Enterprise Standalone	25
Deploying Neo4j Enterprise Causal Cluster	25
Deploying Neo4j Community Standalone	26
Checking to see if your instance is up	26
Cleaning up and removing your stack	27
3.3. Neo4j on Google Cloud Platform	27
3.3.1. Test drive	27
3.3.2. Single instances (VM-based)	27
Prerequisites	27
Create a firewall rule to access your instance	27
Create a Google compute instance from the Neo4j public image	28
Access your new instance	28
Access your instance via SSH	28
Deleting the instance	29
3.3.3. Causal Clusters (VM-based)	29
Prerequisites	29
Deploy Neo4j via the GCP Marketplace	29
Start using Neo4j Browser	29
Access your instance via SSH	30
Your cluster default configuration	30
What's next	30
Terminating the deployment	30
3.3.4. Neo4j deployments automation on Google Cloud Platform (GCP)	30
Prerequisites	30
Google Cloud Deployment Manager	31
Creating a Deployment Manager stack	31
Deploying Neo4j Enterprise Edition with a Causal Cluster	31
Deploying Neo4j Enterprise (or Community) Edition in standalone mode	32
3.4. Neo4j on Microsoft Azure	34
3.4.1. Single instances (VM-based)	34
Prerequisites	34
Deploy Neo4j via the Azure Marketplace	34

Access your new instance.....	34
Access your instance via SSH.....	34
Deleting the instance	35
3.4.2. Causal Clusters (VM-based)	35
Prerequisites	35
Deploy Neo4j from the Azure Marketplace.....	35
Start using Neo4j Browser	35
Access your instance via SSH.....	36
Your cluster default configuration	36
What's next.....	36
Terminating the deployment.....	36
3.4.3. Neo4j deployments automation on Azure.....	36
Prerequisites	36
Azure Resource Manager.....	36
Creating an ARM deployment job	37
Deploying Neo4j Enterprise Causal Cluster.....	37
Deploying Neo4j Enterprise Standalone	38
Cleaning up and removing your deployment	40
4. Docker	41
4.1. Introduction.....	41
4.1.1. Neo4j editions	41
Neo4j Enterprise Edition license	41
4.1.2. Using the Neo4j Docker image	42
4.1.3. Using NE04J_AUTH to set an initial password	42
4.1.4. Running Neo4j as a non-root user	43
4.2. Configuration.....	43
4.2.1. Environment variables	43
4.2.2. Mounting the /conf volume	44
4.2.3. Building a new image with Neo4j as a base image.....	44
4.3. Clustering.....	45
4.3.1. Deploy a Causal Cluster with Docker Compose	45
4.3.2. Deploy a Causal Cluster using environment variables.....	47
Causal Cluster environment variables	48
Set up a Causal Cluster on a single Docker host.....	48
Set up a Causal Cluster on multiple Docker hosts	49
4.4. Docker specific operations	50
4.4.1. Use Neo4j Admin.....	50
4.4.2. Use Neo4j Import	50
4.4.3. Use Neo4j Admin for memory recommendations.....	51
4.4.4. Use Cypher Shell	51
Retrieve data from a database in a Neo4j Docker container	51
Pass a Cypher script file to a Neo4j Docker container	52
4.4.5. Install user-defined procedures	53
4.4.6. Configure Neo4j Labs plugins	53
4.5. Security.....	54
4.5.1. SSL Encryption	54
Set up your certificate folders	54
Configure SSL via <i>neo4j.conf</i>	55
Configure SSL via Docker environment variables.....	55

4.6. Docker maintenance operations	56
4.6.1. Dump and load a Neo4j database (offline)	56
4.6.2. Back up and restore a Neo4j database (online)	57
Back up a database	57
Restore a database	58
4.6.3. Upgrade Neo4j on Docker	58
4.7. Docker specific configuration settings.....	59
5. Configuration	68
5.1. The neo4j.conf file.....	68
5.1.1. Introduction	68
5.1.2. Syntax	68
5.1.3. JVM-specific configuration settings.....	69
5.1.4. List currently active settings	69
5.1.5. Command expansion	69
Usage.....	70
Enabling.....	70
5.2. File locations	70
5.2.1. Default file locations	70
5.2.2. Customize your file locations.....	72
5.2.3. File permissions.....	72
5.3. Ports	73
5.3.1. Backup.....	74
5.3.2. HTTP	74
5.3.3. HTTPS	75
5.3.4. Bolt.....	75
5.3.5. Causal Cluster	75
5.3.6. Graphite monitoring	76
5.3.7. Prometheus monitoring.....	76
5.3.8. JMX monitoring	77
5.3.9. Remote debugging	77
5.4. Set an initial password	77
5.5. Password and user recovery.....	78
5.5.1. Recover a lost password.....	78
5.5.2. Recover an unassigned admin role.....	79
5.5.3. Recover the admin role	80
5.6. Configure connectors.....	82
5.6.1. Available connectors.....	82
5.6.2. Configuration options.....	82
5.6.3. Options for Bolt thread pooling	83
5.6.4. Defaults for addresses	83
5.7. Dynamic settings.....	84
5.7.1. Introduction	85
5.7.2. Discover dynamic settings	85
5.7.3. Update dynamic settings	85
5.7.4. Dynamic settings reference	86
5.8. Transaction logs	87
6. Manage databases	89
6.1. Introduction	89
6.1.1. Concepts.....	89

6.1.2. The system database	90
6.1.3. The default database	91
6.2. Administration and configuration	91
6.2.1. Administrative commands	92
6.2.2. Configuration parameters	93
6.3. Queries	93
6.3.1. Show the status of a specific database	94
6.3.2. Show the status of all databases	94
6.3.3. Show the status of the default database	95
6.3.4. Create a database	96
6.3.5. Switch a database	97
6.3.6. Create or replace a database	97
6.3.7. Stop a database	99
6.3.8. Start a database	99
6.3.9. Drop or remove a database	100
6.4. Error handling	101
6.4.1. Observing errors	101
6.4.2. Database states	102
6.4.3. Retrying failed operations	103
6.4.4. Using quarantine in a cluster for fixing errors	104
6.5. Databases in a Causal Cluster	106
6.5.1. Running Cypher administrative commands from Cypher Shell on a Causal Cluster	106
7. Clustering	110
7.1. Introduction	110
7.1.1. Overview	110
7.1.2. Operational view	111
Core Servers	111
Read Replicas	112
7.1.3. Causal consistency	112
7.1.4. Summary	113
7.2. Deploy a cluster	114
7.2.1. Introduction	114
7.2.2. Configure a Core-only cluster	114
7.2.3. Add a Core Server to an existing cluster	115
7.2.4. Add a Read Replica to an existing cluster	116
7.3. Seed a cluster	116
7.3.1. Introduction	116
7.3.2. Seed from backups	117
7.3.3. Seed using the import tool	118
7.4. Discovery	119
7.4.1. Overview	119
Discovery using a list of server addresses	119
Discovery using DNS with multiple records	119
Discovery in Kubernetes	120
7.5. Intra-cluster encryption	120
7.5.1. Introduction	120
7.5.2. Example deployment	121
Generate and install cryptographic objects	121
Configure the cluster SSL policy	122

Validate the secure operation of the cluster	123
7.6. Internals of clustering	123
7.6.1. Elections and leadership	123
7.6.2. Leadership balancing	124
7.6.3. Multi-database and the reconciler	124
7.6.4. Server-side routing	124
7.6.5. Store copy	125
7.6.6. On-disk state	126
7.7. Settings reference.....	126
7.7.1. Multi-data center settings	129
8. Fabric	131
8.1. Introduction.....	131
8.1.1. Overview.....	131
8.1.2. Fabric concepts	131
The fabric database.....	131
Fabric graphs	132
8.1.3. Deployment examples	132
Development deployment.....	132
Cluster deployment with no single point of failure	132
Multi-cluster deployment.....	133
8.2. Configuration.....	134
8.2.1. Fabric database setup	134
Local development setup example.....	134
Remote development setup example	135
Naming graphs.....	136
Cluster setup with no single point of failure example	137
Cluster routing context.....	138
8.2.2. Authentication and authorization.....	138
Credentials	138
User and role administration.....	138
Privileges on the Fabric database	138
8.2.3. Important settings.....	138
System settings	138
Graph settings	139
Drivers settings	139
8.3. Queries.....	140
8.3.1. Query a single graph.....	140
8.3.2. Query multiple graphs	141
8.3.3. Query all graphs	141
8.3.4. Query result aggregation	141
8.3.5. Correlated subquery.....	141
8.3.6. Updating query	142
8.3.7. Mapping functions	142
8.3.8. Fabric built-in functions	143
8.4. Further considerations.....	143
9. Backup	145
9.1. Backup planning	145
9.1.1. Introduction	145
9.1.2. Online and offline backups	146

9.1.3. Server configuration	146
9.1.4. Databases to backup	146
9.1.5. Storage considerations.....	146
9.1.6. Cluster considerations	147
9.1.7. Using SSL/TLS for backups.....	147
9.1.8. Additional files to back up	147
9.2. Perform a backup	148
9.2.1. Backup command	148
9.2.2. Backup process	150
9.2.3. Memory configuration	151
9.3. Restore a backup	151
9.3.1. Restore command.....	152
9.3.2. Restore a standalone server	153
9.3.3. Restore a cluster	153
10. Upgrade.....	155
10.1. Plan your upgrade	155
10.1.1. Important information.....	155
Understanding upgrades and migration	155
Supported upgrade paths	156
Limitations	156
10.1.2. Upgrade checklist	156
Back up your current deployment	157
Prepare a new <i>neo4j.conf</i> file	157
Perform a test upgrade	157
Monitor the logs	157
Update metrics	157
10.2. Upgrade a single instance.....	158
10.2.1. Prerequisites	158
10.2.2. Upgrade Neo4j using the tarball or ZIP file distribution	158
10.2.3. Upgrade Neo4j using the Debian or RPM distribution	158
10.2.4. Post-upgrade tasks	159
10.3. Upgrade a Causal Cluster	159
10.3.1. Offline upgrade	159
10.3.2. Rolling upgrade	161
Rolling upgrade for fixed servers	161
Rolling upgrade for replaceable resources	163
10.3.3. Post-upgrade	165
Restore to the original state.....	165
Upgrade the shared state	166
When is this needed?	166
How to upgrade the system database schema	167
Troubleshooting	168
11. Authentication and authorization	170
11.1. Introduction.....	170
11.2. Built-in roles	171
11.3. Fine-grained access control.....	173
11.3.1. The data model	173
11.3.2. Security.....	174
11.3.3. Access control using built-in roles	175

11.3.4. Sub-graph access control using privileges	176
Privileges of itadmin	177
Privileges of researcher	179
Privileges of doctor	182
Privileges of receptionist	184
Privileges of nurses	186
Privileges of junior nurses	189
Building a custom administrator role.....	191
11.4. Integration with LDAP directory services	193
11.4.1. Introduction	193
11.4.2. LDAP configuration parameters	193
11.4.3. Set Neo4j to use LDAP	194
11.4.4. Map the LDAP groups to the Neo4j roles	194
11.4.5. Configure Neo4j to use Active Directory	194
Configure Neo4j to support LDAP user ID authentication (Option 1)	194
Configure Neo4j to support sAMAccountName authentication (Option 2)	195
Configure Neo4j to support sAMAccountName authentication (Option 3)	196
11.4.6. Configure Neo4j to use OpenLDAP	196
11.4.7. Verify the LDAP configuration.....	196
11.4.8. The auth cache	197
11.4.9. Available methods of encryption	198
Use LDAP with encryption via StartTLS	198
Use LDAP with encrypted LDAPS	198
11.4.10. Use a self-signed certificate (SSL) in a test environment	199
11.5. Manage procedure and user-defined function permissions	199
11.5.1. Introduction	199
11.5.2. Manage procedure permissions.....	199
11.5.3. Manage user-defined function permissions	200
11.5.4. Manage procedure and user-defined function permissions from config setting	201
11.6. Terminology.....	202
12. Security	204
12.1. Securing extensions	204
12.1.1. White listing	204
12.2. SSL framework	205
12.2.1. Introduction	205
12.2.2. Certificates	205
12.2.3. Configuration.....	205
Settings	205
12.2.4. Choosing an SSL provider	207
12.2.5. Terminology.....	207
12.3. Browser credentials handling.....	209
12.4. Security checklist	209
13. Monitoring.....	211
13.1. Metrics	212
13.1.1. Types of metrics	212
Global metrics	212
Database metrics.....	213
13.1.2. Expose metrics	213
Enable metrics logging	213

Graphite	214
Prometheus	214
CSV files.....	215
JMX MBeans	215
13.1.3. Metrics reference	215
General-purpose metrics	215
Metrics specific to Causal Clustering	219
Java Virtual Machine Metrics	220
13.2. Logging.....	221
13.2.1. General logging	221
Log files.....	222
13.2.2. User log file configuration	222
13.2.3. Debug log file configuration	222
13.2.4. Garbage collection log file configuration.....	223
13.2.5. HTTP request log file configuration	223
13.2.6. Security events logging.....	223
Security log file configuration	224
13.2.7. Query logging.....	224
Query log file configuration	225
Attach metadata to a query.....	226
13.2.8. Transaction logging.....	227
Log location	227
Log rotation	228
Log retention	228
Log pruning	229
13.3. Query management.....	230
13.3.1. List all running queries.....	230
13.3.2. List all active locks for a query.....	232
13.3.3. Terminate multiple queries.....	233
13.3.4. Terminate a single query	234
13.4. Transaction management.....	235
13.4.1. Configure transaction timeout	235
13.4.2. Configure lock acquisition timeout.....	236
13.4.3. List all running transactions	236
13.5. Connection management	238
13.5.1. List all network connections	238
13.5.2. Terminate multiple network connections	239
13.5.3. Terminate a single network connection	240
13.6. Background job management	241
13.6.1. Listing active background jobs	241
13.6.2. Listing failed job executions	243
13.7. Monitoring a Causal Cluster	244
13.7.1. Procedures for monitoring a Causal Cluster.....	245
Find out the role of a cluster member	245
Gain an overview over the instances in the cluster	246
Get routing recommendations	246
13.7.2. Endpoints for status information	247
Adjusting security settings for Causal Clustering endpoints.....	248
Unified endpoints	248

13.8. Monitoring individual database states	251
13.8.1. Listing Databases	251
13.8.2. Listing a single database	255
14. Performance	257
14.1. Memory configuration	257
14.1.1. Overview.....	257
14.1.2. Considerations.....	259
14.1.3. Capacity planning	260
14.1.4. Limit transaction memory usage	262
14.2. Index configuration	263
14.2.1. Introduction	263
14.2.2. B-tree indexes	264
Limitations	264
Index migration	265
Procedures to create index and index backed constraint.....	266
14.2.3. Full-text indexes	267
Configuration	267
14.3. Tuning of the garbage collector	268
14.4. Bolt thread pool configuration	269
14.4.1. How thread pooling works.....	269
14.4.2. Configuration options.....	270
14.4.3. How to size your Bolt thread pool	270
14.5. Linux file system tuning	271
14.6. Disks, RAM and other tips	271
14.6.1. Storage	271
14.6.2. Page cache	271
Active page cache warmup	272
Checkpoint IOPS limit	273
14.7. Statistics and execution plans.....	273
14.7.1. Statistics	273
14.7.2. Execution plans	274
14.8. Space reuse	275
14.8.1. ID files.....	276
14.8.2. Reclaim unused space	276
15. Tools.....	281
15.1. Neo4j Admin	281
15.1.1. Introduction	281
15.1.2. Syntax and commands.....	281
15.1.3. Environment variables	282
15.1.4. Exit codes	283
15.2. Consistency checker.....	283
15.3. Neo4j Admin report	285
15.4. Display store information	286
15.4.1. Syntax and examples	286
15.4.2. Store format versions and limits.....	288
Standard store format	288
Aligned store format.....	288
High limit store format	288
15.5. Memory recommendations	289

15.6. Import	290
15.6.1. Syntax	291
15.6.2. CSV file header format	291
Header files	291
Properties	292
Node files	294
Relationship files	295
Using ID spaces	296
Skipping columns	297
Import compressed files	298
15.6.3. Options	298
Output	302
15.7. Dump and load databases	302
15.8. Unbind a Core Server	303
15.9. Copy a database	304
15.9.1. Sharding data with the copy command	307
15.10. Cypher Shell	309
15.10.1. About Cypher Shell CLI	309
15.10.2. Syntax	309
15.10.3. Query parameters	312
15.10.4. Transactions	313
15.10.5. Procedures	314
15.10.6. Supported operating systems	315
Appendix A: Reference	316
Appendix B: Tutorials	317
B.1. Set up a local Causal Cluster	317
B.2. Use the Import tool	323
B.2.1. Basic example	324
B.2.2. Customizing configuration options	325
B.2.3. Using separate header files	325
B.2.4. Multiple input files	326
Using regular expressions for specifying multiple input files	328
B.2.5. Types and labels	328
Using the same label for every node	328
Using the same relationship type for every relationship	329
B.2.6. Property types	330
B.2.7. ID handling	330
Working with sequential or auto incrementing identifiers	330
B.2.8. Bad input data	331
Relationships referring to missing nodes	331
Multiple nodes with same ID within same ID space	332
B.3. Set up and use Fabric	333
B.3.1. Model your data for Fabric	333
Northwind data	333
The model	334
Remodeling the Northwind dataset	335
B.3.2. Configure Fabric with three databases	337
Create three databases	337
Configure Fabric	339

B.3.3. Import data in your databases	339
Load the Product catalog in db0	340
Load EU customers and related orders in db1	340
Load AME customers and related orders in db2	341
B.3.4. Retrieve data with a single Cypher query	342
Query a single database	343
Query across multiple shards	343
Query across federation and shards	344
B.3.5. The end	345
Appendix C: Advanced Causal Clustering	346
C.1. Causal Clustering lifecycle	346
C.1.1. Introduction	346
C.1.2. Discovery protocol	347
C.1.3. Core membership	348
C.1.4. Read Replica membership	348
C.1.5. Transacting via the Raft protocol	349
C.1.6. Catchup protocol	350
C.1.7. Read Replica shutdown	351
C.1.8. Core shutdown	351
C.2. Multi-data center	351
C.2.1. Licensing for multi-data center operations	352
C.2.2. Multi-data center design	352
Introduction	353
Core Server deployment scenarios	353
Allowing Read Replicas to catch up from other Read Replicas	356
C.2.3. Multi-data center operations	360
Enable multi-data center operations	361
Server groups	361
Strategy plugins	362
C.2.4. Multi-data center load balancing	366
Introduction	366
Prerequisite configuration	367
The load balancing framework	367
Load balancing examples	369
C.2.5. Data center disaster recovery	370
Data center loss scenario	371
Procedure for recovering from data center loss	372
C.3. Embedded usage	373
Appendix D: Deprecated security procedures	374
D.1. Enterprise Edition	374
D.1.1. Activate a suspended user	375
D.1.2. Assign a role to the user	375
D.1.3. Change the current user's password	376
D.1.4. Change the given user's password	376
D.1.5. Create a new role	377
D.1.6. Create a new user	378
D.1.7. Delete the specified role	378
D.1.8. Delete the specified user	379
D.1.9. List all available roles	380

D.1.10. List all roles assigned to the specified user	380
D.1.11. List all local users	381
D.1.12. List all users currently assigned the specified role	382
D.1.13. Unassign a role from the user	383
D.1.14. Suspend the specified user.....	384
D.2. Community Edition	384
D.2.1. Change the current user's password.....	385
D.2.2. Add a user	385
D.2.3. Delete a user.....	386
D.2.4. List all native users.....	386

This is the operations manual for Neo4j version 4.2, authored by the Neo4j Team.

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.
- [Cloud deployments](#) — Information on how to deploy Neo4j on cloud platforms.
- [Docker](#) — Instructions on how to use Neo4j on Docker.
- [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](#) — Instructions on setting up Neo4j backups.
- [Upgrade](#) — Instructions on upgrading Neo4j.
- [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.

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.

There are two editions of Neo4j to choose from, the [Community Edition](#) and the [Enterprise Edition](#). The Enterprise Edition includes all that Community Edition has to offer, plus extra enterprise requirements such as backups, clustering, and failover capabilities.

1.1. Community Edition

The Community Edition is a fully functional edition of Neo4j, suitable for single-instance deployments. It has full support for 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.

1.2. 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 solutions and critical internal solutions.

1.3. 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.

1.4. Cypher

Cypher is a declarative query language for graphs. Neo4j uses the property graph approach, where relationships are stored alongside the data in the model, and not computed at query time. Cypher is a powerful, graph-optimized query language that understands, and takes advantage of, these stored connections. When trying to find patterns or insights in data, Cypher queries are often much simpler and easier to write than massive SQL JOINs. Since Neo4j does not have tables, there are no JOINs to worry about.

For more details, see the [Cypher Manual](#) □ [Cypher - The Graph Query Language](#).

1.5. Interaction

The recommended way of programmatically interacting with the database is either through the official [Driver Manual](#) [Neo4j Drivers](#) or using the [Java Reference](#) [Neo4j Java API](#). Neo4j provides an ACID-compliant transactional backend for your applications.

1.5.1. The official Neo4j Drivers

The official Neo4j Drivers interacts with Neo4j via the [Bolt protocol](#) (<https://neo4j-drivers.github.io/>).

- [Neo4j Java Driver](#)

For Spring-powered applications there is also [Spring Data Neo4j](#).

- [Neo4j JavaScript Driver](#)
- [Neo4j Python Driver](#)
- [Neo4j .NET Driver](#)
- [Neo4j Go Driver](#)



See the [Neo4j Download Center - Drivers](#) for more links.

1.5.2. Other tools

- [Neo4j Cypher Shell](#) - Command line tool for Cypher queries. [Neo4j Download Center - Cypher Shell](#).
- Neo4j Browser - Interact with Neo4j, create Cypher queries, and basic visualization capabilities.
- Neo4j Desktop - Developer IDE or Management Environment for Neo4j instances. [Neo4j Download Center - Neo4j Desktop](#).
- [Neo4j Bloom](#) - Explore and visualize graph data. [Neo4j Download Center - Neo4j Bloom](#).

1.6. Neo4j feature details

1.6.1. Neo4j key features

Table 1. 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#, Java, JavaScript & Python ^[1]	✔	✔
High-performance native API	✔	✔
High-performance caching	✔	✔
Cost-based query optimizer	✔	✔
Graph algorithms library to support AI initiatives ^[1]	✔	✔

Feature	Community Edition	Enterprise Edition
Fast writes via native label indexes	✔	✔
Composite indexes	✔	✔
Full-text node & relationship indexes	✔	✔
Store copy	✔	✔
Auto-reuse of space	✔	✔
Multiple databases (beyond the <code>system</code> and default databases)	✖	✔
<i>Slotted and Pipelined</i> 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]Must be downloaded and installed separately.

1.6.2. Performance and scalability

Table 2. Performance and scalability features

Feature	Community Edition	Enterprise Edition
Causal Clustering for global scale applications	✖	✔
Enterprise lock manager accesses all cores on server	✖	✔
Intra-cluster encryption	✖	✔
Offline backups	✔	✔
Online backups	✖	✔
Encrypted backups	✖	✔
Rolling upgrades	✖	✔
Automatic cache warming	✖	✔
Routing and load balancing with Neo4j Drivers	✖	✔
Advanced monitoring	✖	✔
Graph size limitations	34 billion nodes, 34 billion relationships, and 68 billion properties	No limit
Bulk import tool	✔	✔
Bulk import tool, resumable	✖	✔

Chapter 2. Installation

This chapter describes installation of Neo4j in different deployment contexts, such as Linux, macOS, and Windows.

The topics described are:

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

Installation-free options



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

Neo4j can be run in a **Docker** container. For information on running Neo4j on Docker, see [Docker](#).

2.1. System requirements

This section provides an overview of the system requirements for running Neo4j in a production environment.

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.



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

2.1.1. Supported platforms

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

2.1.2. Hardware requirements

In terms of 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: <ul style="list-style-type: none"> • Neo4j workloads tend significantly toward random reads. • Select media with low average seek time: SSD over spinning disks. • 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. 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. 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 16.04+	OpenJDK 11, OracleJDK 11, and ZuluJDK 11
Debian 9+	OpenJDK 11, 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-premise environments.

Operating System	Supported JDK
Ubuntu Server 16.04+	OpenJDK 11, OracleJDK 11, and ZuluJDK 11
Red Hat Enterprise Linux Server 7.5+	Red Hat OpenJDK 11, Oracle JDK 11, and ZuluJDK 11
CentOS Server 7.7	OpenJDK 11
Amazon Linux 2 AMI	Amazon Corretto 11, OpenJDK 11, and OracleJDK 11
Windows Server 2016+	OracleJDK 11 and ZuluJDK 11

2.1.4. Filesystem

For proper ACID behavior, the filesystem must support flush (*fsync*, *fdatasync*). See [Linux file system tuning](#) for a discussion on how to configure the filesystem in Linux for optimal performance.

2.1.5. Java

It is required to have a pre-installed, compatible Java Virtual Machine (JVM), to run a Neo4j instance.

Table 9. Neo4j version and JVM requirements.

Neo4j Version	JVM compliancy
3.x	Java SE 8 Platform Specifikation
4.x	Java SE 11 Platform Specification

[Neo4j Desktop](#) is available for developers and personal users. Neo4j Desktop is bundled with a JVM.

2.2. Neo4j Desktop

This section introduces Neo4j Desktop.

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

To install Neo4j Desktop, go to [Neo4j Download Center](#) and follow the instructions.



While most functionality is the same, the instructions in this manual are not written for Neo4j Desktop. For example, file locations for a database installed via Neo4j Desktop will be different from those described here.

Neo4j Desktop is not suited for production environments.

2.3. Linux installation

This section describes how to install Neo4j on Linux using Debian or RPM packages, or from a Tar archive.

This section describes the following:

- [Install Neo4j on Debian and Debian-based distributions](#)
 - [Installation](#)
 - [File locations](#)
 - [Operation](#)
- [Deploy Neo4j using the Neo4j RPM package](#)
 - [Install on Red Hat, CentOS, Fedora or Amazon Linux](#)
 - [Standard installation](#)
 - [Non-interactive installation of Neo4j Enterprise Edition](#)
 - [Install on SUSE](#)
 - [Offline installation](#)
- [Install Neo4j on Linux from a tarball](#)
 - [Unix console application](#)
 - [Linux service](#)
 - [Setting the number of open files](#)
- [Install Neo4j as a system service](#)
 - [Configuration](#)
 - [Controlling the service](#)
 - [Log](#)

2.3.1. Debian

This section describes how to install Neo4j on Debian, and Debian-based distributions like Ubuntu, using the Neo4j Debian package.

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, Debian 9+ and Ubuntu 16.04+ only)

Neo4j 4.2 requires the Java 11 runtime. Java 11 is not included in Ubuntu 16.04 LTS or Debian 9 (stretch) and will have to be set up manually prior to installing or upgrading to Neo4j 4.2, as described below. Debian 9 users can find OpenJDK 11 in [backports](#). Debian 10 and Ubuntu 18.04 onwards already have 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.

Java 11 on Debian 9

Add the line `deb http://httpredir.debian.org/debian stretch-backports main` to a file with the ".list" extension in `/etc/apt/sources.list.d/`. Then run `apt-get update`:

```
echo "deb http://httpredir.debian.org/debian stretch-backports main" | sudo tee -a  
/etc/apt/sources.list.d/stretch-backports.list  
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.

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 4.2.0 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 4.2' | sudo tee -a /etc/apt/sources.list.d/neo4j.list
sudo apt-get update
```

- To use the repository for milestone releases, alpha, beta, release candidate and preview versions of Neo4j, run:

```
wget -O - https://debian.neo4j.com/neotechnology.gpg.key | sudo apt-key add -
echo 'deb https://debian.neo4j.com testing 4.2' | 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
```

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=1:4.2.0
```

To install Neo4j Enterprise Edition:

```
sudo apt-get install neo4j-enterprise=1:4.2.0
```

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

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.



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

- The Cypher Shell package can be downloaded from [Neo4j Download Center](#).
- For information on supported versions of Java, see [System requirements](#).

1. Run the following to download the required Debian package:

□ Neo4j Enterprise Edition:

```
curl -O https://dist.neo4j.org/deb/neo4j-enterprise_4.2.0_all.deb
```

□ Neo4j Community Edition:

```
curl -O https://dist.neo4j.org/deb/neo4j_4.2.0_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
<code>NEO4J_SHUTDOWN_TIMEOUT</code>	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.
<code>NEO4J_ULIMIT_NOFILE</code>	60000	Maximum number of file handles that can be opened by the Neo4j process.

2.3.2. Deploy Neo4j using the Neo4j RPM package

This section describes how to deploy Neo4j using the Neo4j RPM package on Red Hat, CentOS, Fedora, or Amazon Linux distributions.

Install on Red Hat, CentOS, Fedora or Amazon Linux

Standard installation

1. Add the repository.

Use the following as [root](#) to add the repository:

```
rpm --import https://debian.neo4j.com/neotechnology.gpg.key
cat <<EOF > /etc/yum.repos.d/neo4j.repo
[neo4j]
name=Neo4j RPM Repository
baseurl=https://yum.neo4j.com/stable
enabled=1
gpgcheck=1
EOF
```



For milestone, alpha, beta, and release candidate releases, replace the [baseurl](#) line with [baseurl=https://yum.neo4j.com/testing](#)

2. Ensure the correct Java version.

Neo4j 4.2 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 4.2.0, 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 manager 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.

- Download and install the Oracle Java 11 JDK from the [Oracle website](#).
- 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 4.2.0.

3. Install Neo4j.

□ To install Neo4j Community Edition as `root`:

```
yum install neo4j-4.2.0
```

□ To install Neo4j Enterprise Edition as `root`:

```
yum install neo4j-enterprise-4.2.0
```



If you are installing a milestone, alpha, beta, or release candidate release, the name of the package is `neo4j-enterprise-<version>-0.<release>.1`. For example, Neo4j Enterprise Edition Milestone Release 3 would be: `neo4j-enterprise-4.0.0-0.beta03mr03.1`

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

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



Neo4j supports Security-Enhanced Linux (SELinux), by default.

Non-interactive installation of Neo4j Enterprise Edition

When installing Neo4j Enterprise Edition, you will be required to accept the license agreement before installation is allowed to complete. This is 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 by setting the environment variable `NEO4J_ACCEPT_LICENSE AGREEMENT` to `yes`:

```
NEO4J_ACCEPT_LICENSE AGREEMENT=yes yum install neo4j-enterprise-4.2.0
```

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-4.2.0
```

- To install Neo4j Enterprise Edition as `root`:

```
zypper install neo4j-enterprise-4.2.0
```

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 which has the relevant access, and then move the RPM package manually.



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 Download Center](#).

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

- Neo4j Enterprise Edition:

```
curl -O https://dist.neo4j.org/rpm/neo4j-enterprise-4.2.0-1.noarch.rpm
```

- Neo4j Community Edition:

```
curl -O https://dist.neo4j.org/rpm/neo4j-4.2.0-1.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, followed by Neo4j:

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

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.
- 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.

2.3.3. Linux tarball installation

This section describes how to install Neo4j on Linux from a tarball, and run it as a console application or service.

Unix console application

1. Download the latest release from [Neo4j Download Center](#).

Select the appropriate tar.gz distribution for your platform.

2. Check that the SHA hash of the downloaded file is correct:

- a. To find the correct SHA hash, go to Neo4j Download 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.
 3. Extract the contents of the archive, using `tar -xf <filename>`

Refer to the top-level extracted directory as: NEO4J_HOME

4. Change directory to: \$NEO4J_HOME

Run `./bin/neo4j console`

5. Stop the server by typing `Ctrl-C` in the console.

Linux service

If you want to run Neo4j as a system service, you can install either the [Debian](#) or [RPM](#) package.

For more information on configuring and 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.

2.3.4. Neo4j system service

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

 *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.

Starting the service automatically on system start

If you installed the RPM package and want Neo4j to start automatically on system boot then you need to enable the service. On Debian-based distributions this is done for you at installation time.

```
systemctl enable neo4j
```

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 which 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.

2.4. macOS installation

This section describes how to install Neo4j on macOS.

2.4.1. Unix console application

1. Download the latest release from [Neo4j Download Center](#).

Select the appropriate tar.gz distribution for your platform.

2. Check that the SHA hash of the downloaded file is correct:

- a. To find the correct SHA hash, go to Neo4j Download 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.
3. Extract the contents of the archive, using `tar -xf <filename>`

Refer to the top-level extracted directory as: NEO4J_HOME

4. Change directory to: \$NEO4J_HOME

Run `./bin/neo4j console`

5. Stop the server by typing `Ctrl-C` in the console.

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

2.4.2. macOS service

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

2.4.3. 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.

2.5. Windows installation

This section describes how to install Neo4j on Windows.

2.5.1. Windows console application

1. Download the latest release from [Neo4j Download Center](#).

Select the appropriate ZIP distribution.

2. Check that the SHA hash of the downloaded file is correct:

- a. To find the correct SHA hash, go to Neo4j Download 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.
3. Right-click the downloaded file, click Extract All.
 4. Change directory to the top-level extracted directory.

Run `bin\neo4j console`

5. Stop the server by typing `Ctrl-C` in the console.

2.5.2. 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`.

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

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
```

2.5.3. 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.ps1
```

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	4.2.0	Neo4j-Management
Function	Invoke-Neo4jAdmin	4.2.0	Neo4j-Management
Function	Invoke-Neo4jBackup	4.2.0	Neo4j-Management
Function	Invoke-Neo4jImport	4.2.0	Neo4j-Management
Function	Invoke-Neo4jShell	4.2.0	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 3. Cloud deployments

This chapter describes the different options for deploying Neo4j in the cloud.

The topics covered are:

- [Neo4j cloud VMs](#) — Deploying Neo4j on cloud virtual machines.
- [Neo4j on Amazon EC2](#) — Deploying Neo4j on Amazon EC2.
- [Neo4j on Google Cloud Platform](#) — Deploying Neo4j on Google Cloud Platform (GCP).
- [Neo4j on Microsoft Azure](#) — Deploying Neo4j on Microsoft Azure.

Other cloud deployment options



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

Neo4j can be run in a **Docker** container. For information on running Neo4j on Docker, see [Docker](#).

3.1. Neo4j cloud VMs

This chapter describes how Neo4j deployed on cloud virtual machines operate and how they differ from other installation platforms for Neo4j.

3.1.1. Basics and file Locations

Neo4j cloud VMs are based on the Ubuntu distribution of Linux. When Neo4j is installed on a VM, the method used to do this matches the Debian install instructions provided in the [Debian](#). Because cloud images are based on the standard Neo4j Debian package, file locations match the file locations described in the [File locations](#), where `neo4j-home` is set to `/var/lib/neo4j`. The remainder of this page deals only with topics that are different from a standard Linux install. If you have any other questions not covered by this page, consult [Linux installation](#).

3.1.2. VM configuration



For the cloud version of Neo4j, you must not modify the `/etc/neo4j/neo4j.conf` file directly, but rather modify `/etc/neo4j/neo4j.template`.

The system service that restarts Neo4j calls a shell script called `pre-neo4j.sh`.

In cloud environments, much of the external configuration environment may change. A machine may have a different IP address or a different set of tags when it restarts. Because of this dynamic nature, the `pre-neo4j.sh` script dynamically overwrites the normal `neo4j.conf` file each time the system service starts. As a result, you must configure the template to do those substitutions and not the configuration file itself, as it will be automatically overwritten.

3.1.3. Configuration via VM tags

On cloud platforms, you may set general `neo4j.conf` configuration parameters as tags on the VM, which will be picked up and substituted into the configuration file. In this way, for example, you might set a tag on a VM of `dbms_backup_enabled` with the value `false` to disable the backup port. When

changing VM tags, the configuration is not immediately applied to the Neo4j system service running inside of the VM. To affect these changes, please [restart the system service](#).



Naming conventions for tags follow the same conventions as docker containers. Dots in a configuration parameter's name must be replaced by underscore characters.

3.1.4. Interacting with the Neo4j Service

You can get system status for `neo4j` within the VM by executing the following:

```
systemctl status neo4j
```

3.2. Neo4j on Amazon EC2

This chapter describes the different options for deploying and running Neo4j on AWS EC2.

There are several options for running Neo4j on AWS EC2, depending on what you want to do.

- [Neo4j Enterprise Causal Clusters in AWS Marketplace](#) – Launching a multi-VM clustered configuration from AWS Marketplace, with the choice to configure many aspects of the cluster, including the number of core nodes, read replicas, hardware sizing, encrypted EBS volumes, and other options.
- [Hosting Neo4j on AWS EC2 AMI](#) – Launching Neo4j using the Amazon's command-line tool.
- [Community Edition in AWS Marketplace](#) – Installing Neo4j Community from the AWS marketplace.
- [Neo4j deployment automation on AWS](#) – Automating Neo4j deployments on AWS.

3.2.1. Neo4j deployment automation on AWS

This chapter describes how to automate Neo4j deployment on AWS.

Automate Neo4j deployment when you want to integrate Neo4j into your CI/CD pipeline to be able to create/destroy instances temporarily, or to spin up a sample instance.

Prerequisites

- You have installed the [AWS command-line interface](#).
- You have generated an access token.
- You have defined the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.
- You have installed jq tool for working with JSON responses. See the [Download jq page](#).

CloudFormation

Neo4j provides CloudFormation templates for Neo4j Enterprise standalone, Neo4j Causal Cluster (highly-available clusters), and Neo4j Community.

CloudFormation is a recipe that tells AWS how to deploy a whole set of interrelated resources.

The Neo4j CloudFormation templates have the following properties:

- Deploying one or more EC2 VMs in a specified region.
- Deploying EC2 VMs in multiple availability zones within a region, so that if one goes down, your entire database does not go down.
- Deploying a new virtual private cloud (VPC) and installing Neo4j in it. In this way, you can control network access by tuning your VPC and security rules.

Creating a CloudFormation stack

Depending on what Neo4j edition you want to deploy, you create a CloudFormation stack by running a bash script. Each script contains the following configurations:

- The URL of the Neo4j stack template that tells AWS what to deploy.
- Various parameters that control how much hardware you want to use.
- **SSHKEY** – the name of your SSH key on AWS to be used to SSH into the instances as the user “ubuntu”.
- **NetworkWhitelist** - it is set to **0.0.0.0/0** by default, which means that any IP on the internet can contact your instance. If you want to lock it down to just your company’s IP block, this is where you must specify that.
- **INSTANCE** - the AWS instance type you want to launch, which controls your database capacity.
- **REGION** - specifies where to deploy Neo4j. Possible values are: **us-east-1, us-east-2, us-west-1, us-west-2, eu-west-1, eu-central-1, ap-southeast-1, ap-northeast-1, ap-south-1, and sa-east-1**.

Deploying Neo4j Enterprise Standalone

To deploy Neo4j Enterprise Standalone, use the Single instance template. It does not have high-availability failover capabilities, but it is a very fast way to get started.

```
#!/bin/bash
VERSION=4.2.0
export SINGLE_TEMPLATE=http://neo4j-cloudformation.s3.amazonaws.com/neo4j-enterprise-standalone-stack-$VERSION.json
export STACKNAME=neo4j-enterprise-$($echo $VERSION | sed s/[A-Za-z0-9]/-/g)
export INSTANCE=r4.large
export REGION=us-east-1
export SSHKEY=my-ssh-keyname
aws cloudformation create-stack \
--stack-name $STACKNAME \
--region $REGION \
--template-url $SINGLE_TEMPLATE \
--parameters ParameterKey=InstanceType,ParameterValue=$INSTANCE \
ParameterKey=NetworkWhitelist,ParameterValue=0.0.0.0/0 \
ParameterKey=Password,ParameterValue=s00pers3cret \
ParameterKey=SSHKeyName,ParameterValue=$SSHKEY \
ParameterKey=VolumeSizeGB,ParameterValue=37 \
ParameterKey=VolumeType,ParameterValue=gp2 \
--capabilities CAPABILITY_NAMED_IAM
```

Deploying Neo4j Enterprise Causal Cluster

To deploy Neo4j Enterprise Causal Cluster, use the Causal Cluster template.



You indicate how many core servers you want in your cluster by configuring the **ClusterNodes** parameter. Minimum value: **3**.

```

#!/bin/bash
VERSION=4.2.0
export CLUSTER_TEMPLATE=http://neo4j-cloudformation.s3.amazonaws.com/neo4j-enterprise-stack-$VERSION.json
export STACKNAME=neo4j-enterprise-$(echo $VERSION | sed s/[A-Za-z0-9]/-/g)
export INSTANCE=r4.large
export REGION=us-east-1
export SSHKEY=my-ssh-keyname
aws cloudformation create-stack \
--stack-name $STACKNAME \
--region $REGION \
--template-url $CLUSTER_TEMPLATE \
--parameters ParameterKey=ClusterNodes,ParameterValue=3 \
ParameterKey=InstanceType,ParameterValue=$INSTANCE \
ParameterKey=NetworkWhitelist,ParameterValue=0.0.0.0/0 \
ParameterKey=Password,ParameterValue=s00pers3cret \
ParameterKey=SSHKeyName,ParameterValue=$SSHKEY \
ParameterKey=VolumeSizeGB,ParameterValue=37 \
ParameterKey=VolumeType,ParameterValue=gp2 \
--capabilities CAPABILITY_NAMED_IAM

```

Deploying Neo4j Community Standalone

To deploy Neo4j Community Standalone, use the Community template.

```

#!/bin/bash
VERSION=4.2.0
export COMMUNITY_TEMPLATE=http://neo4j-cloudformation.s3.amazonaws.com/neo4j-community-standalone-stack-$VERSION.json
export STACKNAME=neo4j-comm-$(echo $VERSION | sed s/[A-Za-z0-9]/-/g)
export INSTANCE=r4.large
export REGION=us-east-1
export SSHKEY=my-ssh-keyname
aws cloudformation create-stack \
--stack-name $STACKNAME \
--region $REGION \
--template-url $COMMUNITY_TEMPLATE \
--parameters ParameterKey=InstanceType,ParameterValue=$INSTANCE \
ParameterKey=NetworkWhitelist,ParameterValue=0.0.0.0/0 \
ParameterKey=Password,ParameterValue=s00pers3cret \
ParameterKey=SSHKeyName,ParameterValue=$SSHKEY \
ParameterKey=VolumeSizeGB,ParameterValue=37 \
ParameterKey=VolumeType,ParameterValue=gp2 \
--capabilities CAPABILITY_NAMED_IAM

```

Checking to see if your instance is up

In each case, the commands submit a CloudFormation stack to be deployed, but they do not wait for the stack to be available. If you want to wait for the CloudFormation stack to finish deploying, use the following command:

```
aws cloudformation wait stack-create-complete --region $REGION --stack-name "$STACKNAME"
```

Finally, you can get the stack outputs, like this:

```
aws cloudformation describe-stacks --region $REGION --stack-name "$STACKNAME"
```

In general, this outputs a lot JSON content. To cut straight to the outputs of the stack, use the `jq` tool.

```
jq -r '.Stacks[0].Outputs[]'
```

The result is a set of outputs with the IP address and password of your new instance. By the time the CloudFormation template finishes deploying, the service will be live and ready to go.

Cleaning up and removing your stack

When you are done with your CloudFormation stack, you can delete it by using the following script:

```
#!/bin/bash
echo "Deleting stack $1"
aws cloudformation delete-stack --stack-name "$1" --region us-east-1
```

3.3. Neo4j on Google Cloud Platform

This chapter describes the different options for deploying and running Neo4j on Google Cloud Platform.

There are several options for running Neo4j on GCP, depending on what you want to do.

- [Test Drive](#) — A free Neo4j test drive on GCP.
- [Single instances \(VM-based\)](#) — Launching a single instance from an image.
- [Causal Clusters \(VM-Based\)](#) — Deploying Neo4j on GCP.
- [Neo4j deployments automation on GCP](#) – Automating Neo4j deployments on GCP.

3.3.1. Test drive

This chapter provides a link to a free Neo4j test drive on GCP.

The [GCP test drive](#) provides guided walk-throughs of use cases for graph databases. Once you launch your test drive, you will be provided with a user guide, which will walk you through a series of labs to help you tour the product, learn key concepts, use cases, and features.

3.3.2. Single instances (VM-based)

This chapter describes how to launch a single instance from an image on GCP.

Prerequisites

- You know how to run and operate Neo4j locally.
- You know how to access cloud-hosted Neo4j from your application. See the [Driver Manual](#).
- You have [installed and set up Google Cloud SDK](#) to be able to use the `gcloud` command-line tool.
- You have [authenticated your gcloud CLI](#), to interact with your GCP projects.

Create a firewall rule to access your instance

Create a firewall rule to be able to access your instance when it is launched:

```
gcloud compute firewall-rules create allow-neo4j-bolt-https \
--allow tcp:7473,tcp:7687 \
--source-ranges 0.0.0.0/0 \
--target-tags neo4j
```

It allows traffic on port [7473](#) (HTTPS for Neo4j Browser) and [7687](#) (Bolt protocol for clients to work with

the database). The `--source-ranges` provided allows the entire Internet to contact your new instance. The `--target-tags` specifies that this rule applies only to VMs tagged `neo4j`. When you launch your instance, you will have to apply that tag to it.

Create a Google compute instance from the Neo4j public image

You launch the instance by using the following command:

```
gcloud config set project my-project-id
gcloud compute instances create my-neo4j-instance \
--image neo4j-enterprise-1-3-4-9-apoc \
--tags neo4j \
--image-project launcher-public
```

The first line sets your project configuration to ensure you know where you are launching your instance. The second line launches an image found in the provided public project. The image name `neo4j-enterprise-1-3-4-9-apoc` corresponds to an Ubuntu-based image that contains Neo4j 1:3.3.9, with the APOC plugin.



The `gcloud` tool comes with many command-line options. For more details on how to deal with machine type, memory, available storage, etc., consult [the Google Cloud documentation](#).

The `--tags` argument allows you to configure the correct network permissions. By default, Google blocks all external access to the network services unless you open them.

When the launch is successful, you get the following result:

```
Created [https://www.googleapis.com/compute/v1/projects/testbed-187316/zones/us-east1-b/instances/my-neo4j-instance].
NAME          ZONE        MACHINE_TYPE    PREEMPTIBLE   INTERNAL_IP  EXTERNAL_IP    STATUS
my-neo4j-instance  us-east1-b  n1-standard-1           10.142.0.3  35.231.125.253  RUNNING
```

Access your new instance

Navigate to `https://[External_IP]:7473` and log in with the username `neo4j` and password `neo4j`. You will be prompted to change the password immediately.

Because you do not have a hostname or a valid SSL certificate configured by default, your browser will warn you that the certificate is not trusted. You can configure the certificate later.

Access your instance via SSH

You can run the following command to SSH into the instance:

```
gcloud compute ssh my-neo4j-instance
```

Inside the VM, you can check the status of the `neo4j` service:

```
$ sudo systemctl status neo4j
● neo4j.service - Neo4j Graph Database
   Loaded: loaded (/etc/systemd/system/neo4j.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2018-03-14 11:19:56 UTC; 15min ago
     Main PID: 1290 (pre-neo4j.sh)
        Tasks: 46
       Memory: 325.7M
          CPU: 20.690s
        CGroup: /system.slice/neo4j.service
                └─1290 /bin/bash /etc/neo4j/pre-neo4j.sh
                  ├─1430 /usr/bin/java -cp /var/lib/neo4j/plugins:/etc/neo4j:/usr/share/neo4j/lib/*
                  *─*: /var/lib/neo4j/plugins/* -server -XX:+UseG1GC
```

For details on internals of Google VMs, including how to stop and start system services, configure Neo4j from the VM, etc., consult [Neo4j cloud VMs](#).

Deleting the instance

You can run the following command to delete your instance:

```
gcloud compute instances delete my-neo4j-instance
```

3.3.3. Causal Clusters (VM-based)

This chapter describes how to deploy and run Neo4j Causal Cluster from the GCP Marketplace.

Neo4j Enterprise is registered in GCP Marketplace.

Prerequisites

- You have a Neo4j Enterprise license.
- You are familiar with the [Causal Cluster architecture](#).
- You know how to access cloud-hosted Neo4j from your application. See the [Driver Manual](#).

Deploy Neo4j via the GCP Marketplace

Deploy Neo4j Enterprise from the [Google Cloud Launcher console](#) following the interactive prompts.

Once the deploy finishes, save the URL, username, and password.

Start using Neo4j Browser

Use your browser to access the cloud-based database URL, and log in with the initial username and password provided. You may see an SSL warning screen because the out-of-the-box deployment uses an unsigned SSL certificate. The initial password is set to a strong, random password and is saved as a metadata entry on the VMs.

To verify that the cluster has formed correctly, run the following Cypher statement:

```
CALL dbms.cluster.overview()
```

The result is one leader and minimum two followers. The IP addresses and endpoints must be the same as the ones for your running instances, displayed by the Compute Engine.

Access your instance via SSH

Cluster members are regular Google Compute Engine VMs. Therefore, you can access any of them via SSH from the Deployment Manager screen, or by running the following command in the Google Cloud CLI:

```
gcloud compute ssh my-cluster-deploy-vm-1
```

For details on internals of Google VMs, including how to stop and start system services, configure Neo4j from the VM, etc., consult [Neo4j cloud VMs](#).

Your cluster default configuration

The following notes are provided on your default cluster configuration.

- Ports [7687](#) (bolt) and [7473](#) (HTTPS access) are the only ports exposed to the entire internet. Consider narrowing the access to these ports to only your needed networks. External unencrypted HTTP access is disabled by default.
- Ports [5000](#), [6000](#), and [7000](#) are enabled only for internal network access ([10.0.0.8](#)), between the cluster nodes.
- Because cloud VMs can start and stop with different IP addresses, the configuration of these VMs is driven by a file in `/etc/neo4j/neo4j.template`. Configuration changes must be made to the template, not to the `/etc/neo4j/neo4j.conf` file, which is overwritten with the template substitutions at every startup. The template allows you to configure aspects of the cluster with the VMs metadata. The template's behavior and layout match the usual `neo4j.conf` file.

What's next

- Visit [Clustering](#) for more information on how to configure your cluster.
- Add users and change passwords as necessary.
- Consider creating DNS entries with Google to be able to address your cluster with client applications under a single hostname.

Terminating the deployment

You can use the deployment manager to delete the deployment. To ensure data safety, the disks that back the VMs are not removed when you delete the cluster deployment.

3.3.4. Neo4j deployments automation on Google Cloud Platform (GCP)

This chapter describes how to automate Neo4j deployments on GCP.

Automate Neo4j deployment when you want to integrate Neo4j into your CI/CD pipeline to be able to create/destroy instances temporarily, or to spin up a sample instance.

Prerequisites

- You have [installed and set up Google Cloud SDK](#) to be able to use the `gcloud` command-line tool.
- You have [authenticated your gcloud CLI](#), to make sure it can interact with your GCP projects.

Google Cloud Deployment Manager

Neo4j provides Deployment Manager templates for Neo4j Causal Cluster (highly available clusters), and VM images for Neo4j Enterprise standalone. Deployment Manager is a recipe that tells GCP how to deploy a whole set of interrelated resources. By deploying all of this as a stack you can keep all of your resources together, and delete just one thing when you are done.

Creating a Deployment Manager stack

Depending on what Neo4j edition you want to deploy, you create a Deployment Manager stack by running a bash script.

Each script contains the following configurations:

- The URL of the Neo4j stack template that tells GCP what to deploy.
- Various parameters that control how much hardware you want to use.
- **MACHINE** - the GCP machine type you want to launch, which controls how much hardware you will be giving to your database.
- **DISK_TYPE** and **DISK_SIZE**- controls whether Neo4j uses standard spinning magnetic platters (pd-standard) or SSD disks (pd-ssd), and how many GB of storage you want to allocate. Note that with some disk sizes, GCP warns that the root partition type may need to be resized if the underlying OS does not support the disk size. This warning can be ignored, because the underlying OS will recognize any disk size.
- **ZONE** - specifies where to deploy Neo4j.
- **PROJECT** - the project ID you want to deploy on GCP.

Deploying Neo4j Enterprise Edition with a Causal Cluster

To deploy Neo4j Enterprise Edition with a Causal Cluster, use the Causal Cluster template.



You indicate how many core servers and read replicas you want in your cluster by configuring the **CORES** and **READ_REPLICAS** parameters.

```
#!/bin/bash
export NAME=neo4j-cluster
PROJECT=my-gcp-project-ID
MACHINE=n1-standard-2
DISK_TYPE=pd-ssd
DISK_SIZE=64
ZONE=us-east1-b
CORES=3
READ_REPLICAS=0
NEO4J_VERSION=3.5.3
TEMPLATE_URL=https://storage.googleapis.com/neo4j-deploy/$NEO4J_VERSION/causal-cluster/neo4j-causal-cluster.jinja
OUTPUT=$(gcloud deployment-manager deployments create $NAME \
    --project $PROJECT \
    --template "$TEMPLATE_URL" \
    --properties "zone:'$ZONE',clusterNodes:'$CORES',readReplicas:'$READ_REPLICAS',bootDiskSizeGb:$DISK_SIZE,bootDiskType:'$DISK_TYPE',machineType:'$MACHINE'")
echo $OUTPUT
PASSWORD=$(echo $OUTPUT | perl -ne 'm/password\s+([^\s]+); print $1;')
IP=$(echo $OUTPUT | perl -ne 'm/vm1URL\s+https:\//([^\s]+); print $1; ')
echo NEO4J_URI=$IP
echo NEO4J_PASSWORD=$PASSWORD
echo STACK_NAME=$NAME
```

After you configure the parameters of what you are deploying, you call to **gcloud deployment-manager deployments create** to do the work. The variable **OUTPUT** contains all the information about your deployment. Then, you use **perl** to pull out the password and IP address of your new deployment,

because it will have a strong randomly assigned password.



This command blocks and does not succeed until the entire stack is deployed and ready. This means that by the time you get the IP address back, your Neo4j is up. If you lose these stack outputs (IP, password, and so on), you can find them in your Deployment Manager window within the GCP console.

To delete your deployment, take note of the `STACK_NAME` and use the utility script:

```
#!/bin/bash
PROJECT=my-google-project-id
if [ -z $1 ] ; then
  echo "Usage: call me with deployment name"
  exit 1
fi
gcloud -q deployment-manager deployments delete $1 --project $PROJECT
# OPTIONAL! Destroy the disk
# gcloud --quiet compute disks delete $(gcloud compute disks list --project $PROJECT --filter="name~'$1'" --uri)
```



When you delete Neo4j stacks on GCP, the GCP disks are left behind, to make it hard for you to accidentally destroy your valuable data. To completely clean up your disks, uncomment the last line of the script.

Deploying Neo4j Enterprise (or Community) Edition in standalone mode

To deploy Neo4j Enterprise Edition in standalone mode, create a simple VM and configure its firewall/security rules. It will not have high-availability failover capabilities, but it is a very fast way to get started.

You choose a random password by running some random bytes through a hash. The script also provides an example of polling and waiting until the VM service comes up, and then changing the Neo4j default password.

The `launcher-public` project on GCP hosts Neo4j's VM images for GCP. In the example script, `neo4j-enterprise-1-3-5-3-apoc` is used, but other versions are also available. By substituting a different image name here, you can use this same technique to run Neo4j Community Edition in standalone mode.

```

#!/bin/bash
export PROJECT=my-gcp-project-id
export MACHINE=n1-standard-2
export DISK_TYPE=pd-ssd
export DISK_SIZE=64GB
export ZONE=us-east1-b
export NEO4J_VERSION=3.5.3
export PASSWORD=$(head -n 20 /dev/urandom | md5)
export STACK_NAME=neo4j-standalone
export IMAGE=neo4j-enterprise-1-3-5-3-apoc
# Setup firewalling.
echo "Creating firewall rules"
gcloud compute firewall-rules create "$STACK_NAME" \
    --allow tcp:7473,tcp:7687 \
    --source-ranges 0.0.0.0/0 \
    --target-tags neo4j \
    --project $PROJECT
if [ $? -ne 0 ] ; then
    echo "Firewall creation failed. Bailing out"
    exit 1
fi
echo "Creating instance"
OUTPUT=$(gcloud compute instances create $STACK_NAME \
    --project $PROJECT \
    --image $IMAGE \
    --tags neo4j \
    --machine-type $MACHINE \
    --boot-disk-size $DISK_SIZE \
    --boot-disk-type $DISK_TYPE \
    --image-project launcher-public)
echo $OUTPUT
# Pull out the IP addresses, and toss out the private internal one (10.*)
IP=$(echo $OUTPUT | grep -oE '((1?[0-9][0-9]?|2[0-4][0-9]|25[0-5])\.){3}(1?[0-9][0-9]?|2[0-4][0-9]|25[0-5])' | grep --invert-match "^10\.")
echo "Discovered new machine IP at $IP"
tries=0
while true ; do
    OUTPUT=$(echo "CALL dbms.changePassword('$PASSWORD');" | cypher-shell -a $IP -u neo4j -p "neo4j" 2>&1)
    EC=$?
    echo $OUTPUT

    if [ $EC -eq 0 ]; then
        echo "Machine is up ... $tries tries"
        break
    fi
    if [ $tries -gt 30 ] ; then
        echo STACK_NAME=$STACK_NAME
        echo "Machine is not coming up, giving up"
        exit 1
    fi
    tries=$((tries+1))
    echo "Machine is not up yet ... $tries tries"
    sleep 1;
done
echo NEO4J_URI=bolt://$IP:7687
echo NEO4J_PASSWORD=$PASSWORD
echo STACK_NAME=$STACK_NAME
exit 0

```

To delete your deployment, take note of the `STACK_NAME` and use the utility script:

```

#!/bin/bash
export PROJECT=my-google-project-id
if [ -z $1 ] ; then
    echo "Missing argument"
    exit 1
fi
echo "Deleting instance and firewall rules"
gcloud compute instances delete --quiet "$1" --project "$PROJECT" && gcloud compute firewall-rules --quiet
delete "$1" --project "$PROJECT"
exit $?

```

3.4. Neo4j on Microsoft Azure

This chapter describes the different options for deploying and running Neo4j on Microsoft Azure.

There are several options for running Neo4j on Azure, depending on what you want to do.

- [Single instances \(VM-based\)](#) — Deploying Neo4j single instances on Azure.
- [Causal Clusters \(VM-Based\)](#) — Deploying Neo4j Causal cluster on Azure.
- [Neo4j deployments automation on Azure](#) – Automating Neo4j deployments on Azure.

3.4.1. Single instances (VM-based)

This chapter describes how to launch a single instance from an image on Azure.

Prerequisites

- You know how to run and operate Neo4j locally.
- You have a Neo4j Enterprise or [a trial license for Azure](#).
- You know how to access cloud-hosted Neo4j from your application. See the [Driver Manual](#).
- You have [installed and set up Azure Command Line Interface](#).

Deploy Neo4j via the Azure Marketplace

Deploy Neo4j Enterprise VM from the [Azure Marketplace](#) following the interactive prompts.



The most important setting to consider are [Size](#), which controls the available CPU and memory, and optionally [Disks](#), where you configure high-speed SSDs and larger disk capacity sizes. It is recommended to create a new resource group to hold the artifacts of your deployment.

Once the deploy finishes, save the URL, username, and password.

Access your new instance

Navigate to [https://\[MY_Azure_IP\]:7473](https://[MY_Azure_IP]:7473) and log in with the username `neo4j` and password `neo4j`. You will be prompted to change the password immediately.

Because you do not have a hostname or a valid SSL certificate configured by default, your browser will warn you that the certificate is not trusted. You can configure the certificate later.

Access your instance via SSH

You can use any SSH client as normal to connect to the public IP of your instance. Use the administrative user credentials (password or SSH key) configured during the launch. This user has `sudo` access on the machine.

Inside the VM, you can check the status of the `neo4j` service:

```
$ sudo systemctl status neo4j
neo4j.service - Neo4j Graph Database
   Loaded: loaded (/etc/systemd/system/neo4j.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2018-03-14 11:19:56 UTC; 15min ago
       Main PID: 1290 (pre-neo4j.sh)
          Tasks: 46
         Memory: 325.7M
            CPU: 20.690s
           CGroup: /system.slice/neo4j.service
                   ├─1290 /bin/bash /etc/neo4j/pre-neo4j.sh
                   └─1430 /usr/bin/java -cp /var/lib/neo4j/plugins:/etc/neo4j:/usr/share/neo4j/lib/*
*:~/var/lib/neo4j/plugins/* -server -XX:+UseG1GC
```

For details on internals of Azure VMs, including how to stop and start system services, configure Neo4j from the VM, etc., consult [Neo4j cloud VMs](#).

Deleting the instance

You can remove the infrastructure by deleting the entire resource group you created as part of the deployment. If you deployed into an existing resource group, you have to individually delete the resources that are part of the deployment.

3.4.2. Causal Clusters (VM-based)

This chapter describes how to deploy and run Neo4j Causal Cluster on Azure.

Prerequisites

- You have a Neo4j Enterprise or [a trial license for Azure](#).
- You are familiar with the [Causal Cluster architecture](#).
- You know how to access cloud-hosted Neo4j from your application. See the [Driver Manual](#).

Deploy Neo4j from the Azure Marketplace

Deploy Neo4j Enterprise Causal Cluster from the [Azure Marketplace](#) following the interactive prompts. Create a new resource group to hold the artifacts of your deployment, as the admin account name is used for SSH access to the machines in your cluster.

Once the deploy finishes, save the URL, username, and password.



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

Start using Neo4j Browser

Use your browser to access the cloud-based database URL, and log in with the initial username and password provided. You may see an SSL warning screen because the out-of-the-box deployment uses an unsigned SSL certificate.

To verify that the cluster has formed correctly, run the following Cypher statement:

```
CALL dbms.cluster.overview()
```

Access your instance via SSH

You can SSH into any of the machines using the configured hostname and admin credentials.

For details on internals of Azure VMs, including how to stop and start system services, configure Neo4j from the VM, etc., consult [Neo4j cloud VMs](#).

Your cluster default configuration

The following notes are provided on your default cluster configuration.

- Ports **7687** (bolt) and **7473** (HTTPS access) are the only ports exposed to the entire internet. Consider narrowing the access to these ports to only your needed networks. External unencrypted HTTP access is disabled by default.
- Ports **5000**, **6000**, and **7000** are enabled only for internal network access (**10.0.0.8**), between the cluster nodes.

What's next

- Visit [Clustering](#) for more information on how to configure your cluster.
- Add users and change passwords as necessary.
- Consider creating DNS entries with Google to be able to address your cluster with client applications under a single hostname.

Terminating the deployment

You can remove the infrastructure by deleting the entire resource group you created as part of the deployment.

3.4.3. Neo4j deployments automation on Azure

This chapter describes how to automate Neo4j deployments on Azure.

Automate Neo4j deployment when you want to integrate Neo4j into your CI/CD pipeline to be able to create/destroy instances temporarily, or to spin up a sample instance.

Prerequisites

- You have installed the [Azure command-line interface](#).
- You have installed jq tool for working with JSON responses. See the [Download jq page](#).
- You have authenticated your `az` CLI to be able to interact with your resource groups and use the right subscription by default. For more information on how to change the tool subscription, see [the Azure CLI documentation](#).

Azure Resource Manager

Neo4j provides Azure Resource Manager (ARM) templates for Neo4j Enterprise standalone and Neo4j Causal Cluster (highly-available clusters).

ARM templates are a recipe that tells Azure how to deploy a whole set of interrelated resources. By deploying all of this as a stack you can keep all your resources together, and manage the entire instance by managing this resource group.

Creating an ARM deployment job

Depending on what Neo4j edition you want to deploy, you create ARM Deployment job by running a shell script.

Each script contains the following configurations:

- The URL of the Neo4j stack template that tells Azure what to deploy.
- Various parameters that control how much hardware you want to use.
- **VM_SIZE** – the Azure VM type you want to launch, which controls how much hardware you will be using.
- **DISK_SIZE** and **DISK_TYPE** – controls whether Neo4j uses standard spinning magnetic platters (pd-standard) or SSD disks (pd-ssd), and how many GB of storage you want to allocate.
- **LOCATION** - specifies where to deploy Neo4j.
- Authentication details - the administrative username and password for access to the VMs.

Deploying Neo4j Enterprise Causal Cluster

To deploy Neo4j Enterprise Causal Cluster, use the Causal Cluster template.



You indicate how many core servers and read replicas you want in your cluster by configuring the **CORE_NODES** and **READ_REPLICAS** parameters.

Take note of the **TEMPLATE_BASE** parameter, which contains the Neo4j version you want to launch. This can be adjusted to any version of Neo4j where there are published ARM templates. Create a simple JSON file with your deployment configurations and pass it to ARM. Based on your inputs, ARM produces a set of infrastructure as an output.

```

#!/bin/bash
export CORE_NODES=3
export READ_REPLICAS=0
export NEO4J_PASSWORD=s00pers3cR3T:
export ADMIN_AUTH_TYPE=password
export USERNAME=graph-hacker
export ADMIN_PASSWORD=s00pers3cR3T:
export VM_SIZE=Standard_B2ms
export DISK_TYPE=StandardSSD_LRS
export DISK_SIZE=256
export IP_ALLOCATION=Dynamic
export SEED=$(head -c 3 /dev/urandom | base64 | sed 's/[a-zA-Z0-9]/X/g')
export RESOURCE_GROUP="neo4j-RG-${SEED}"
export CLUSTERNAME="neo4j-${SEED}"
export DEPLOYMENT=neo4j-bmdeploy
export LOCATION="East US"
# The ARM template to deploy.
export TEMPLATE_BASE=http://neo4j-arm.s3.amazonaws.com/3.5.16/causal-cluster/
export TEMPLATE_URL=${TEMPLATE_BASE}mainTemplate.json
echo $(cat <>JSON
{
"ClusterName": { "value": "${CLUSTERNAME}" },
"CoreNodes": { "value": ${CORE_NODES} },
"ReadReplicas": { "value": ${READ_REPLICAS} },
"VmSize": { "value": "${VM_SIZE}" },
"DataDiskType": { "value": "${DISK_TYPE}" },
"DataDiskSizeGB": { "value": ${DISK_SIZE} },
"AdminUserName": { "value": "${USERNAME}" },
"AdminAuthType": { "value": "${ADMIN_AUTH_TYPE}" },
"AdminCredential": { "value": "${ADMIN_PASSWORD}" },
"PublicIPAllocationMethod": { "value": "${IP_ALLOCATION}" },
"Neo4jPassword": { "value": "${NEO4J_PASSWORD}" },
"_artifactsLocation": { "value": "${TEMPLATE_BASE}" }
}
JSON
) > "${RESOURCE_GROUP}.json"
echo "Creating resource group named ${RESOURCE_GROUP}"
if ! az group create --name "${RESOURCE_GROUP}" --location "${LOCATION}"; then
    echo STACK_NAME=$RESOURCE_GROUP
    echo "Failed to create necessary resource group ${RESOURCE_GROUP}"
    exit 1
fi
echo "Creating deployment"
az group deployment create \
--template-uri "$TEMPLATE_URL" \
--parameters @./${RESOURCE_GROUP}.json \
--resource-group "${RESOURCE_GROUP}" \
--name "${DEPLOYMENT}"
if [ $? -ne 0 ]; then
    echo STACK_NAME=$RESOURCE_GROUP
    echo "Stack deploy failed"
    exit 1
fi
# JSON Path to server response where the IP address is.
ADDR_FIELD=".[].virtualMachine.network.publicIpAddresses[0].ipAddress"
IP_ADDRESS=$(az vm list-ip-addresses --resource-group "${RESOURCE_GROUP}" | jq -r "$ADDR_FIELD" | head -n 1)
echo STACK_NAME=$RESOURCE_GROUP
echo NEO4J_URI=bolt+routing://$IP_ADDRESS:7687

```

As a result, a new resource group is created with all the assets, and you get a URI of a bolt endpoint you can use. Alternatively, go to <https://<IP address>:7473/> to access Neo4j Browser for your new clustered instance.

Deploying Neo4j Enterprise Standalone

To deploy Neo4j Enterprise Standalone, create a simple VM and configure its firewall/security rules. It will not have high-availability failover capabilities, but it is a very fast way to get started.

Neo4j provides the VM through an Azure marketplace offer. To refer to the right VM image, you need to know the publisher (that's Neo4j), the "offer" (Neo4j version series), and SKU (the particular Neo4j version). Because you are not using ARM for this deployment, the script polls and waits until the VM service comes up, and then changes the Neo4j default password. At the top, you can choose a

different password for the `neo4j` user as for a system administrator. Make sure to customize the `SUBSCRIPTION` variable to make this work.

```
#!/bin/bash
export LOCATION=eastus
export SUBSCRIPTION=My-Subscription-Name
export RG=neo4j-standalone-RG
export NAME=neo4j-standalone
export ADMIN_USERNAME=graph-hacker
export ADMIN_PASSWORD=ch00se:A@PASSw0rd
export NEO4J_PASSWORD=ch00se:A@PASSw0rd
export NETWORK_SECURITY_GROUP=neo4j-nsg
# Options: https://azure.microsoft.com/en-us/pricing/details/virtual-machines/
export VM_SIZE=Standard_D2_v3
# Can change this to static if desired
export ADDRESS_ALLOCATION=dynamic
# Configuration bits of what you're launching
# Publisher:Offer:Sku:Version
export PUBLISHER=neo4j
export OFFER=neo4j-enterprise-3_5
export SKU=neo4j_3_5_5_apoc
export VERSION=latest
export IMAGE=$PUBLISHER:$OFFER:$SKU:$VERSION
echo "Creating resource group named $RG"
az group create --location $LOCATION \
    --name $RG \
    --subscription $SUBSCRIPTION
echo "Creating Network Security Group named $NETWORK_SECURITY_GROUP"
az network nsg create \
    --resource-group $RG \
    --location $LOCATION \
    --name $NETWORK_SECURITY_GROUP
echo "Assigning NSG rules to allow inbound traffic on Neo4j ports..."
prio=1000
for port in 7473 7474 7687; do
    az network nsg rule create \
        --resource-group $RG \
        --nsg-name "$NETWORK_SECURITY_GROUP" \
        --name neo4j-allow-$port \
        --protocol tcp \
        --priority $prio \
        --destination-port-range $port
    prio=$((($prio+1)))
done
echo "Creating Neo4j VM named $NAME"
az vm create --name $NAME \
    --resource-group $RG \
    --image $IMAGE \
    --vnet-name $NAME-vnet \
    --subnet $NAME-subnet \
    --admin-username "$ADMIN_USERNAME" \
    --admin-password "$ADMIN_PASSWORD" \
    --public-ip-address-allocation $ADDRESS_ALLOCATION \
    --size $VM_SIZE
if [ $? -ne 0 ] ; then
    echo "VM creation failed"
    exit 1
fi
echo "Updating NIC to have your NSG"
# Uses default assigned NIC name
az network nic update \
    --resource-group "$RG" \
    --name "${NAME}VMNic" \
    --network-security-group "$NETWORK_SECURITY_GROUP"
# Get the IP address of our instance
IP_ADDRESS=$(az vm list-ip-addresses -g "$RG" -n "$NAME" | jq -r
'.[0].virtualMachine.network.publicIpAddresses[0].ipAddress')
export NEO4J_URI=bolt:///$IP_ADDRESS
# Change password
echo "Checking if Neo4j is up and changing password...."
while true; do
    if curl -s -I http://$IP_ADDRESS:7474 | grep "200 OK"; then
        echo "Neo4j is up; changing default password" 2>&1
        curl -v -H "Content-Type: application/json" \
            -XPOST -d '{"password":"'${NEO4J_PASSWORD}'"}' \
            -u neo4j:neo4j \
            http://$IP_ADDRESS:7474/user/neo4j/password 2>&1
        echo "Password reset, signaling success" 2>&1
    fi
    sleep 10
done
```

```
break
fi
echo "Waiting for neo4j to come up" 2>&1
sleep 1
done
echo NEO4J_URI=$NEO4J_URI
exit 0
```

Cleaning up and removing your deployment

When you are done with your deployment, you can delete the entire resource group by using the following script:

```
#!/bin/bash
if [ -z $1 ] ; then
    echo "Usage: call me with deployment name"
    exit 1
fi
STACK_NAME=$1
if [ -f "$STACK_NAME.json" ] ; then
    rm -f "$STACK_NAME.json"
fi
az group delete -n "$STACK_NAME" --no-wait --yes
exit $?
```

Chapter 4. Docker

This chapter describes how run Neo4j in a Docker container.

This chapter describes the following:

- [Introduction](#) — Introduction to running Neo4j in a Docker container.
- [Configuration](#) — How to configure Neo4j to run in a Docker container.
- [Clustering](#) — How to set up Causal Clustering when using Docker.
- [Docker specific operations](#) - Descriptions of various 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.



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

4.1. Introduction

An introduction to how Neo4j runs in a Docker container.

Docker can be downloaded for MacOS, Windows and Linux operating systems here:

<https://www.docker.com/get-started>. There is an official Neo4j image on DockerHub that provides a standard, ready-to-run package of Neo4j. From the DockerHub repo, it is possible to run Community Edition or Enterprise Edition with a variety of versions of the database. The Neo4j Docker image, and instructions on how to start using it, can be found here: https://hub.docker.com/_/neo4j/.

4.1.1. Neo4j editions

Tags are available for both Community Edition and Enterprise Edition. Version-specific Enterprise Edition tags have an `-enterprise` suffix, for example: `neo4j:4.2.0-enterprise`. Community Edition tags have no suffix, for example `neo4j:4.2.0`. The latest Neo4j Enterprise Edition release is available as `neo4j:enterprise`.

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

Neo4j Enterprise Edition license

In order to use Neo4j Enterprise Edition you must accept the license agreement.

© Network Engine for Objects in Lund AB. 2018. 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/>

To accept the license agreement set the environment variable `NEO4J_ACCEPT_LICENSE AGREEMENT=yes`.

To do this you can use the following docker argument:

```
--env NEO4J_ACCEPT_LICENSE AGREEMENT=yes
```

4.1.2. Using the Neo4j Docker image

A Neo4j container can be started using the following command:

```
docker run \
--restart always \
--publish=7474:7474 --publish=7687:7687 \
--volume=$HOME/neo4j/data:/data \
neo4j:{neo4j-version}
```

However, there are several options with the `docker run` command. This table lists some of the options available:

Table 10. Options for docker run

Option	Description	Example
<code>--name</code>	Name your container to avoid generic ID	<code>docker run --name myneo4j neo4j</code>
<code>-p</code>	Specify which container port to expose	<code>docker run -p7687:7687 neo4j</code>
<code>-d</code>	Detach container to run in background	<code>docker run -d neo4j</code>
<code>-v</code>	Bind mount a volume	<code>docker run -v \$HOME/neo4j/data:/data neo4j</code>
<code>--env</code>	Set config as environment variables for Neo4j database	<code>docker run --env NEO4J_AUTH=neo4j/test</code>
<code>--help</code>	Output full list of <code>docker run</code> options	<code>docker run --help</code>

4.1.3. Using `NEO4J_AUTH` to set an initial password

By default, Neo4j requires authentication and prompts you to login with a username/password of `neo4j/neo4j` at the first connection. You are then prompted to set a new password. For more information about setting the initial password for Neo4j, see [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:

```
--env NEO4J_AUTH=neo4j/your_password
```

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`.

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 to log in you will need a username and password already associated with the database.

4.1.4. 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 \
  --volume=$HOME/neo4j/data:/data \
  --volume=$HOME/neo4j/logs:/logs \
  --user="$(id -u):$(id -g)" \
  neo4j:4.2
```



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

4.2. Configuration

This chapter describes how configure Neo4j to run in a Docker container.

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.

4.2.1. 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:4.2
```

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"
```

4.2.2. 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:4.2
```

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:4.2 dump-config
```

4.2.3. Building a new image with Neo4j as a base image

To use a Neo4j Docker image as the base image for a custom image, use the `FROM` instruction in the Dockerfile as such:

```
FROM neo4j:4.2
```

It is recommended to specify an explicit version. See https://hub.docker.com/_/neo4j for available

Neo4j Docker images.

For more information, see [the official Dockerfile instructions](#).

The Neo4j Dockerfile (the base image) specifies `ENTRYPOINT` that checks if the environment variable `EXTENSION_SCRIPT` is set, runs the script that `EXTENSION_SCRIPT` is pointing at and then runs any other commands.

The following is an example of how to create a custom Dockerfile based on the Neo4j image, build the image, and run a container based on it. The current working directory is `/example`:

```
/example/Dockerfile

FROM neo4j:4.2.0-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:4.2.0-enterprise-custom-container-1 /example
```

Create and run a container based on your 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:4.2.0-enterprise-custom-
container-1
```

For more information on Docker's command-line commands, see <https://docs.docker.com/engine/reference/commandline/docker/>.

4.3. Clustering

How to deploy a Causal Cluster setup in a containerized environment without an orchestration tool.

4.3.1. 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. Prepare your `docker-compose.yml` file using the following example. For more information, see the [Docker Compose Service configuration reference](#).

Example 2. Example docker-compose.yml file

```
version: '3.8'

networks:
  lan:

services:

core1:
  image: neo4j:{neo4j-version}-enterprise
  networks:
    - lan
  ports:
    - 7474
    - 7687
  environment:
    - NEO4J_ACCEPT_LICENSE AGREEMENT=yes
    - NEO4J_dbms_memory_pagecache_size=10M \
    - NEO4J_dbms_memory_heap_initial_size=10M \
    - NEO4J_AUTH=neo4j/your_password \
    - NEO4J_dbms_mode=CORE
    - NEO4J_causalClustering_discoveryAdvertisedAddress=core1-public-address:5000 \
    - NEO4J_causalClustering_transactionAdvertisedAddress=core1-public-address:6000 \
    - NEO4J_causalClustering_raftAdvertisedAddress=core1-public-address:7000 \
    - NEO4J_causalClustering_expectedCoreClusterSize=3
    - NEO4J_causalClustering_initialDiscoveryMembers=core1-public-address:5000,core2-public-
address:5000,core3-public-address:5000 \
    - NEO4J_causal__clustering_disable__middleware__logging=false

core2:
  image: neo4j:{neo4j-version}-enterprise
  networks:
    - lan
  ports:
    - 7474
    - 7687
  environment:
    - NEO4J_ACCEPT_LICENSE AGREEMENT=yes
    - NEO4J_dbms_memory_pagecache_size=10M
    - NEO4J_dbms_memory_heap_initial_size=10M
    - NEO4J_AUTH=neo4j/your_password
    - NEO4J_dbms_mode=CORE
    - NEO4J_causalClustering_discoveryAdvertisedAddress=core2-public-address:5000
    - NEO4J_causalClustering_transactionAdvertisedAddress=core2-public-address:6000
    - NEO4J_causalClustering_raftAdvertisedAddress=core2-public-address:7000
    - NEO4J_causalClustering_expectedCoreClusterSize=3
    - NEO4J_causalClustering_initialDiscoveryMembers=core1-public-address:5000,core2-public-
address:5000,core3-public-address:5000
    - NEO4J_causalClustering_refuseToBeLeader=true

core3:
  image: neo4j:{neo4j-version}-enterprise
  networks:
    - lan
  ports:
    - 7474
    - 7687
  environment:
    - NEO4J_ACCEPT_LICENSE AGREEMENT=yes
    - NEO4J_dbms_memory_pagecache_size=10M
    - NEO4J_dbms_memory_heap_initial_size=10M
    - NEO4J_AUTH=neo4j/your_password
    - NEO4J_dbms_mode=CORE
    - NEO4J_causalClustering_discoveryAdvertisedAddress=core3-public-address:5000
    - NEO4J_causalClustering_transactionAdvertisedAddress=core3-public-address:6000
    - NEO4J_causalClustering_raftAdvertisedAddress=core3-public-address:7000
    - NEO4J_causalClustering_expectedCoreClusterSize=3
    - NEO4J_causalClustering_initialDiscoveryMembers=core1-public-address:5000,core2-public-
address:5000,core3-public-address:5000
    - NEO4J_causalClustering_refuseToBeLeader=true
    - NEO4J_dbms_backup_enabled=true
    - NEO4J_dbms_backup_address=0.0.0.0:6362

readreplica1:
  image: neo4j:{neo4j-version}-enterprise
  networks:
    - lan
```

```

ports:
  - 7474
  - 7687
environment:
  - NEO4J_ACCEPT_LICENSE AGREEMENT=yes
  - NEO4J_dbms_memory_pagecache_size=10M
  - NEO4J_dbms_memory_heap_initial_size=10M
  - NEO4J_AUTH=neo4j/your_password
  - NEO4J_dbms_mode=READ_REPLICA
  - NEO4J_causalClustering_discoveryAdvertisedAddress=readreplica1-public-address:5000
  - NEO4J_causalClustering_transactionAdvertisedAddress=readreplica1-public-address:6000
  - NEO4J_causalClustering_raftAdvertisedAddress=readreplica1-public-address:7000
  - NEO4J_causalClustering_initialDiscoveryMembers=core1-public-address:5000,core2-public-address:5000,core3-public-address:5000

```

Custom top-level network.

For more information on how and why to use custom networks, see [Docker official documentation](#).

Service-level network, which specifies the networks, from the list of the top-level networks (in this case only `lan`), that the server will connect to.

The ports that will be accessible from outside the container - HTTP (7474) and Bolt (7687).

For more information on the Neo4j ports, see [Ports](#).

Setting that specifies how much memory Neo4j is allowed to use for the page cache.

Setting that specifies the initial JVM heap size.

For further information, [Memory configuration](#).

Initial password for the container.

For more information on Neo4j authentication, see [Using NEO4J_AUTH to set an initial password](#) and [Running Neo4j as a non-root user](#).

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.

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.

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.

The values of `initial_discovery_members` match the advertised addresses and ports of the `NEO4J_causalClustering_discoveryAdvertisedAddress` setting.

2. Deploy your Causal Cluster by running `docker-compose up` from your project folder.

3. Open `core1` at <http://core1-public-address:7474>.

4. Authenticate with the default `neo4j/your_password` credentials.

5. Check the status of the cluster by running the following in Neo4j Browser:

```
:sysinfo
```

4.3.2. 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

`NE04J_causal__clustering_expected__core__cluster__size` and
`NE04J_causal__clustering_initial__discovery__members` environment variables must be set for Cores.
Read Replicas only need to define `NE04J_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:

- `NE04J_dbms_mode`: the database mode, defaults to `SINGLE`, set to `CORE` or `READ_REPLICA` for Causal Clustering.
- `NE04J_causal__clustering_expected__core__cluster__size`: the initial cluster size (number of Core instances) at startup.
- `NE04J_causal__clustering_initial__discovery__members`: the network addresses of an initial set of Core cluster members.
- `NE04J_causal__clustering_discovery__advertised__address`: hostname/IP address and port to advertise for member discovery management communication.
- `NE04J_causal__clustering_transaction__advertised__address`: hostname/IP address and port to advertise for transaction handling.
- `NE04J_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:4.2-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:4.2-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:4.2-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:4.2-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:4.2-enterprise
```

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



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.

4.4. Docker specific operations

How to use Neo4j tools when running Neo4j in a Docker container.

4.4.1. 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](#).

4.4.2. 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 file 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:{neo4j-version} \
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
```

4.4.3. 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 following example shows how `neo4j-admin memrec --docker` provides a memory recommendation in a docker-friendly format.

Example 3. Invoke `neo4j-admin memrec --docker`

```
$neo4j-home> bin/neo4j-admin memrec --memory=16g --docker
...
...
#
# Based on the above, the following memory settings are recommended:
EXPORT NEO4j_dbms_memory_heap_initial_size=5g
EXPORT NEO4j_dbms_memory_heap_max_size=5g
EXPORT NEO4j_dbms_memory_pagecache_size=7g
```

4.4.4. 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.

- 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:{neo4j-version}
```

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 neo4j
```

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

Because `cypher-shell` does not support file streams, such as `cypher-shell -a localhost -u <username> -p <password> --file myscript.cypher`, you can use the commands `cat` or `curl` to pipe the contents of your script file into your container. The following are syntax examples of how to use these commands:

Invoke curl with Cypher Shell

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

Invoke cat with Cypher Shell

```
# Prepare the example.cypher script, for instance, containing the query:
match (n:Actors)-[r]->(m:Movies) return n.name AS Actors, m.title AS Movies, m.year AS MovieYear;
# Run the following command:
cat example.cypher | sudo docker exec --interactive <containerID/name> cypher-shell -u neo4j -p neo4j
# The command runs the script and returns the following result:
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. Then, they run `cypher-shell`, authenticate with the provided `<username>` and `<password>`, and execute a cypher example, LOAD CSV dataset, which might be hosted somewhere on a server (with `curl`), create indexes, constraints, or do other administrative operations.

4.4.5. 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:4.2
```

4.4.6. Configure Neo4j Labs plugins

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



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 both the APOC and GraphQL plugins, you can use the Docker argument `--env NEO4JLABS_PLUGINS='["apoc", "graphql"]'` 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", "graphql"]' \
neo4j:{neo4j-version}
```

If a single plugin is required, it must still be specified as a JSON-formatted list. For example, to install the Neo Semantics `n10s` plugin, you can use the following Docker argument:

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

Table 11. Supported Neo4j Labs plugins

Name	Key	Further information
APOC	apoc	https://neo4j.com/labs/apoc/
Graph Data Science	graph-data-science	https://neo4j.com/docs/graph-data-science/current/
GraphQL	graphql	https://neo4j.com/labs/grandstack-graphql/
Neo Semantics	n10s	https://neo4j.com/labs/nsmtx-rdf/
Streams	streams	https://neo4j.com/docs/labs/neo4j-streams/current/

Name	Key	Further information
Graph-algorithms	graph-algorithms	https://neo4j.com/docs/graph-algorithms/current/

4.5. Security

This chapter describes security in Neo4j when running in a Docker container.

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

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

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

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

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

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

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

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

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

Configure SSL via *neo4j.conf*

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

```
# Https SSL configuration
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
```



For more information on configuring SSL policies, see [Configuration](#).

For more information on configuring connectors, see [Configuration options](#).

Example 4. 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:{neo4j-version}
```

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 5. 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:{neo4j-version}
```

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.

4.6. Docker maintenance operations

Basic maintenance operations when running Neo4j in a Docker container.

4.6.1. Dump and load a Neo4j database (offline)

The [Neo4j dump and 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 two containers for each operation (dump and load), with the `--rm` flag.

Example 6. Invoke `neo4j-admin dump` to dump your database.

```
docker run --interactive --tty --rm \
--publish=7474:7474 --publish=7687:7687 \
--volume=$HOME/neo4j/data:/data \
--volume=$HOME/neo4j/backups:/backups \
--user="$(id -u):$(id -g)" \
neo4j:{neo4j-version} \
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 7. Invoke `neo4j-admin load` to load your data into the new database.

```
docker run --interactive --tty --rm \
--publish=7474:7474 --publish=7687:7687 \
--volume=$HOME/neo4j/data:/data \
--volume=$HOME/neo4j/backups:/backups \
--user="$(id -u):$(id -g)" \
neo4j:{neo4j-version} \
neo4j-admin load --from=/backups/<dump-name>.dump --database=neo4j --force
```

The volume that contains 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 [Dump and load databases](#).

For more information on managing volumes, see [the official Docker documentation](#).

4.6.2. 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.

Back up a database

The following are examples of how to back up a database.

Example 8. A `docker run` command that creates a container to be used for backing up an online database.

```
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:{neo4j-version}-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 9. Invoke `neo4j-admin backup` to back up an online database.

```
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 [Perform a backup](#).

Restore a database

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

Example 10. 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:{neo4j-version}-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 11. 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 12. 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>
```



For more information on the `neo4j-admin restore` syntax and options, see [Restore a backup](#).

Finally, you can use [the Cypher Shell tool](#) to verify that your data has been restored.

4.6.3. 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 dbms_allow_upgrade=true \
neo4j:{neo4j-version} \
```

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.

For more details on upgrading, see:

- Single-instance upgrade
- Upgrade a Neo4j Causal Cluster

4.7. Docker specific configuration settings

This chapter provides a conversion table for the Neo4j configuration settings to the Docker format.

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 `NE04J_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
<code>browser.allow_outgoing_connections</code>	<code>NE04J_browser_allow__outgoing__connections</code>
<code>browser.credential_timeout</code>	<code>NE04J_browser_credential__timeout</code>
<code>browser.post_connect_cmd</code>	<code>NE04J_browser_post__connect__cmd</code>
<code>browser.remote_content_hostname_whitelist</code>	<code>NE04J_browser_remote__content__hostname__whitelist</code>
<code>browser.retain_connection_credentials</code>	<code>NE04J_browser_retain__connection__credentials</code>
<code>causal_clustering.catch_up_client_inactivity_timeout</code>	<code>NE04J_causal__clustering_catch__up__client__inactivity__timeout</code>
<code>causal_clustering.catchup_batch_size</code>	<code>NE04J_causal__clustering_catchup__batch__size</code>
<code>causal_clustering.cluster_allow_reads_on_followers</code>	<code>NE04J_causal__clustering_cluster__allow__reads__on__followers</code>
<code>causal_clustering.cluster_binding_timeout</code>	<code>NE04J_causal__clustering_cluster__binding__timeout</code>
<code>causal_clustering.cluster_topology_refresh</code>	<code>NE04J_causal__clustering_cluster__topology__refresh</code>
<code>causal_clustering.command_applier_parallelism</code>	<code>NE04J_causal__clustering_command__applier__parallelism</code>
<code>causal_clustering.connect-randomly-to-server-group</code>	<code>NE04J_causal__clustering_connect-randomly-to-server-group</code>
<code>causal_clustering.discovery_advertised_address</code>	<code>NE04J_causal__clustering_discovery__advertised__address</code>
<code>causal_clustering.discovery_listen_address</code>	<code>NE04J_causal__clustering_discovery__listen__address</code>
<code>causal_clustering.discovery_type</code>	<code>NE04J_causal__clustering_discovery__type</code>
<code>causal_clustering.election_failure_detection_window</code>	<code>NE04J_causal__clustering_election__failure__detection__window</code>
<code>causal_clustering.enable_pre_voting</code>	<code>NE04J_causal__clustering_enable__pre__voting</code>
<code>causal_clustering.global_session_tracker_state_size</code>	<code>NE04J_causal__clustering_global__session__tracker__state__size</code>

Neo4j format	Docker format
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_formatation	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
causal_clustering.protocol_implementations.catchup	NEO4J_causal__clustering_protocol__implementations_cat_chup
causal_clustering.protocol_implementations.compression	NEO4J_causal__clustering_protocol__implementations_com pression
causal_clustering.protocol_implementations.raft	NEO4J_causal__clustering_protocol__implementations_raf t
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_logImplementation	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

Neo4j format	Docker format
causal_clustering.raft_log_rotation_size	NE04J_causal__clustering_raft__log__rotation__size
causal_clustering.raft_membership_state_size	NE04J_causal__clustering_raft__membership__state__size
causal_clustering.raft_term_state_size	NE04J_causal__clustering_raft__term__state__size
causal_clustering.raft_vote_state_size	NE04J_causal__clustering_raft__vote__state__size
causal_clustering.refuse_to_be_leader	NE04J_causal__clustering_refuse__to__be__leader
causal_clustering.replicated_lease_state_size	NE04J_causal__clustering_replicated__lease__state__size
causal_clustering.replication_leader_await_timeout	NE04J_causal__clustering_replication__leader__await__timeout
causal_clustering.replication_retry_timeout_base	NE04J_causal__clustering_replication__retry__timeout__base
causal_clustering.replication_retry_timeout_limit	NE04J_causal__clustering_replication__retry__timeout__limit
causal_clustering.server_groups	NE04J_causal__clustering_server__groups
causal_clustering.state_machine_apply_max_batch_size	NE04J_causal__clustering_state__machine__apply__max__batch__size
causal_clustering.state_machine_flush_window_size	NE04J_causal__clustering_state__machine__flush__window__size
causal_clustering.status_throughput_window	NE04J_causal__clustering_status__throughput__window
causal_clustering.store_copy_chunk_size	NE04J_causal__clustering_store__copy__chunk__size
causal_clustering.store_copy_max_retry_time_per_request	NE04J_causal__clustering_store__copy__max__retry__time__per__request
causal_clustering.transaction_advertised_address	NE04J_causal__clustering_transaction__advertised__address
causal_clustering.transaction_listen_address	NE04J_causal__clustering_transaction__listen__address
causal_clustering.unknown_address_logging_throttle	NE04J_causal__clustering_unknown__address__logging__throttle
causal_clustering.upstream_selection_strategy	NE04J_causal__clustering_upstream__selection__strategy
causal_clustering.user_defined_upstream_strategy	NE04J_causal__clustering_user__defined__upstream__strategy
cypher.default_language_version	NE04J_cypher_default__language__version
cypher.forbid_exhaustive_shortestpath	NE04J_cypher_forbid__exhaustive__shortestpath
cypher.forbid_shortestpath_common_nodes	NE04J_cypher_forbid__shortestpath__common__nodes
cypher.hints_error	NE04J_cypher_hints__error
cypher.lenient_create_relationship	NE04J_cypher_lenient__create__relationship
cypher.min_replan_interval	NE04J_cypher_min__replan__interval
cypher.planner	NE04J_cypher_planner
cypher.statistics_divergence_threshold	NE04J_cypher_statistics__divergence__threshold
db.temporal.timezone	NE04J_db_temporal_timezone
dbms.allow_single_automatic_upgrade	NE04J_dbms_allow__single__automatic__upgrade
dbms.allow_upgrade	NE04J_dbms_allow__upgrade
dbms.backup.enabled	NE04J_dbms_backup_enabled
dbms.backup.listen_address	NE04J_dbms_backup_listen__address
dbms.checkpoint	NE04J_dbms_checkpoint
dbms.checkpoint.interval.time	NE04J_dbms_checkpoint_interval_time

Neo4j format	Docker format
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
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_quoteEscaping	NEO4J_dbms_import_csv_legacy_quoteEscaping
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
dbms.index.fulltext.eventually_consistent_index_update_queue_max_length	NEO4J_dbms_index_fulltext_eventually_consistent_index_update_queue_max_length

Neo4j format	Docker format
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
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

Neo4j format	Docker format
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.database_max_size	NEO4J_dbms_memory_transaction_database_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
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

Neo4j format	Docker format
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_ur ls
dbms.security.auth_cache_max_capacity	NEO4J_dbms_security_auth_cache_max_capacity
dbms.security.auth_cache_ttl	NEO4J_dbms_security_auth_cache_ttl
dbms.security.auth_cache_use_ttl	NEO4J_dbms_security_auth_cache_use_ttl
dbms.security.auth_enabled	NEO4J_dbms_security_auth_enabled
dbms.security.auth_lock_time	NEO4J_dbms_security_auth_lock_time
dbms.security.auth_max_failed_attempts	NEO4J_dbms_security_auth_max_failed_attempts
dbms.security.authentication_providers	NEO4J_dbms_security_authentication_providers
dbms.security.authorization_providers	NEO4J_dbms_security_authorization_providers
dbms.security.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

Neo4j format	Docker format
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
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
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

Neo4j format	Docker format
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
metrics.prometheus.enabled	NEO4J_metrics_prometheus_enabled
metrics.prometheus.endpoint	NEO4J_metrics_prometheus_endpoint

Chapter 5. Configuration

This chapter describes the configuration of Neo4j components.

The topics described are:

- [The `neo4j.conf` file](#) — An introduction to the primary configuration file in Neo4j.
- [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.
- [Set initial password](#) — How to set an initial password.
- [Password and user recovery](#) — How to recover after a lost admin password.
- [Configure Neo4j connectors](#) — How to configure Neo4j connectors.
- [Configure dynamic settings](#) — How to configure certain Neo4j parameters while Neo4j is running.
- [Transaction logs](#) — The transaction logs record all write operations in the database.

For a complete reference of Neo4j configuration settings, see [\[configuration-settings\]](#).

5.1. The `neo4j.conf` file

Introduction of the `neo4j.conf` file, and its syntax.



For a complete reference of Neo4j configuration settings, see [\[configuration-settings\]](#).

5.1.1. 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 some settings which apply 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.

5.1.2. 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.

5.1.3. 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 12. JVM-specific settings

Setting	Description
<code>dbms.memory.heap.initial_size</code>	Sets the initial heap size for the JVM. By default, the JVM heap size is calculated based on the available system resources.
<code>dbms.memory.heap.max_size</code>	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.
<code>dbms.jvm.additional</code>	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.

5.1.4. List currently active settings

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

Example 13. 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" |
+-----+
```

See also [Dynamic settings](#) for information about dynamic settings.

5.1.5. Command expansion

Usage

In addition to providing values directly for settings, each setting can optionally execute a custom script to provide the value, using the syntax `dbms.setting=$(script_to_execute)`.

Example 14. Example scripts to calculate value

By providing the following in the `neo4j.conf` file, upon server start Neo4j will evaluate the scripts `expr 10 * 10` and `curl -E example.cert https://example.com/password`, and use the corresponding result as value:

```
dbms.max_databases=$(expr 10 * 10)
dbms.ssl.policy.private_key_password=$(curl -E example.cert https://example.com/password)
```



Scripts and their syntax differ between operating systems.

Scripts are run by the Neo4j process, and are expected to exit with code `0` within a reasonable period of time. All output of a script should be of a valid type for the setting. Failure to do so will prevent Neo4j from starting.

Enabling

This feature needs to be explicitly enabled by providing the `--expand-commands` argument to the Neo4j startup script. The server will fail to start if the config contains scripts, but the correct argument is not provided.

To start Neo4j with command expansion enabled, run:

```
neo4j start --expand-commands
```

In addition, the `neo4j.conf` file is required to be owned by the same user running the Neo4j process, and be read/writeable (but not executable) by only that user. Neo4j will not start if this is not in place.

5.2. File locations

An overview of where files are stored in the different Neo4j distributions, and the necessary file permissions for running Neo4j.

5.2.1. Default file locations

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

Table 13. Default file locations

File type	Description	Linux / macOS / Docker	Windows	Debian / RPM	Neo4j Desktop ^[4]
Bin	The Neo4j running script and built-in tools, such as, cypher-shell and neo4j-admin.	<code><neo4j-home>/bin</code>	<code><neo4j-home>\bin</code>	<code>/usr/bin</code>	From the Open dropdown menu of your Neo4j instance, select Terminal, and navigate to <code><installation-version>/bin</code> .

File type	Description	Linux / macOS / Docker	Windows	Debian / RPM	Neo4j Desktop ^[4]
Configuration ^[1]	The Neo4j configuration settings and the JMX access credentials.	<code><neo4j-home>/conf/neo4j.conf</code>	<code><neo4j-home>\conf\neo4j.conf</code>	<code>/etc/neo4j/neo4j.conf</code>	From the <i>Open</i> dropdown menu of your Neo4j instance, select <i>Terminal</i> , and navigate to <code><installation-version>/conf/neo4j.conf</code> .
Data ^[2]	All data-related content, such as, databases, transactions, cluster state (if applicable).	<code><neo4j-home>/data</code>	<code><neo4j-home>\data</code>	<code>/var/lib/neo4j/data</code>	From the <i>Open</i> dropdown menu of your Neo4j instance, select <i>Terminal</i> , and navigate to <code><installation-version>/data</code> .
Import	All CSV files that the command <code>LOAD CSV</code> uses as sources to import data in Neo4j.	<code><neo4j-home>/import</code>	<code><neo4j-home>\import</code>	<code>/var/lib/neo4j/import</code>	From the <i>Open</i> dropdown menu of your Neo4j instance, select <i>Terminal</i> , and navigate to <code><installation-version>/import</code> .
Labs ^[5]	Contains APOC Core.	<code><neo4j-home>/labs</code>	<code><neo4j-home>\labs</code>	<code>/usr/share/neo4j/labs</code>	From the <i>Open</i> dropdown menu of your Neo4j instance, select <i>Terminal</i> , and navigate to <code><installation-version>/labs</code> .
Lib	All Neo4j dependencies.	<code><neo4j-home>/lib</code>	<code><neo4j-home>\lib</code>	<code>/usr/share/neo4j/lib</code>	From the <i>Open</i> dropdown menu of your Neo4j instance, select <i>Terminal</i> , and navigate to <code><installation-version>/lib</code> .
Logs	The Neo4j log files.	<code><neo4j-home>/logs</code>	<code><neo4j-home>\logs</code>	<code>/var/log/neo4j/</code> ^[3]	From the <i>Open</i> dropdown menu of your Neo4j instance, select <i>Terminal</i> , and navigate to <code><installation-version>/logs</code> .
Metrics	The Neo4j built-in metrics for monitoring the Neo4j DBMS and each individual database.	<code><neo4j-home>/metrics</code>	<code><neo4j-home>\metrics</code>	<code>/var/lib/neo4j/metrics</code>	From the <i>Open</i> dropdown menu of your Neo4j instance, select <i>Terminal</i> , and navigate to <code><installation-version>/metrics</code> .
Plugins	Custom code that extends Neo4j, for example, user-defined procedures, functions, and security plugins.	<code><neo4j-home>/plugins</code>	<code><neo4j-home>\plugins</code>	<code>/var/lib/neo4j/plugins</code>	From the <i>Open</i> dropdown menu of your Neo4j instance, select <i>Terminal</i> , and navigate to <code><installation-version>/plugins</code> .

File type	Description	Linux / macOS / Docker	Windows	Debian / RPM	Neo4j Desktop ^[4]
Run	The processes IDs.	<neo4j-home>/run	<neo4j-home>\run	/var/lib/neo4j/run	From the <i>Open</i> dropdown menu of your Neo4j instance, select <i>Terminal</i> , and navigate to <installation-version>/run.

^[1]For details about neo4j.conf, see: [The neo4j.conf file](#).

^[2]The data directory is internal to Neo4j and its structure is subject to change between versions without notice.

^[3]To view the neo4j.log for Debian and RPM, use `journctl --unit=neo4j`.

^[4]Applicable to all operating systems where Neo4j Desktop is supported.

^[5]For more information, see [APOC User Guide □ Installation](#).

5.2.2. 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 14. 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.metrics=metrics
```

5.2.3. File permissions

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

Read only

- conf
- import
- bin
- lib
- plugins

- *certificates*

Read and write

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

Execute

- all files in *bin*

5.3. Ports

Ports relevant to a Neo4j installation.

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.



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 15. Listen address configuration settings overview

Name	Default port	Related configuration setting
Backup	6362	<code>dbms.backup.listen_address</code>
HTTP	7474	<code>dbms.connector.http.listen_address</code>
HTTPS	7473	<code>dbms.connector.https.listen_address</code>
Bolt	7687	<code>dbms.connector.bolt.listen_address</code>
Causal Cluster discovery management	5000	<code>causal_clustering.discovery_listen_address</code>
Causal Cluster transaction	6000	<code>causal_clustering.transaction_listen_address</code>
Causal Cluster RAFT	7000	<code>causal_clustering.raft_listen_addresses</code>
Causal Cluster routing connector	7688	<code>dbms.routing.listen_address</code>
Graphite monitoring	2003	<code>metrics.graphite.server</code>
Prometheus monitoring	2004	<code>metrics.prometheus.endpoint</code>
JMX monitoring	3637	<code>dbms.jvm.additional=-Dcom.sun.management.jmxremote.port=3637</code>
Remote debugging	5005	<code>dbms.jvm.additional=-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=:5005</code>



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 16. Advertised address configuration settings overview

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



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

5.3.1. Backup

Default port: `6362`

Table 17. Backup

Related configuration setting	Default value	Description
<code>dbms.backup.listen_address</code>	<code>127.0.0.1:6362</code>	Network interface and port for the backup server to listen on.
<code>dbms.backup.enabled</code>	<code>true</code>	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](#).

5.3.2. HTTP

Default port: `7474`

Table 18. HTTP connector

Related configuration setting	Default value	Description
<code>dbms.connector.http.listen_address</code>	<code>:7474</code>	Network interface and port for the HTTP connector to listen on.

Related configuration setting	Default value	Description
<code>dbms.connector.http.advertised_address</code>	<code>:7474</code>	Advertised hostname/IP-address and port for the HTTP connector.
<code>dbms.connector.http.enabled</code>	<code>true</code>	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](#).

5.3.3. HTTPS

Default port: `7473`

Table 19. HTTPS connector

Related configuration setting	Default value	Description
<code>dbms.connector.https.listen_address</code>	<code>:7473</code>	Network interface and port for the HTTPS connector to listen on.
<code>dbms.connector.https.advertised_address</code>	<code>:7473</code>	Advertised hostname/IP-address and port for the HTTPS connector.
<code>dbms.connector.https.enabled</code>	<code>false</code>	Enable the HTTPS connector.

- The network communication is encrypted.
- Used by Neo4j Browser and the HTTP API.

For more information, see [Configure connectors](#).

5.3.4. Bolt

Default port: `7687`

Table 20. Bolt connector

Related configuration setting	Default value	Description
<code>dbms.connector.bolt.listen_address</code>	<code>:7687</code>	Network interface and port for the Bolt connector to listen on.
<code>dbms.connector.bolt.advertised_address</code>	<code>:7687</code>	Advertised hostname/IP-address and port for the Bolt connector.
<code>dbms.connector.bolt.enabled</code>	<code>true</code>	Enable the Bolt connector.
<code>dbms.connector.bolt.tls_level</code>	<code>DISABLED</code>	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](#).

5.3.5. Causal Cluster

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

Table 21. 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_addresses

Table 22. 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](#)

5.3.6. Graphite monitoring

Default port: [2003](#)

Table 23. Graphite

Related configuration setting	Default value	Description
<code>metrics.graphite.server</code>	<code>:2003</code>	Hostname/IP-address and port of the Graphite server.
<code>metrics.graphite.enabled</code>	<code>false</code>	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](#).

5.3.7. Prometheus monitoring

Default port: [2004](#)

Table 24. Prometheus

Related configuration setting	Default value	Description
<code>metrics.prometheus.endpoint</code>	<code>localhost:2004</code>	Network interface and port for the Prometheus endpoint to listen on.
<code>metrics.prometheus.enabled</code>	<code>false</code>	Enable exporting metrics with the Prometheus endpoint.

For more information, see [Prometheus](#).

5.3.8. JMX monitoring

Default port: `3637`

Table 25. Java Management Extensions

Related configuration setting	Default value	Description
<code>dbms.jvm.additional=-Dcom.sun.management.jmxremote.port=3637</code>	<code>3637</code>	Additional setting for exposing the Java Management Extensions (JMX).

For further information, see [Java Reference □ JMX metrics](#) and [the official documentation on Monitoring and Management Using JMX](#).

5.3.9. Remote debugging

Default port: `5005`

Table 26. Remote debugging

Related configuration setting	Default value	Description
<code>dbms.jvm.additional=-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=:5005</code>	<code>:5005</code>	Additional setting for exposing remote debugging.

For more information, see the [Java Reference □ Setup for remote debugging](#).

5.4. Set an initial password

This section describes how to set an initial password for Neo4j.

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.

Syntax:

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

Example 15. 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 16. 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 'secret' 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 secret --require-password-change
```

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.

5.5. Password and user recovery

This section describes how to recover from a lost password, specifically for an admin user, 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.

It is recommended to block network connections during the recovery phase, so users should connect to Neo4j only via localhost. This can be achieved by editing the `neo4j.conf` file.

You can temporarily comment out the `dbms.connectors.default_listen_address` parameter:



```
#dbms.connectors.default_listen_address=<your_configuration>
```

or provide the specific localhost value:

```
dbms.connectors.default_listen_address=127.0.0.1
```

5.5.1. Recover a lost password

Use the following steps to set a new password (assuming your admin user is named `neo4j`):

1. Stop Neo4j:

```
$ bin/neo4j stop
```

2. Disable the `dbms.security.auth_enabled` parameter by modifying the `neo4j.conf` file:

```
dbms.security.auth_enabled=false
```

3. Start Neo4j:

```
$ bin/neo4j start
```

4. Modify the admin user password using a client such as [Cypher Shell](#), or the Neo4j Browser:

- Connect to the `system` database via Cypher Shell, and modify the admin user password:

```
$ bin/cypher-shell -d system  
neo4j@system> ALTER USER neo4j SET PASSWORD 'mynewpass';  
neo4j@system> :exit
```

- Alternatively, you can run the following statement on the `system` database via another client, such as the Neo4j Browser:

```
ALTER USER neo4j SET PASSWORD 'mynewpass';
```

5. Stop Neo4j:

```
$ bin/neo4j stop
```

6. Enable the `dbms.security.auth_enabled` parameter by modifying the `neo4j.conf` file.

You can achieve this either by commenting out `dbms.security.auth_enabled` (the default value is `true`), or by specifically setting `dbms.security.auth_enabled` to `true`:

```
#dbms.security.auth_enabled=false
```

or,

```
dbms.security.auth_enabled=true
```

7. Restart Neo4j:

```
$ bin/neo4j start
```

5.5.2. Recover an unassigned admin role

If you have no user assigned to the admin role, you can grant an admin role to an existing user (assuming your existing user is named `neo4j`):

1. Stop Neo4j:

```
$ bin/neo4j stop
```

2. Disable the `dbms.security.auth_enabled` parameter by modifying the `neo4j.conf` file:

```
dbms.security.auth_enabled=false
```

3. Start Neo4j:

```
$ bin/neo4j start
```

4. Grant the admin user role to an existing user using a client such as [Cypher Shell](#), or the Neo4j Browser:
- Connect to the `system` database via Cypher Shell, and grant the admin user role to an existing user:

```
$ bin/cypher-shell -d system  
neo4j@system> GRANT admin TO neo4j;  
neo4j@system> :exit
```

- Alternatively, you can run the following statement on the `system` database via another client, such as the Neo4j Browser:

```
GRANT admin TO neo4j;
```

5. Stop Neo4j:

```
$ bin/neo4j stop
```

6. Enable the `dbms.security.auth_enabled` parameter by modifying the `neo4j.conf` file.

You can achieve this either by commenting out `dbms.security.auth_enabled` (the default value is `true`), or by specifically setting `dbms.security.auth_enabled` to `true`:

```
#dbms.security.auth_enabled=false
```

or,

```
dbms.security.auth_enabled=true
```

7. Restart Neo4j:

```
$ bin/neo4j start
```

5.5.3. Recover the admin role

If you have removed the admin role from your system entirely, you can recreate the role with its original capabilities (but minus the ability to run admin procedures) by following these steps:

1. Stop Neo4j:

```
$ bin/neo4j stop
```

2. Disable the `dbms.security.auth_enabled` parameter by modifying the `neo4j.conf` file:

```
dbms.security.auth_enabled=false
```

3. Start Neo4j:

```
$ bin/neo4j start
```

4. Create a custom admin role using a client such as [Cypher Shell](#), or the Neo4j Browser:

- Connect to the `system` database via Cypher Shell, and grant the admin user role to an existing user:

```
$ bin/cypher-shell -d system

neo4j@system> CREATE ROLE admin;
neo4j@system> GRANT ALL DBMS PRIVILEGES ON DBMS TO admin;
neo4j@system> GRANT TRANSACTION MANAGEMENT ON DATABASE * TO admin;
neo4j@system> GRANT START ON DATABASE * TO admin;
neo4j@system> GRANT STOP ON DATABASE * TO admin;
neo4j@system> GRANT MATCH {*} ON GRAPH * TO admin;
neo4j@system> GRANT WRITE ON GRAPH * TO admin;
neo4j@system> GRANT ALL ON DATABASE * TO admin;

neo4j@system> :exit
```



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 admin TO neo4j;`

- Alternatively, you can run the following statement on the `system` database via another client, such as the Neo4j Browser:

```
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;
```



Before exiting your client, 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 admin TO neo4j;`

5. Stop Neo4j:

```
$ bin/neo4j stop
```

6. Enable the `dbms.security.auth_enabled` parameter by modifying the `neo4j.conf` file.

You can achieve this either by commenting out `dbms.security.auth_enabled` (the default value is `true`), or by specifically setting `dbms.security.auth_enabled` to `true`:

```
#dbms.security.auth_enabled=false
```

or,

```
dbms.security.auth_enabled=true
```

7. Restart Neo4j:

```
$ bin/neo4j start
```

5.6. Configure connectors

How to configure connectors (Bolt, HTTP, and HTTPS) for Neo4j.

5.6.1. Available connectors

The table below lists the available Neo4j connectors:

Table 27. 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.

5.6.2. 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 28. Configuration option suffixes for connectors

Option name	Default	Setting(s)	Description
enabled	<code>true</code> ^[1]	<code>dbms.connector.bolt.enabled</code> , <code>dbms.connector.http.enabled</code> , <code>dbms.connector.https.enabled</code> ^[2]	This setting allows the client connector to be enabled or disabled. When disabled, Neo4j does not listen for incoming connections on the relevant port. ^[1] When Neo4j is used in embedded mode, the default value is <code>false</code> . ^[2] The default value for <code>dbms.connector.https.enabled</code> is <code>false</code> .
listen_address	<code>127.0.0.1:<connector-default-port></code>	<code>dbms.connector.bolt.listen_address</code> , <code>dbms.connector.https.listen_address</code> , <code>dbms.connector.http.listen_address</code>	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 <code><ip-address>:<port-number></code> . See below for an example of usage.

Option name	Default	Setting(s)	Description
advertise_d_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.
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. 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 17. 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 18. 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
```

5.6.3. 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.

5.6.4. 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 19. 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 20. 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
```

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`.

5.7. Dynamic settings

How to change your Neo4j configuration while Neo4j is running, and which settings can be changed.

5.7.1. 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 make sure to update `neo4j.conf` as well.

5.7.2. Discover dynamic settings

Use the procedure `dbms.listConfig()` to discover which configuration values can be dynamically updated, or consult [Dynamic settings reference](#).

Example 21. Discover dynamic settings

```
CALL dbms.listConfig()
YIELD name, dynamic
WHERE dynamic
RETURN name
ORDER BY name
LIMIT 4;
```

```
+-----+
| name
+-----+
| "dbms.checkpoint.iops.limit"
| "dbms.logs.query.allocation_logging_enabled"
| "dbms.logs.query.enabled"
| "dbms.logs.query.page_logging_enabled"
+-----+
4 rows
```

5.7.3. 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 22. 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 23. 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 24. Reset a config value to default

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

5.7.4. Dynamic settings reference

Dynamic settings reference

- [causal_clustering.cluster_allow_reads_on_leader](#): Configure if the `dbms.routing.getRoutingTable()` procedure should include the leader as read endpoint or return only read replicas/followers.
- [causal_clustering.connect_randomly_to_server_group](#): Comma separated list of groups to be used by the connect-randomly-to-server-group selection strategy.
- [causal_clustering.server_groups](#): A list of group names for the server used when configuring load balancing and replication policies.
- [dbms.allow_single_automatic_upgrade](#): Whether to allow a system graph upgrade to happen automatically in single instance mode (dbms.mode=SINGLE).
- [dbms.allow_upgrade](#): Whether to allow a store upgrade in case the current version of the database starts against an older version of the store.
- [dbms.checkpoint.iops.limit](#): Limit the number of IOs the background checkpoint process will consume per second.
- [dbms.logs.debug.level](#): Debug log level threshold.
- [dbms.logs.query.allocation_logging_enabled](#): Log allocated bytes for the executed queries being logged.
- [dbms.logs.query.early_raw_logging_enabled](#): Log query text and parameters without obfuscating passwords.
- [dbms.logs.query.enabled](#): Log executed queries.
- [dbms.logs.query.page_logging_enabled](#): Log page hits and page faults for the executed queries being logged.
- [dbms.logs.query.parameter_full_entities](#): Log complete parameter entities including id, labels or relationship type, and properties.

- `dbms.logs.query.parameter_logging_enabled`: Log parameters for the executed queries being logged.
- `dbms.logs.query.rotation.keep_number`: Maximum number of history files for the query log.
- `dbms.logs.query.rotation.size`: The file size in bytes at which the query log will auto-rotate.
- `dbms.logs.query.runtime_logging_enabled`: Logs which runtime that was used to run the query.
- `dbms.logs.query.threshold`: If the execution of query takes more time than this threshold, the query is logged once completed - provided query logging is set to INFO.
- `dbms.logs.query.time_logging_enabled`: Log detailed time information for the executed queries being logged.
- `dbms.memory.pagecache.flush.buffer.enabled`: Page cache can be configured to use a temporal buffer for flushing purposes.
- `dbms.memory.pagecache.flush.buffer.size_in_pages`: Page cache can be configured to use a temporal buffer for flushing purposes.
- `dbms.memory.transaction.database_max_size`: 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').
- `dbms.memory.transaction.global_max_size`: 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').
- `dbms.memory.transaction.max_size`: 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').
- `dbms.track_query_allocation`: Enables or disables tracking of how many bytes are allocated by the execution of a query.
- `dbms.track_query_cpu_time`: Enables or disables tracking of how much time a query spends actively executing on the CPU.
- `dbms.transaction.concurrent.maximum`: The maximum number of concurrently running transactions.
- `dbms.transaction.sampling.percentage`: Transaction sampling percentage.
- `dbms.transaction.timeout`: The maximum time interval of a transaction within which it should be completed.
- `dbms.transaction.tracing.level`: Transaction creation tracing level.
- `dbms.tx_log.preallocate`: Specify if Neo4j should try to preallocate logical log file in advance.
- `dbms.tx_log.rotation.retention_policy`: 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.
- `dbms.tx_log.rotation.size`: Specifies at which file size the logical log will auto-rotate.
- `dbms.upgrade_max_processors`: Max number of processors used when upgrading the store.
- `fabric.routing.servers`: A comma-separated list of Fabric instances that form a routing group.

5.8. Transaction logs

The transaction logs record all write operations in the database.

- The transaction logs record all write operations in the database.
- The transaction logs are the "source of truth" in scenarios where the database needs to be recovered.

- They are 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.

See [Transaction logging](#).

Chapter 6. Manage databases

This chapter describes how to create and manage multiple active databases.

This chapter describes the following:

- [Introduction](#)
- [Administration and configuration](#)
- [Queries](#)
- [Error handling](#)
- [Databases in a Causal Cluster](#)

6.1. Introduction

Introduction to managing multiple active databases with Neo4j.

6.1.1. Concepts

With Neo4j 4.2 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 4.2 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:

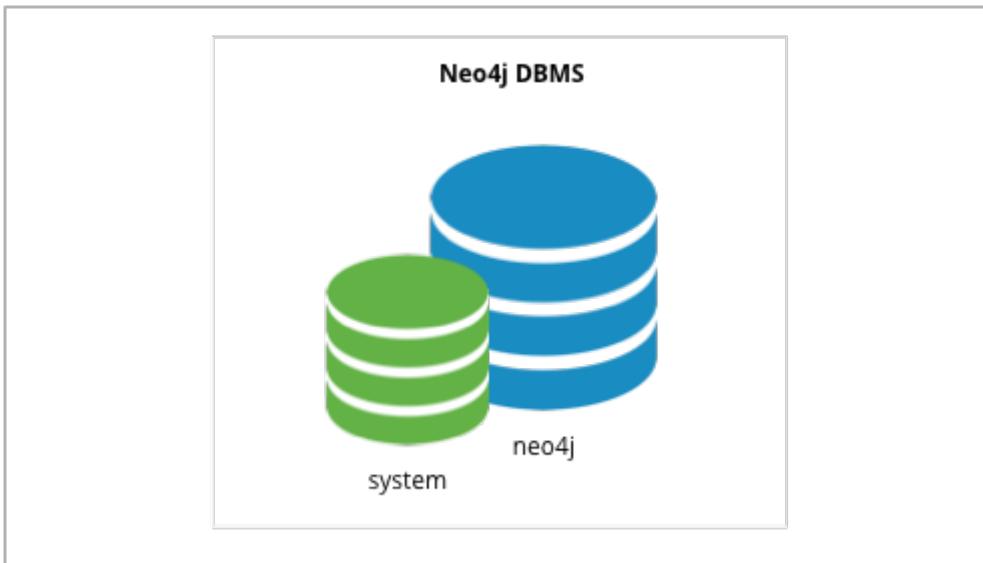


Figure 1. A default Neo4j installation.

Editions

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.

6.1.2. 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 □ Administration](#).

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 □ Security of administration](#).

The following image illustrates an installation of Neo4j with multiple active databases, named `marketing`, `sales`, and `hr`:

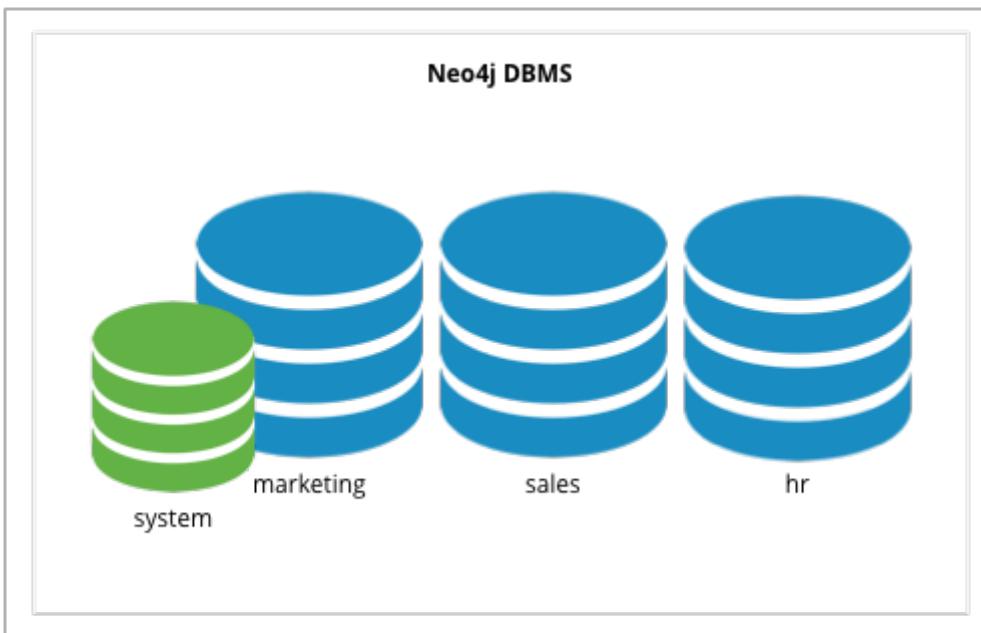


Figure 2. A multiple database Neo4j installation.

6.1.3. The default database

Each Neo4j instance has a default database. If a user connects to Neo4j without specifying a database, it will connect to the default database.

The default database is configurable. See [configuration parameters](#) for details.

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`:

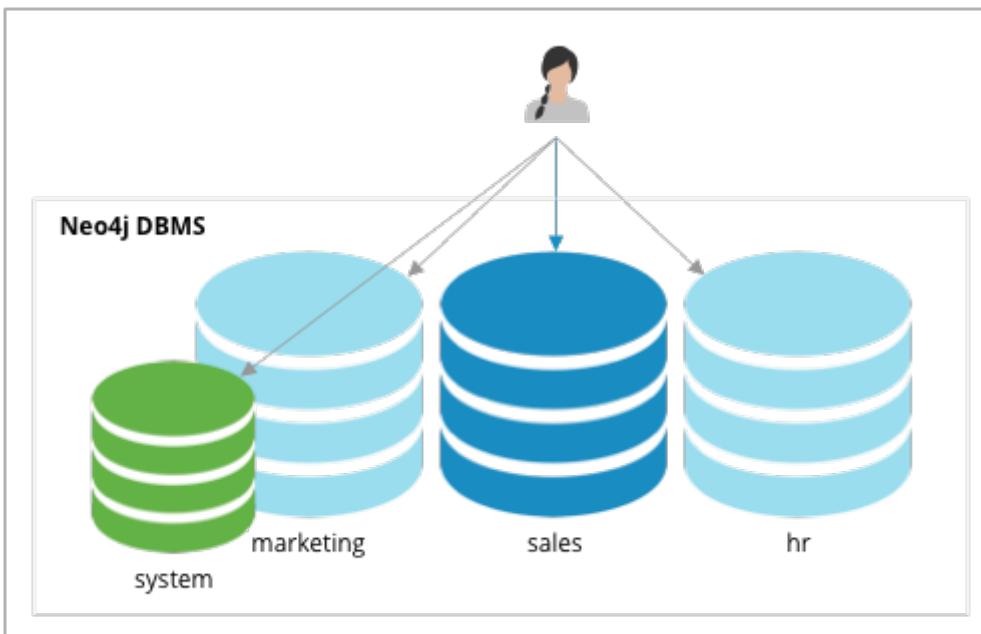


Figure 3. A multiple database Neo4j installation, with a default database.

6.2. Administration and configuration

How to manage multiple active databases.

6.2.1. Administrative commands

Administrative commands should not be used during a rolling upgrade. For more information, see [Rolling upgrade](#).



For detailed information on Cypher administrative commands, see [Cypher Manual Administration](#).

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
<code>CREATE DATABASE name</code> ^[1]	Create and start a new database.
<code>DROP DATABASE name</code> ^[1]	Drop (remove) an existing database.
<code>START DATABASE name</code>	Start a database that has been stopped.
<code>STOP DATABASE name</code>	Shut down a database.
<code>SHOW DATABASE name</code>	Show the status of a specific database.
<code>SHOW DATABASES</code>	Show the name and status of all the databases.
<code>SHOW DEFAULT DATABASE</code>	Show the name and status of the default database.

^[1]Enterprise Edition only.

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.



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.

For detailed information on Cypher administrative commands, see [Cypher Manual □ Administration](#).

For examples of using the Cypher administrative commands to manage multiple active databases, see [Queries](#).

6.2.2. Configuration parameters

Configuration parameters are defined in the [neo4j.conf](#) file.

The following configuration parameters are applicable for managing databases:

Parameter name	Description	Default value
<code>dbms.default_database</code>	Name of the default database for the Neo4j instance. The database is created if it does not exist when the instance starts.	<code>neo4j</code>
<code>dbms.max_databases</code> ^[1]	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. Note that 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.	100

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, use the `dbms.cluster.setDefaultDatabase(databaseName)` procedure. This procedure is only available when running a cluster.

^[1]*Enterprise Edition only.*

6.3. Queries

Examples of Cypher queries and commands that can be used to create and manage multiple active databases.



For detailed information on Cypher administrative commands, see [Cypher Manual □ Administration](#).



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.

6.3.1. Show the status of a specific database

Example 25. SHOW DATABASE

```
neo4j@system> SHOW DATABASE neo4j;
```

In standalone mode:

```
+-----+  
| name | address | role | requestedStatus | currentStatus | error | default |  
+-----+  
| "neo4j" | "localhost:7687" | "standalone" | "online" | "online" | "" | TRUE |  
+-----+  
  
1 row available after 100 ms, consumed after another 6 ms
```

Or in a Causal Cluster:

```
+-----+  
| name | address | role | requestedStatus | currentStatus | error | default |  
+-----+  
| "neo4j" | "localhost:7687" | "leader" | "online" | "online" | "" | TRUE |  
| "neo4j" | "localhost:7688" | "follower" | "online" | "online" | "" | TRUE |  
| "neo4j" | "localhost:7689" | "follower" | "online" | "online" | "" | TRUE |  
+-----+  
  
3 row available after 100 ms, consumed after another 6 ms
```

6.3.2. Show the status of all databases

Example 26. SHOW DATABASES

```
neo4j@system> SHOW DATABASES;
```

In standalone mode:

```
+-----+  
| name | address | role | requestedStatus | currentStatus | error | default |  
+-----+  
| "neo4j" | "localhost:7687" | "standalone" | "online" | "online" | "" | TRUE |  
| "system" | "localhost:7687" | "standalone" | "online" | "online" | "" | FALSE |  
+-----+
```

2 rows available after 5 ms, consumed after another 1 ms

Or in a Causal Cluster:

```
+-----+  
| name | address | role | requestedStatus | currentStatus | error | default |  
+-----+  
| "neo4j" | "localhost:7687" | "leader" | "online" | "online" | "" | TRUE |  
| "neo4j" | "localhost:7688" | "follower" | "online" | "online" | "" | TRUE |  
| "neo4j" | "localhost:7689" | "follower" | "online" | "online" | "" | TRUE |  
| "system" | "localhost:7687" | "follower" | "online" | "online" | "" | FALSE |  
| "system" | "localhost:7688" | "leader" | "online" | "online" | "" | FALSE |  
| "system" | "localhost:7689" | "follower" | "online" | "online" | "" | 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.

6.3.3. 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 27. SHOW DEFAULT DATABASE

```
neo4j@system> SHOW DEFAULT DATABASE;
```

In standalone mode:

```
+-----+  
| name | address | role | requestedStatus | currentStatus | error |  
+-----+  
| "neo4j" | "localhost:7687" | "standalone" | "online" | "online" | "" |  
+-----+  
  
1 row available after 57 ms, consumed after another 2 ms
```

Or in a Causal Cluster:

```
+-----+  
| name | address | role | requestedStatus | currentStatus | error |  
+-----+  
| "neo4j" | "localhost:7687" | "follower" | "online" | "online" | "" |  
| "neo4j" | "localhost:7688" | "leader" | "online" | "online" | "" |  
| "neo4j" | "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.



In Community Edition, the default database is the only database available, other than the `system` database.

6.3.4. Create a database

Example 28. 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 | address | role | requestedStatus | currentStatus | error | default |  
+-----+  
| "neo4j" | "localhost:7687" | "standalone" | "online" | "online" | "" | TRUE |  
| "system" | "localhost:7687" | "standalone" | "online" | "online" | "" | FALSE |  
| "sales" | "localhost:7687" | "standalone" | "online" | "online" | "" | FALSE |  
+-----+
```

```
3 rows available after 4 ms, consumed after another 1 ms
```

Or in a Causal Cluster:

```
+-----+  
| name | address | role | requestedStatus | currentStatus | error | default |  
+-----+  
| "neo4j" | "localhost:7687" | "leader" | "online" | "online" | "" | TRUE |  
| "neo4j" | "localhost:7688" | "follower" | "online" | "online" | "" | TRUE |  
| "neo4j" | "localhost:7689" | "follower" | "online" | "online" | "" | TRUE |  
| "system" | "localhost:7687" | "follower" | "online" | "online" | "" | FALSE |  
| "system" | "localhost:7688" | "leader" | "online" | "online" | "" | FALSE |  
| "system" | "localhost:7689" | "follower" | "online" | "online" | "" | FALSE |  
| "sales" | "localhost:7687" | "follower" | "online" | "online" | "" | FALSE |  
| "sales" | "localhost:7688" | "follower" | "online" | "online" | "" | FALSE |  
| "sales" | "localhost:7689" | "leader" | "online" | "online" | "" | FALSE |  
+-----+
```

```
9 rows available after 4 ms, consumed after another 1 ms
```

6.3.5. Switch a database

Example 29. :use <database-name>

```
neo4j@system> :use sales  
neo4j@sales>
```

6.3.6. Create or replace a database

Example 30. 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      | address          | role        | requestedStatus | currentStatus | error | default |  
+-----+  
| "neo4j"   | "localhost:7687" | "standalone" | "online"       | "online"     | ""    | TRUE   |  
| "system"  | "localhost:7687" | "standalone" | "online"       | "online"     | ""    | FALSE  |  
| "sales"   | "localhost:7687" | "standalone" | "online"       | "online"     | ""    | FALSE  |  
+-----+
```

```
3 rows available after 2 ms, consumed after another 2 ms
```

Or in a Causal Cluster:

```
+-----+  
| name      | address          | role        | requestedStatus | currentStatus | error | default |  
+-----+  
| "neo4j"   | "localhost:7687" | "leader"    | "online"       | "online"     | ""    | TRUE   |  
| "neo4j"   | "localhost:7688" | "follower"  | "online"       | "online"     | ""    | TRUE   |  
| "neo4j"   | "localhost:7689" | "follower"  | "online"       | "online"     | ""    | TRUE   |  
| "system"  | "localhost:7687" | "follower"  | "online"       | "online"     | ""    | FALSE  |  
| "system"  | "localhost:7688" | "leader"    | "online"       | "online"     | ""    | FALSE  |  
| "system"  | "localhost:7689" | "follower"  | "online"       | "online"     | ""    | FALSE  |  
| "sales"   | "localhost:7687" | "follower"  | "online"       | "online"     | ""    | FALSE  |  
| "sales"   | "localhost:7688" | "follower"  | "online"       | "online"     | ""    | FALSE  |  
| "sales"   | "localhost:7689" | "leader"    | "online"       | "online"     | ""    | 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
```

6.3.7. Stop a database

Example 31. 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	address	role	requestedStatus	currentStatus	error	default
"neo4j"	"localhost:7687"	"standalone"	"online"	"online"	"	TRUE
"system"	"localhost:7687"	"standalone"	"online"	"online"	"	FALSE
"sales"	"localhost:7687"	"standalone"	"offline"	"offline"	"	FALSE

```
3 rows available after 2 ms, consumed after another 1 ms
```

Or in a Causal Cluster:

name	address	role	requestedStatus	currentStatus	error	default
"neo4j"	"localhost:7687"	"leader"	"online"	"online"	"	TRUE
"neo4j"	"localhost:7688"	"follower"	"online"	"online"	"	TRUE
"neo4j"	"localhost:7689"	"follower"	"online"	"online"	"	TRUE
"system"	"localhost:7687"	"follower"	"online"	"online"	"	FALSE
"system"	"localhost:7688"	"leader"	"online"	"online"	"	FALSE
"system"	"localhost:7689"	"follower"	"online"	"online"	"	FALSE
"sales"	"localhost:7687"	"unknown"	"offline"	"offline"	"	FALSE
"sales"	"localhost:7688"	"unknown"	"offline"	"offline"	"	FALSE
"sales"	"localhost:7689"	"unknown"	"offline"	"offline"	"	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]>
```

6.3.8. Start a database

Example 32. 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      | address          | role       | requestedStatus | currentStatus | error | default |
+-----+
| "neo4j"   | "localhost:7687" | "standalone" | "online"      | "online"     | ""    | TRUE   |
| "system"  | "localhost:7687" | "standalone" | "online"      | "online"     | ""    | FALSE  |
| "sales"   | "localhost:7687" | "standalone" | "online"      | "online"     | ""    | FALSE  |
+-----+
```

```
3 rows available after 2 ms, consumed after another 1 ms
```

Or in a Causal Cluster:

```
+-----+
| name      | address          | role       | requestedStatus | currentStatus | error | default |
+-----+
| "neo4j"   | "localhost:7687" | "leader"   | "online"      | "online"     | ""    | TRUE   |
| "neo4j"   | "localhost:7688" | "follower" | "online"      | "online"     | ""    | TRUE   |
| "neo4j"   | "localhost:7689" | "follower" | "online"      | "online"     | ""    | TRUE   |
| "system"  | "localhost:7687" | "follower" | "online"      | "online"     | ""    | FALSE  |
| "system"  | "localhost:7688" | "leader"   | "online"      | "online"     | ""    | FALSE  |
| "system"  | "localhost:7689" | "follower" | "online"      | "online"     | ""    | FALSE  |
| "sales"   | "localhost:7687" | "follower" | "online"      | "online"     | ""    | FALSE  |
| "sales"   | "localhost:7688" | "follower" | "online"      | "online"     | ""    | FALSE  |
| "sales"   | "localhost:7689" | "leader"   | "online"      | "online"     | ""    | FALSE  |
+-----+
```

```
9 rows available after 2 ms, consumed after another 1 ms
```

6.3.9. Drop or remove a database

Example 33. DROP DATABASE

```
neo4j@system> DROP DATABASE sales;
```

```
0 rows available after 82 ms, consumed after another 1 ms
```

```
neo4j@system> SHOW DATABASES;
```

name	address	role	requestedStatus	currentStatus	error	default
"neo4j"	"localhost:7687"	"standalone"	"online"	"online"	" "	TRUE
"system"	"localhost:7687"	"standalone"	"online"	"online"	" "	FALSE

```
2 rows available after 6 ms, consumed after another 0 ms
```

6.4. Error handling

This section describes how to manage errors that you may encounter while managing databases.

When running the [database management queries](#), such as `CREATE DATABASE`, it is possible to encounter errors.

6.4.1. Observing errors

Because database management operations are performed asynchronously, these errors may not be returned immediately upon query execution. Instead, you must monitor the output of `SHOW DATABASE`; particularly the `error` and `currentStatus` columns.

Example 34. 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      | address          | role        | requestedStatus | currentStatus | error
| default   |
+-----+
| "foo"    | "localhost:7687" | "standalone" | "online"       | "dirty"       | "File system
permissions" | FALSE   |
+-----+
-----+
```

```
1 rows available after 4 ms, consumed after another 1 ms
```

In a Causal Cluster:

```
+-----+
| name      | address          | role        | requestedStatus | currentStatus | error
| default   |
+-----+
| "foo"    | "localhost:7687" | "leader"    | "online"       | "online"      | ""
| FALSE   |
| "foo"    | "localhost:7688" | "follower"  | "online"       | "online"      | ""
| FALSE   |
| "foo"    | "localhost:7689" | "follower"  | "online"       | "dirty"       | "File system
permissions" | FALSE   |
+-----+
-----+
```

```
3 row available after 100 ms, consumed after another 6 ms
```

6.4.2. 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
<code>initial</code>	The database has not yet been created.
<code>online</code>	The database is running.

Current state	Description
<code>offline</code>	The database is not running.
<code>store copying</code>	The database is currently being updated from another instance of Neo4j.
<code>dropped</code>	The database has been deleted.
<code>dirty</code>	This state implies an error has occurred. The database's underlying store files may be invalid. For more information, consult the server's logs.
<code>quarantined</code>	The database is effectively stopped and its state may not be changed until no longer quarantined.
<code>unknown</code>	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`.



Whilst `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.

6.4.3. 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.

Example 35. 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	address	role	requestedStatus	currentStatus	error	default
"foo"	"localhost:7687"	"standalone"	"online"	"offline"	"Some error message"	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	address	role	requestedStatus	currentStatus	error	default
"foo"	"localhost:7687"	"standalone"	"online"	"online"	"	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](#)  [Administration](#).



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.

6.4.4. Using quarantine in a cluster for fixing errors

You can use the `dbms.cluster.quarantineDatabase` procedure locally (only on the cluster member where it is executed) to isolate a specific database. For example, when a database is unable to start on a given member 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 cluster member and prevents its state from being changed, for

example, via the `START DATABASE` command. After lifting the quarantine, the cluster member tries to bring the database to the desired state.



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.cluster.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	<code>true</code> for placing the database into quarantine; <code>false</code> for lifting the quarantine.
reason	String	(Optional) The reason for placing the database in quarantine.

Returns:

Name	Type	Description
databaseName	String	The name of the database.
quarantined	String	Actual state.
result	String	Result of the last operation. The result contains the user, the time, and the reason for the quarantine.

Quarantine a database

```
neo4j@system> CALL dbms.cluster.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 | address      | role      | requestedStatus | currentStatus | error
| default |
+-----+
| "foo" | "localhost:7688" | "unknown" | "online"       | "quarantined" | "By neo4j at 2020-10-
15T15:10:41.348Z: No reason given" | FALSE    |
| "foo" | "localhost:7689" | "follower" | "online"       | "online"      | ""
| FALSE |
| "foo" | "localhost:7687" | "leader"   | "online"       | "online"      | ""
| FALSE |
+-----+
3 row available after 100 ms, consumed after another 6 ms

```

6.5. Databases in a Causal Cluster

This section describes how to manage multiple active 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.

6.5.1. Running 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 36. View the members of a Causal Cluster

```
neo4j@neo4j> CALL dbms.cluster.overview();
```

```
+-----+-----+
| id      | addresses          | groups |
+-----+-----+
| "8c...3d" | ["bolt://localhost:7683", "http://localhost:7473", "https://localhost:7483"] | {neo4j: "FOLLOWER", system: "FOLLOWER"} | []   |
| "8f...28" | ["bolt://localhost:7681", "http://localhost:7471", "https://localhost:7481"] | {neo4j: "LEADER", system: "LEADER"} | []   |
| "e0...4d" | ["bolt://localhost:7684", "http://localhost:7474", "https://localhost:7484"] | {neo4j: "READ_REPLICA", system: "READ_REPLICA"} | []   |
| "1a...64" | ["bolt://localhost:7682", "http://localhost:7472", "https://localhost:7482"] | {neo4j: "FOLLOWER", system: "FOLLOWER"} | []   |
| "59...87" | ["bolt://localhost:7685", "http://localhost:7475", "https://localhost:7485"] | {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 37. 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;
```

```
+-----+  
| 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;
```

```
+-----+  
| name | status | default |  
+-----+  
| "neo4j" | "online" | TRUE |  
| "system" | "online" | FALSE |  
| "data001" | "online" | FALSE |  
+-----+
```

```
3 rows available after 2 ms, consumed after another 1 ms
```

Example 38. 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;
```

```
+-----+  
| 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;
```

```
+-----+  
| 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.

Chapter 7. Clustering

This chapter describes the configuration and operation of a Neo4j Causal Cluster.

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.

Further information:

- For instructions on setting up Causal Clustering when running Neo4j in a Docker container, see [Causal Clustering on Docker](#).
- For an example of managing multiple databases in a Causal Cluster, see [Multiple databases in a Causal Cluster](#).
- For instructions on how you can upgrade your Neo4j Causal Cluster, see [Upgrading a Causal Cluster](#).
- For a summary of the facilities that are available for monitoring a Neo4j Causal Cluster, see [Monitoring](#) (and specifically, [Monitoring a Causal 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](#)

7.1. Introduction

Introduction to the Neo4j Causal Clustering architecture.

7.1.1. 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.

7.1.2. Operational view

From an operational point of view, it is useful to view the cluster as being composed of servers with two different roles: Cores and Read Replicas.

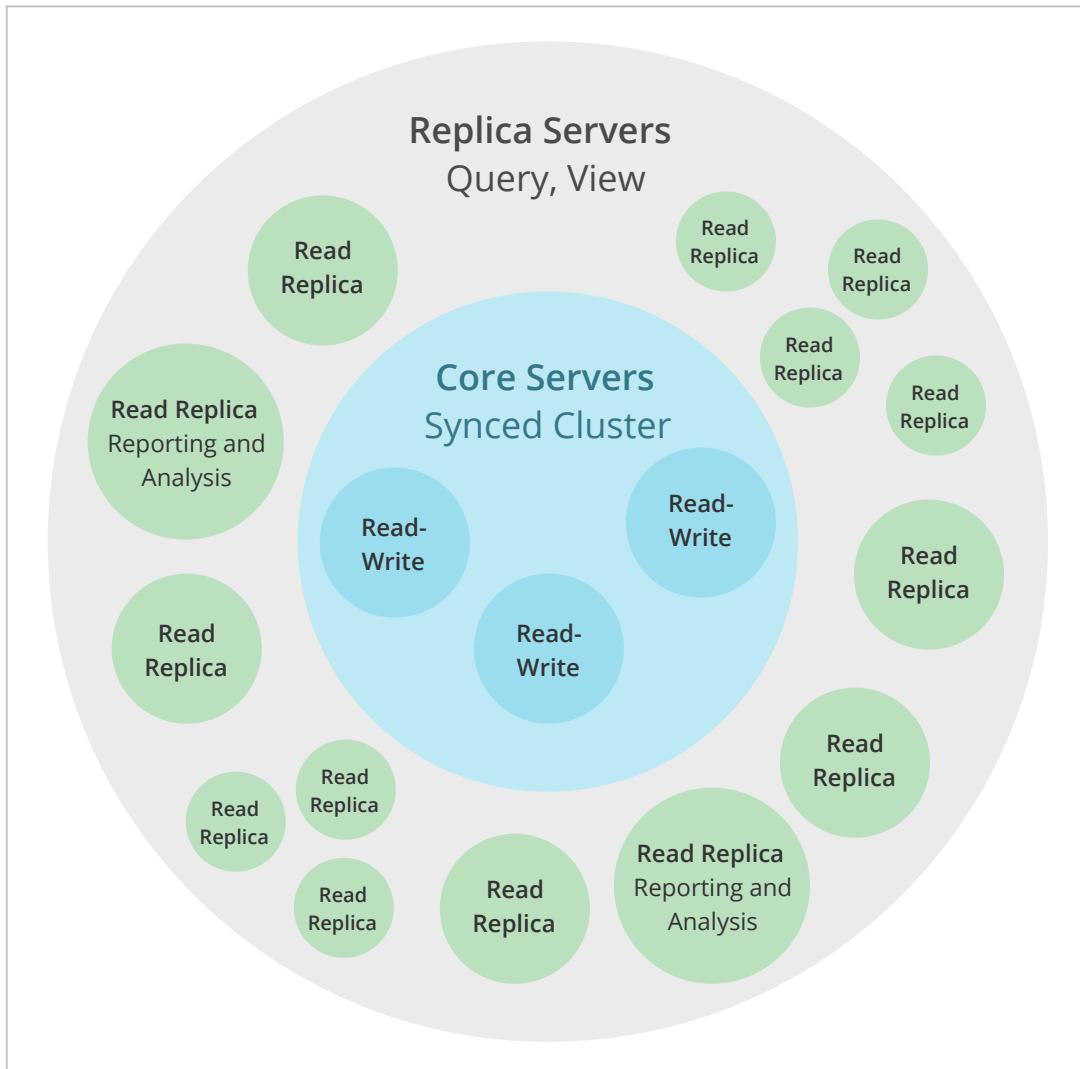


Figure 4. Causal Cluster Architecture

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.

Core Servers

The main responsibility of Core Servers is to safeguard data. Core Servers do so by replicating all transactions using the Raft protocol. Raft ensures that the data is safely durable before confirming transaction commit to the end user application. In practice this means once a majority of Core Servers 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 will be acknowledged by the fastest majority, but as the number of Core Servers in the cluster grows so do the size of the majority needed to acknowledge a write.

In practice this means that there are relatively few machines in a typical Core Server 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 Servers required to tolerate F faults. For example:

- In order to tolerate two failed Core Servers we would need to deploy a cluster of five Cores.
- The smallest *fault tolerant* cluster, a cluster that can tolerate one fault, must have three Cores.
- It is also possible to create a Causal Cluster consisting of only two Cores. However, that cluster will not be fault-tolerant. If one of the two servers fails, the remaining server will become read-only.



Should the cluster suffer enough Core failures then it can no longer process writes and it will become read-only to preserve safety.

Read Replicas

The main responsibility of Read Replicas is to scale out graph workloads. Read Replicas act like caches for the graph data that the Core Servers safeguard and are fully capable of executing arbitrary (read-only) queries and procedures.

Read Replicas are asynchronously replicated from Core 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 Core Servers, allowing for a large fan out of the query workload for scale.

Read Replicas 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.

7.1.3. 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 an application we typically want to read from the graph and write to the graph. Depending on the nature of the workload we usually want reads from the graph to take into account previous writes to ensure causal consistency.



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.

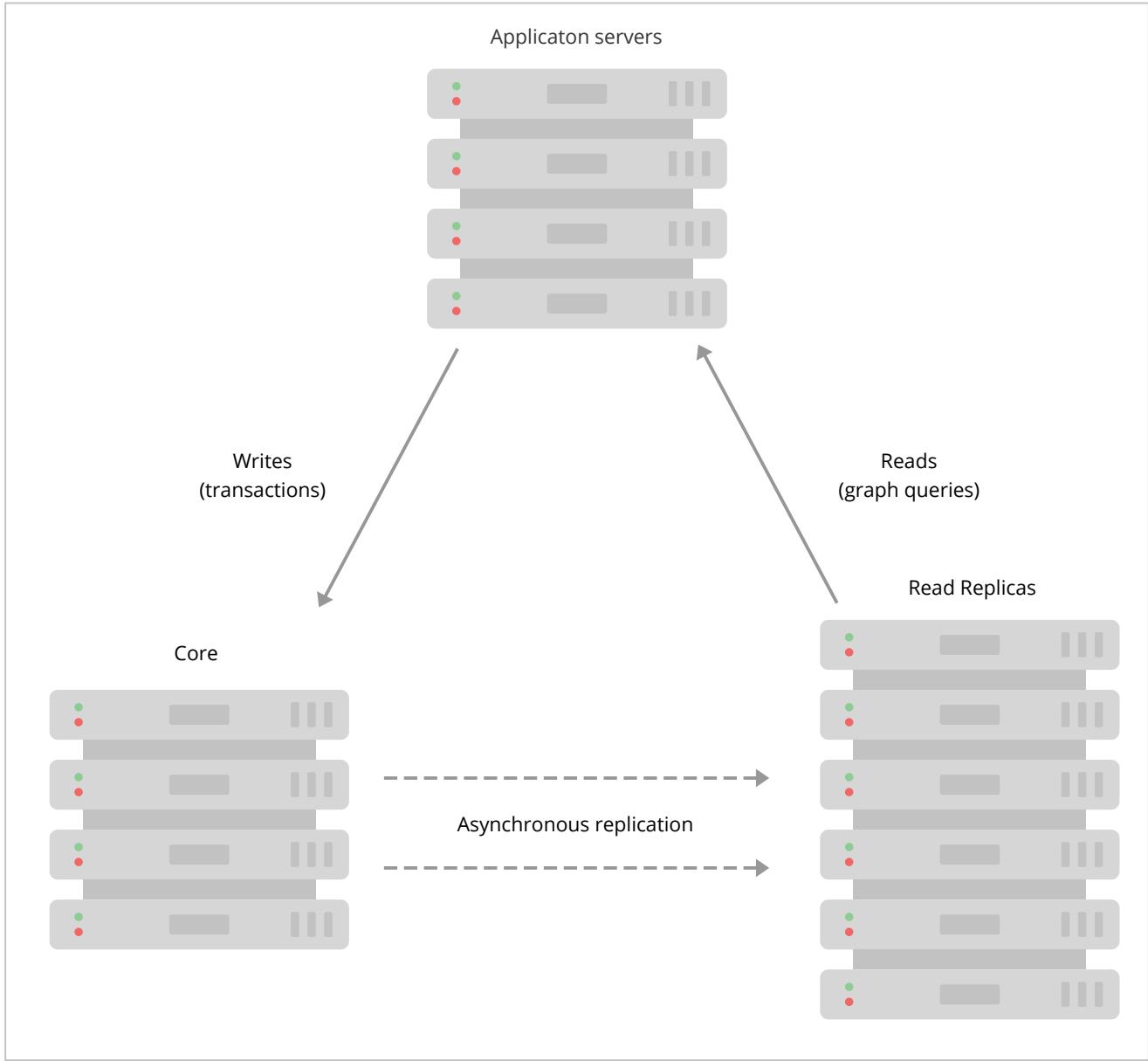


Figure 5. Causal Cluster setup with causal consistency via Neo4j drivers

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.

7.1.4. Summary

In this section we have examined Causal Clustering at a high level from an operational and an application development point of view. We now understand that the Core Servers are responsible for the long-term safekeeping of data while the more numerous Read Replicas are responsible for scaling out graph query workloads. Reasoning about this powerful architecture is greatly simplified by the Neo4j drivers which abstract the cluster topology to easily provide read levels like causal consistency.

7.2. Deploy a cluster

How to deploy a new Neo4j Causal Cluster.

7.2.1. Introduction

In this section we describe how to set up a new Causal Cluster consisting of three Core instances. We then proceed to show how more Core Servers as well as Read Replicas can be added to a running cluster.

Three Cores is the minimum number of servers needed in order to form a fault-tolerant Causal Cluster. See [Core Servers](#) for a discussion on the number of servers required in various scenarios.

Refer to [Set up a local Causal Cluster](#) for a tutorial on how to set up a Causal Cluster on a local machine.

7.2.2. Configure a Core-only cluster

The following configuration settings are important to consider when deploying a new Causal Cluster. See also [Settings reference](#) for more detailed descriptions and examples.

Table 29. Important settings for a new Causal Cluster

Option name	Description
<code>dbms.default_listen_address</code>	The address or network interface this machine uses to listen for incoming messages. Setting this value to <code>0.0.0.0</code> makes Neo4j bind to all available network interfaces.
<code>dbms.default_advertised_address</code>	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.
<code>dbms.mode</code>	The operating mode of a single server instance. For Causal Clustering, there are two possible modes: <code>CORE</code> or <code>READ_REPLICA</code> .
<code>causal_clustering.minimum_core_cluster_size_at_formatation</code>	The minimum number of Core machines 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.
<code>causal_clustering.minimum_core_cluster_size_at_runtime</code>	The minimum number of Core instances which will exist in the consensus group.
<code>causal_clustering.initial_discovery_members</code>	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 <code>:5000</code> . It is good practice to set this parameter to the same value on all Core Servers. The behavior of this setting can be modified by configuring the setting <code>causal_clustering.discovery_type</code> . This is described in detail in Discovery .



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 39. Configure a Core-only cluster

In this example, we will configure three Core Servers named `core01.example.com`, `core02.example.com` and `core03.example.com`. We have already installed Neo4j Enterprise Edition on all three servers. We configure them 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.example.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.example.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.example.com:5000
```

Now we are ready to start the Neo4j servers. The startup order does not matter.

After the cluster has started, we can connect to any of the instances and run `CALL dbms.cluster.overview()` to check the status of the cluster. This will show information about each member of the cluster.

We now have a Neo4j Causal Cluster of three instances running.



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`.

7.2.3. 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 40. Add a Core Server to an existing cluster

In this example, we will add a Core Server, `core04.example.com`, to the cluster that we created in [Configure a Core-only cluster](#).

We 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.example.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`.

Now we can start the new Core Server and let it add itself to the existing cluster.

7.2.4. Add a Read Replica to an existing cluster

Initial Read Replica configuration is provided similarly to Core Servers via `neo4j.conf`. Since Read Replicas do not participate in cluster quorum decisions, their configuration is shorter; they only need to know the addresses of some of the Core Servers which they can bind to in order to discover the cluster. They can then choose an appropriate Core Server from which to copy data.

Example 41. Add a Read Replica to an existing cluster

In this example, we will add a Read Replica, `replica01.example.com`, to the cluster that we created in [Configure a Core-only cluster](#).

We configure the following entries in `neo4j.conf`:

neo4j.conf on replica01.example.com:

```
dbms.mode=READ_REPLICA
causal_clustering.initial_discovery_members=core01.example.com:5000,core02.example.com:5000,core03.example.com:5000
```

Now we can start the new Read Replica and let it add itself to the existing cluster.

7.3. Seed a cluster

How to seed a new Neo4j Causal Cluster with existing data.

7.3.1. Introduction

In [Deploy a cluster](#) we learned how to create a cluster with empty databases. However, regardless of whether we are just playing around with Neo4j or setting up a production environment, it is likely that we have some existing data that we wish to transfer into our cluster.

This section outlines how to create a Causal Cluster containing data either seeded from an existing online or offline Neo4j database, or imported from some other data source using the import tool. The general steps to seed a cluster will follow the same pattern, regardless of which format our data is in:

1. Create a new Neo4j Core-only cluster.
2. Seed the cluster.
3. Start the cluster.



The databases which you are using to seed the cluster must be of the same version of Neo4j as the cluster itself.

7.3.2. Seed from backups

For this section, it is assumed that we already have healthy backups of an existing Neo4j deployment. This could be online or offline backups from a standalone Neo4j instance or a Neo4j Causal Cluster. For details on performing online backups, please refer to [Backup](#).



Moving files and directories manually in or out of a Neo4j installation is not recommended and considered unsupported usage. If you have an existing Neo4j database which you wish to use for a new cluster, then use `neo4j-admin dump` to create an offline backup.

The process described here can also be used to seed a new Causal Cluster from an existing Read Replica. This can be useful, for example, in disaster recovery where some servers have retained their data during a catastrophic event.

1. Create a new Neo4j Core-only cluster.

Follow the instructions in [Configure a Core-only cluster](#) to create a new Neo4j Core-only cluster.

You could start the cluster now in order to test that everything is correctly configured, but this will create default databases as part of cluster formation. Since you are trying to seed the cluster with a set of pre-existing databases, after your test-start you must remove every database whose name conflicts with one of your seeds. Remember that you can remove a database by executing `DROP DATABASE`. You may be unable drop some databases, for example because Neo4j is not running or because your seeds include a `system` database (which cannot be dropped). In this case you must instead stop every cluster instance, unbind them using `neo4j-admin unbind` and remove the store and transaction log files for the database in question. The locations of these files may be [configured](#).

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.

2. Seed the cluster.

Use `neo4j-admin restore` or `neo4j-admin load` to seed all the Core instances in the cluster.

The examples assume that we are restoring one user database with the default name of `neo4j` in addition to the `system` database which contains replicated configuration state. Modify the command line arguments to match your exact setup.

Example 42. Seed using neo4j-admin restore.

```
neo4j-01$ ./bin/neo4j-admin restore --from=/path/to/system-backup-dir --database=system  
neo4j-01$ ./bin/neo4j-admin restore --from=/path/to/neo4j-backup-dir --database=neo4j
```

```
neo4j-02$ ./bin/neo4j-admin restore --from=/path/to/system-backup-dir --database=system  
neo4j-02$ ./bin/neo4j-admin restore --from=/path/to/neo4j-backup-dir --database=neo4j
```

```
neo4j-03$ ./bin/neo4j-admin restore --from=/path/to/system-backup-dir --database=system  
neo4j-03$ ./bin/neo4j-admin restore --from=/path/to/neo4j-backup-dir --database=neo4j
```

Example 43. Seed using neo4j-admin load.

```
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 the cluster.

At this point, all of the instances in the Core cluster have been seeded. Between them, the Core Servers have everything necessary to form a cluster. We are ready to start all instances. The cluster will form and the replicated Neo4j DBMS deployment will come online.

Example 44. Start each of the Core instances.

```
neo4j-01$ ./bin/neo4j start
```

```
neo4j-02$ ./bin/neo4j start
```

```
neo4j-03$ ./bin/neo4j start
```

7.3.3. Seed using the import tool

In order 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.

- Deploy a standalone Neo4j DBMS.
- 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 from backups](#).

Skip the `system` database in this scenario since it is not needed.

7.4. Discovery

How members of a cluster discover each other.

7.4.1. 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.

If the addresses of the other cluster members are known upfront, they can be listed explicitly. This is convenient, but has limitations:

- If Core members 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 members as part of the discovery service.

The following DNS-based mechanisms can be used to get the addresses of Core Cluster members 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 member when a DNS lookup is performed. Each A record returned by DNS should contain the IP address of the Core Server. The configured Core 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 Cores 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 Core 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 Causal Cluster is running in [Kubernetes](#) and each Core Server is running as a Kubernetes service. Then, the addresses of the Core Cluster members 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.

-  • 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.

7.5. Intra-cluster encryption

How to secure the cluster communication between server instances.



Securing client to server communication is not covered in this chapter (e.g. Bolt, HTTPS, Backup).

7.5.1. 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.

7.5.2. 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.



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 45. 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 30. Example settings

Setting suffix	Value	Comment
<code>client_auth</code>	<code>REQUIRED</code>	Setting this to <code>REQUIRED</code> effectively enables mutual authentication for servers.
<code>ciphers</code>	<code>TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384</code>	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 (<i>AES</i>) for symmetric encryption which has great support for acceleration in hardware and thus allows performance to generally be negligibly affected.
<code>tls_versions</code>	<code>TLSv1.2</code>	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 46. 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 47. 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.

7.6. Internals of clustering

This section details a few selected internals of a Neo4j Causal Cluster. Understanding the internals is not vital but can be helpful in diagnosing and resolving operational issues.

7.6.1. Elections and leadership

The Core Servers in a Causal Cluster use the Raft protocol to ensure consistency and safety. An implementation detail of Raft is that it uses a *Leader* role to impose an ordering on an underlying log with other instances acting as *Followers* which replicate the leader's state. Specifically in Neo4j, this means that writes to the database are ordered by the Core instance currently playing the *Leader* role for the respective database. If multiple databases have been installed, each one of those databases

will operate within a logically separate Raft group, and therefore each have an individual leader. This means that a Core Server may act both as *Leader* for some databases, and as *Follower* for other databases.

If a follower has not heard from the leader for a while, then it can initiate an election and attempt to become the new leader. The follower makes itself a *Candidate* and asks other 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.

It is expected for elections 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).

7.6.2. 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.

7.6.3. Multi-database and the reconciler

Databases operate as independent entities in a Neo4j DBMS, both in standalone and in a cluster. Since a cluster consists 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 server could be the leader for one database and a follower for another.

7.6.4. Server-side routing

Server-side routing is a complement to the client-side routing, performed by a Neo4j Driver. It is activated for queries where the driver is oblivious to which database is accessed (such as queries with the [USE clause](#)), or when the drivers' routing information has become stale.

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 in write transactions, when a query addresses a database for which the receiving member is not the Leader. For more information, see [Driver Manual □ Routing transactions using access modes](#).

Server-side routing is enabled by the DBMS by setting `dbms.routing.enabled=true`. Client connections also need to state that server-side routing should be enabled.



Connections made using the `neo4j://` protocol will have server-side routing enabled, while connections made using `bolt://` will not.

Support for enabling server-side routing is available in Drivers version 4.1 and later.

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.*`



The configuration options described in [Driver Manual □ Configuration](#) have an equivalent in the server-side routing configuration.

Server-side routing encryption

Encryption of server-side routing communication is configured by the cluster SSL policy. For more information, see [Intra-cluster encryption](#).

7.6.5. Store copy

Store copies are initiated when an instance does not have an up-to-date copy of the database. For example, this will be 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 shutdown. Upon re-establishing connection with the cluster, an instance will recognize that it is too far behind and fetch 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 will be directed at an upstream member according to configuration and they will be retried despite transient failures. The maximum amount of time to retry every request can be modified by setting `causal_clustering.store_copy_max_retry_time_per_request`. If a request fails and the maximum retry time has elapsed then it will stop retrying and the store copy will fail.

Use `causal_clustering.catch_up_client_inactivity_timeout` to configure the inactivity timeout for any particular request.



This setting is for all requests from the catchup client, including the pulling of transactions.

The default upstream strategy differs for Cores and Read Replicas. Cores will 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 Cores is to vary every other request to a random Read Replica and every other

to a random Core member.

Read Replicas use the same strategy for store copies as it uses for pulling transactions. The default is to pull from a random Core member.

If you are running a multi-data center cluster, then upstream strategies for both Cores and Read Replicas can be configured. Remember that for Read Replicas this also affects from where transactions are pulled. See more in [Configure for multi-data center operations](#).

7.6.6. On-disk state

The on-disk state of cluster instances is different to that of standalone instances. The biggest difference being the existence of 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. This is because the cluster and the standalone deployment now have separate databases, with different and irreconcilable writes applied to them.



If you try to reinsert the 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 will not happen and will likely lead to unpredictable cluster behavior.

7.7. Settings reference

This section lists the important settings related to running a Neo4j Causal Cluster.

Parameter	Explanation
<code>dbms.mode</code>	This setting configures the operating mode of the database. For Causal Clustering, there are two possible modes: CORE or READ_REPLICA . Example: <code>dbms.mode=READ_REPLICA</code> will define this server as a Read Replica.
<code>dbms.read_only</code>	This setting is not supported.
<code>causal_clustering.minimum_core_cluster_size_atFormation</code>	Minimum number of Core machines required to form a cluster. Example: <code>causal_clustering.minimum_core_cluster_size_atFormation=3</code> will specify that the cluster will form when at least three Core members have discovered each other.

Parameter	Explanation
<code>causal_clustering.minimum_core_cluster_size_at_runtime</code>	<p>The minimum size of the dynamically adjusted voting set (which only Core members may be a part of).</p> <p>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. 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.</p> <p>A majority of the voting set must be available before members are voted in or out.</p> <p>Example: <code>causal_clustering.minimum_core_cluster_size_at_runtime=3</code> will specify that the cluster should not try to dynamically adjust below three Core members in the voting set.</p>
<code>causal_clustering.discovery_type</code>	<p>This setting specifies the strategy that the instance will use to determine the addresses for other instances in the cluster to contact for bootstrapping. Possible values are: <code>LIST</code>, <code>DNS</code>, <code>SRV</code>, and <code>K8S</code>.</p> <p>LIST Treat <code>causal_clustering.initial_discovery_members</code> as a list of addresses of Core Servers to contact for discovery.</p> <p>DNS Treat <code>causal_clustering.initial_discovery_members</code> as a domain name to resolve via DNS. Expect DNS resolution to provide A records with hostnames or IP addresses of Cores to contact for discovery, on the port specified by <code>causal_clustering.initial_discovery_members</code>.</p> <p>SRV Treat <code>causal_clustering.initial_discovery_members</code> as a domain name to resolve via DNS. Expect DNS resolution to provide SRV records with hostnames or IP addresses, and ports, of Cores to contact for discovery.</p> <p>K8S Access the Kubernetes list service API to derive addresses of Cores to contact for discovery. Requires <code>causal_clustering.kubernetes.label_selector</code> to be a Kubernetes label selector for Kubernetes services running a Core each and <code>causal_clustering.kubernetes.service_port_name</code> to be a service port name identifying the discovery port of Core services. The value of <code>causal_clustering.initial_discovery_members</code> is ignored for this option.</p> <p>The value of this setting determines how <code>causal_clustering.initial_discovery_members</code> is interpreted. Detailed information about discovery and discovery configuration options is given in Discovery using DNS with multiple records.</p> <p>Example: <code>causal_clustering.discovery_type=DNS</code> combined with <code>causal_clustering.initial_discovery_members=cluster01.example.com:5000</code> will fetch all DNS A records for <code>cluster01.example.com</code> and attempt to reach Neo4j instances listening on port <code>5000</code> for each A record's IP address.</p>

Parameter	Explanation
<code>causal_clustering.initial_discovery_members</code>	<p>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.</p> <p>It is good practice to set this parameter to the same value on all Core Servers.</p> <p>The behavior of this setting can be modified by configuring the setting <code>causal_clustering.discovery_type</code>. This is described in detail in Discovery using DNS with multiple records.</p> <p>Example: <code>causal_clustering.discovery_type=LIST</code> combined with <code>core01.example.com:5000,core02.example.com:5000,core03.example.com:5000</code> will attempt to reach Neo4j instances listening on <code>core01.example.com</code>, <code>core01.example.com</code> and <code>core01.example.com</code>; all on port <code>5000</code>.</p>
<code>causal_clustering.discovery_advertised_address</code>	<p>The address/port setting that specifies where the instance advertises that it will listen for discovery protocol messages from other members of the cluster. If this instance is included in the <code>initial_discovery_members</code> of other cluster members, the value there must exactly match this advertised address.</p> <p>Example: <code>causal_clustering.discovery_advertised_address=192.168.33.21:5001</code> indicates that other cluster members can communicate with this instance using the discovery protocol at host <code>192.168.33.20</code> and port <code>5001</code>.</p>
<code>causal_clustering.raft_advertised_address</code>	<p>The address/port setting that specifies where the Neo4j instance advertises to other members of the cluster that it will listen for Raft messages within the Core cluster.</p> <p>Example: <code>causal_clustering.raft_advertised_address=192.168.33.20:7000</code> will listen for cluster communication in the network interface bound to <code>192.168.33.20</code> on port <code>7000</code>.</p>
<code>causal_clustering.transaction_advertised_address</code>	<p>The address/port setting that specifies where the instance advertises where it will listen for requests for transactions in the transaction-shipping catchup protocol.</p> <p>Example: <code>causal_clustering.transaction_advertised_address=192.168.33.20:6001</code> will listen for transactions from cluster members on the network interface bound to <code>192.168.33.20</code> on port <code>6001</code>.</p>
<code>causal_clustering.discovery_listen_address</code>	<p>The address/port setting that specifies which network interface and port the Neo4j instance will bind to for the cluster discovery protocol.</p> <p>Example: <code>causal_clustering.discovery_listen_address=0.0.0.0:5001</code> will listen for cluster membership communication on any network interface at port <code>5001</code>.</p>

Parameter	Explanation
<code>causal_clustering.raft_listen_address</code>	The address/port setting that specifies which network interface and port the Neo4j instance will bind to for cluster communication. This setting must be set in coordination with the address this instance advertises it will listen at in the setting <code>causal_clustering.raft_advertised_address</code> . Example: <code>causal_clustering.raft_listen_address=0.0.0.0:7000</code> will listen for cluster communication on any network interface at port <code>7000</code> .
<code>causal_clustering.transaction_listen_address</code>	The address/port setting that specifies which network interface and port the Neo4j instance will bind to for cluster communication. This setting must be set in coordination with the address this instance advertises it will listen at in the setting <code>causal_clustering.transaction_advertised_address</code> . Example: <code>causal_clustering.transaction_listen_address=0.0.0.0:6001</code> will listen for cluster communication on any network interface at port <code>6001</code> .
<code>causal_clustering.store_copy_max_retry_time_per_request</code>	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. Example: <code>causal_clustering.store_copy_max_retry_time_per_request=60min</code>

7.7.1. Multi-data center settings

Parameter	Explanation
<code>causal_clustering.multi_dc_license</code>	Enables multi-data center features. Requires appropriate licensing. Example: <code>causal_clustering.multi_dc_license=true</code> will enable the multi-data center features.
<code>causal_clustering.server_groups</code>	A list of group names for the server used when configuring load balancing and replication policies. Example: <code>causal_clustering.server_groups=us,us-east</code> will add the current instance to the groups <code>us</code> and <code>us-east</code> .
<code>causal_cluster.leadership_priority_group>></code>	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 <i>is</i> 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 <code>leadership_priority_group</code> setting. Example: <code>causal_cluster.leadership_priority_group.foo=us</code> will ensure that if the leader for <code>foo</code> is not held by a server configured with <code>causal_clustering.server_groups=us</code> , the cluster will attempt to transfer leadership to a server which is.
<code>causal_clustering.upstream_selection_strategy</code>	An ordered list in descending preference of the strategy which Read Replicas use to choose upstream database server from which to pull transactional updates. Example: <code>causal_clustering.upstream_selection_strategy=connect-randomly-within-server-group,typically-connect-to-random-read-replica</code> will configure the behavior so that the Read Replica will first try to connect to any other instance in the group(s) specified in <code>causal_clustering.server_groups</code> . Should we fail to find any live instances in those groups, then we will connect to a random Read Replica. A value of <code>user-defined</code> will enable custom strategy definitions using the setting <code>causal_clustering.user_defined_upstream_strategy</code> .

Parameter	Explanation
<code>causal_clustering.user_defined_upstream_strategy</code>	<p>Defines the configuration of upstream dependencies. Can only be used if <code>causal_clustering.upstream_selection_strategy</code> is set to <code>user-defined</code>.</p> <p>Example: <code>causal_clustering.user_defined_upstream_strategy=groups(north2); groups(north); halt()</code>; will look for servers in the <code>north2</code>. If none are available it will look in the <code>north</code> server group. Finally, if we cannot resolve any servers in any of the previous groups, then rule chain will be stopped via <code>halt()</code>.</p>
<code>causal_clustering.load_balancing.plugin</code>	<p>The load balancing plugin to use. One pre-defined plugin named <code>server_policies</code> is available by default.</p> <p>Example: <code>causal_clustering.load_balancing.plugin=server_policies</code> will enable custom policy definitions.</p>
<code>causal_clustering.load_balancing.config.server_policies</code>	<p>Defines a custom policy under the name <code><policy-name></code>. Note that load balancing policies are cluster-global configurations and should be defined the exact same way on all core machines.</p> <p>Example: <code>causal_clustering.load_balancing.config.server_policies.north1_only=groups(north1) >min(2); halt();</code>; will define a load balancing policy named <code>north1_only</code>. Queries are only sent to servers in the <code>north1</code> server group, provided there are two of them available. If there are less than two servers in <code>north1</code> then the chain is halted.</p>

Chapter 8. Fabric

This chapter describes the configuration and operation of Neo4j Fabric.

This chapter describes the following:

- [Introduction](#)
- [Configuration](#)
- [Queries](#)
- [Further Considerations](#)

8.1. Introduction

An introduction of Neo4j Fabric.

8.1.1. Overview

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.

8.1.2. 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 [Driver Manual □ Databases and execution context](#).

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`.

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.

8.1.3. 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.

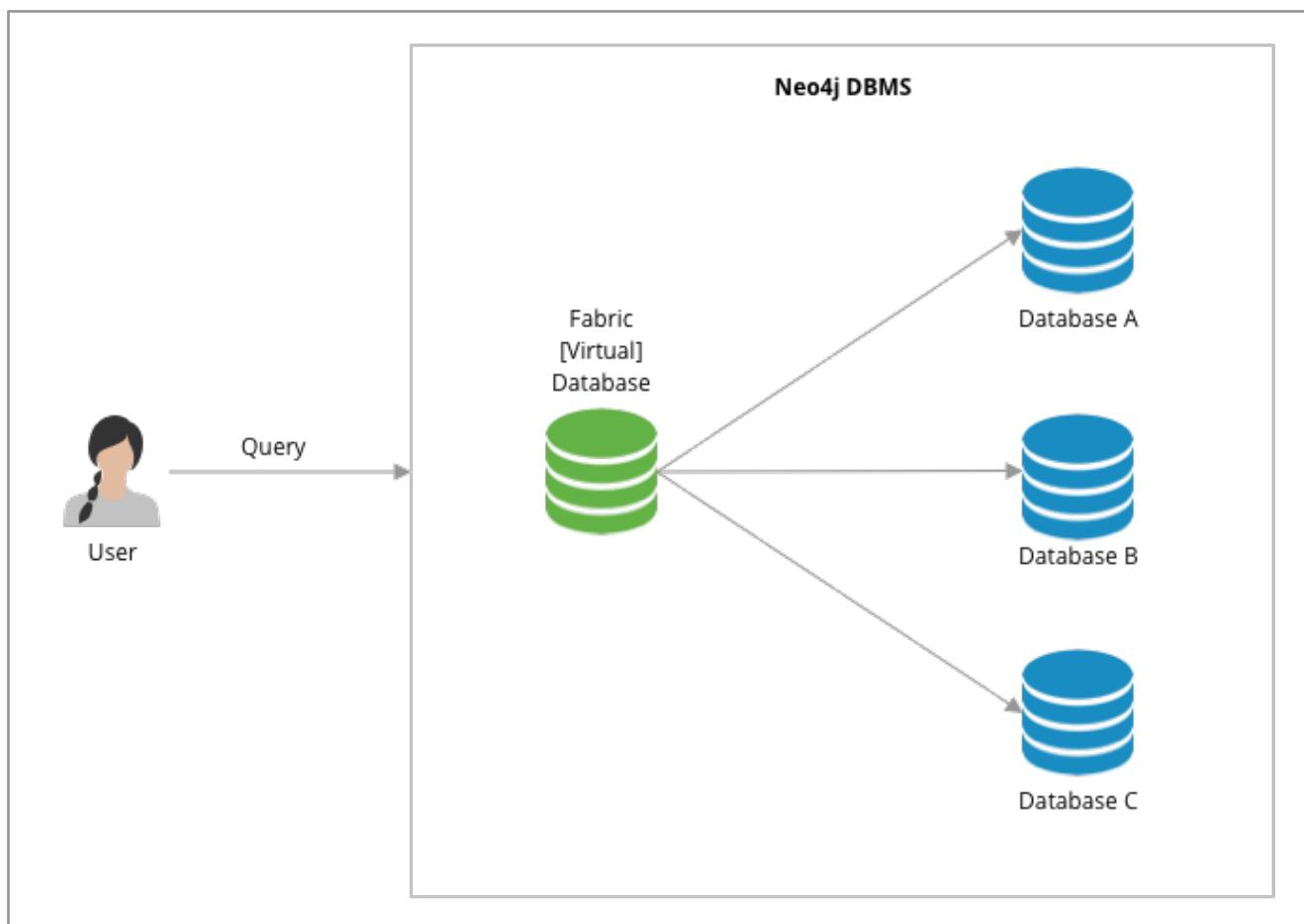


Figure 6. Fabric deployment in a single instance

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. 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.

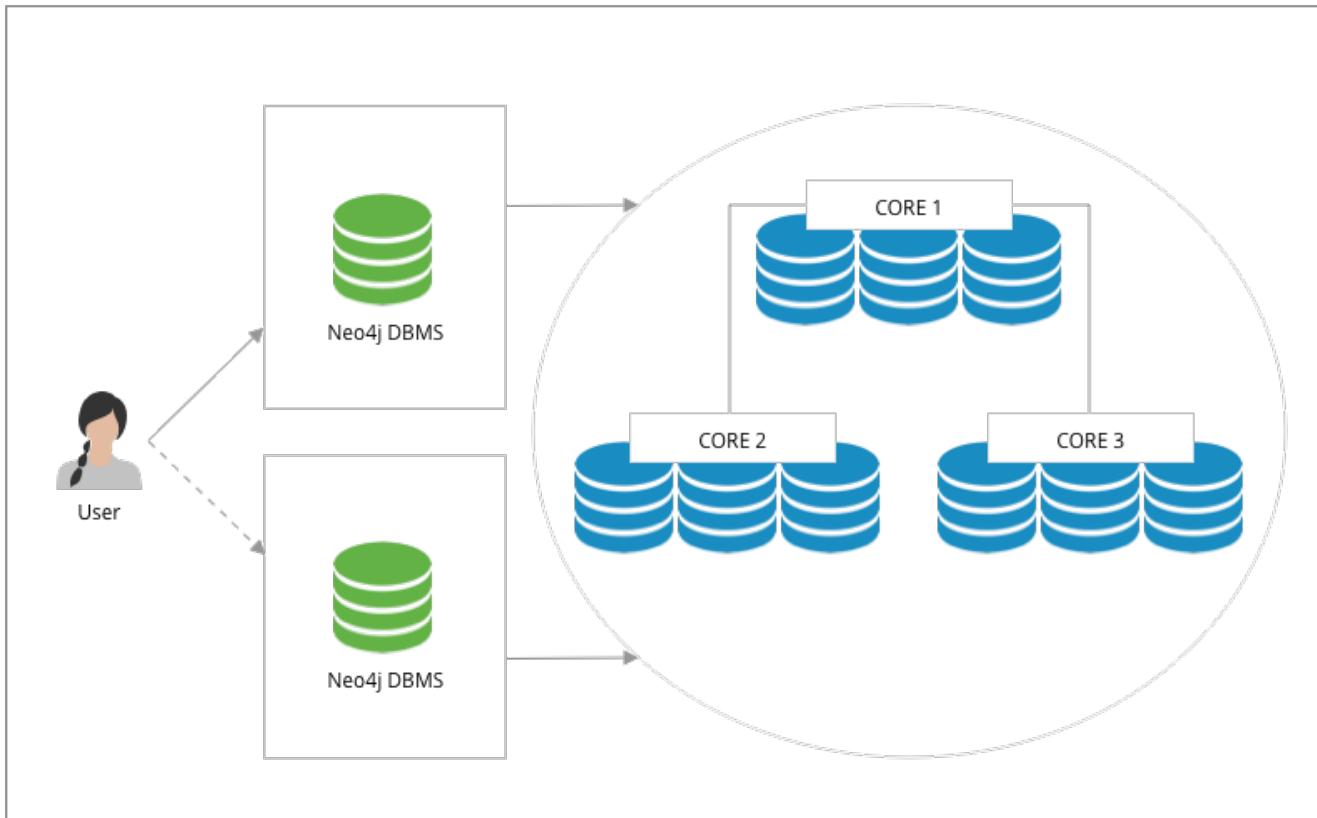


Figure 7. Fabric deployment with no single point of failure

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.

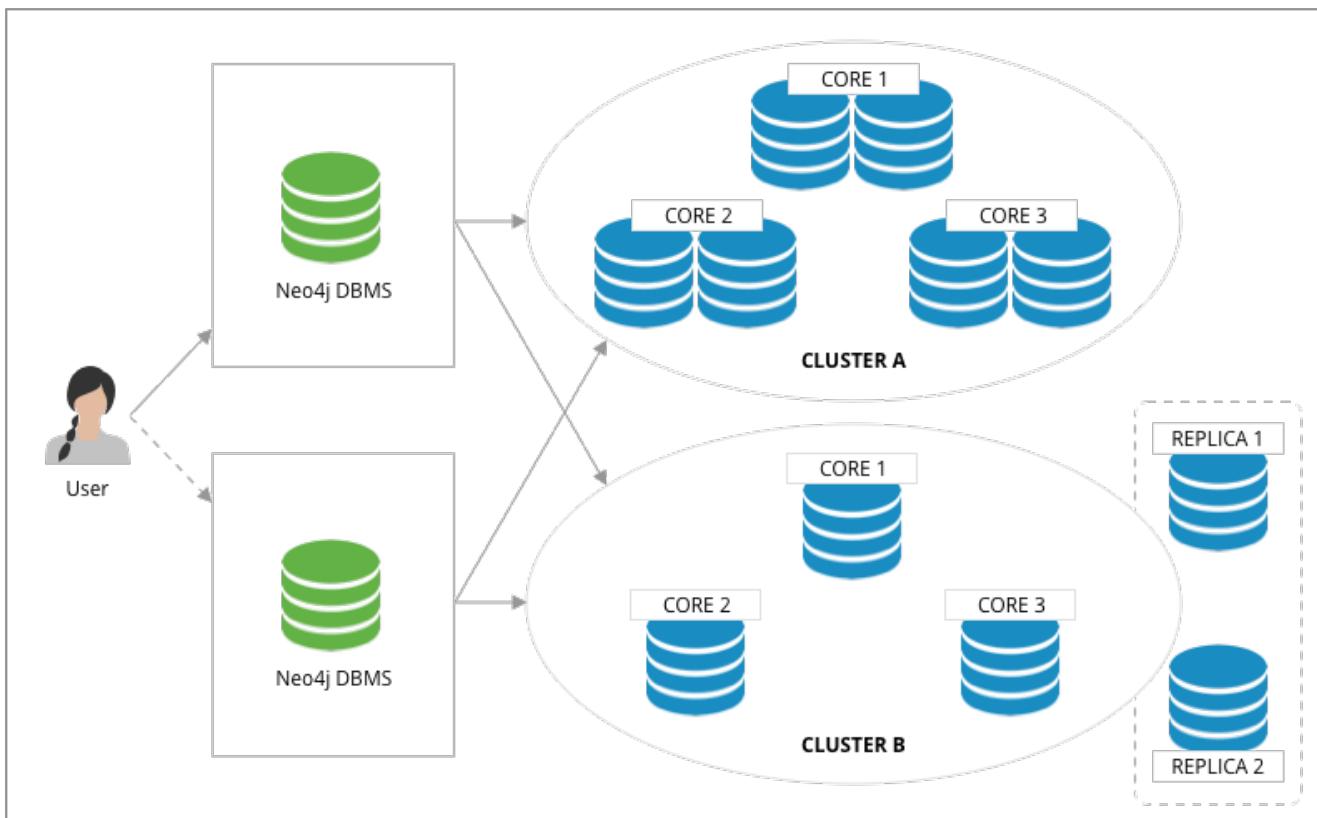


Figure 8. Fabric deployment for scalability with no single point of failure

8.2. Configuration

How to configure Neo4j Fabric.

8.2.1. 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, i.e., `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

```

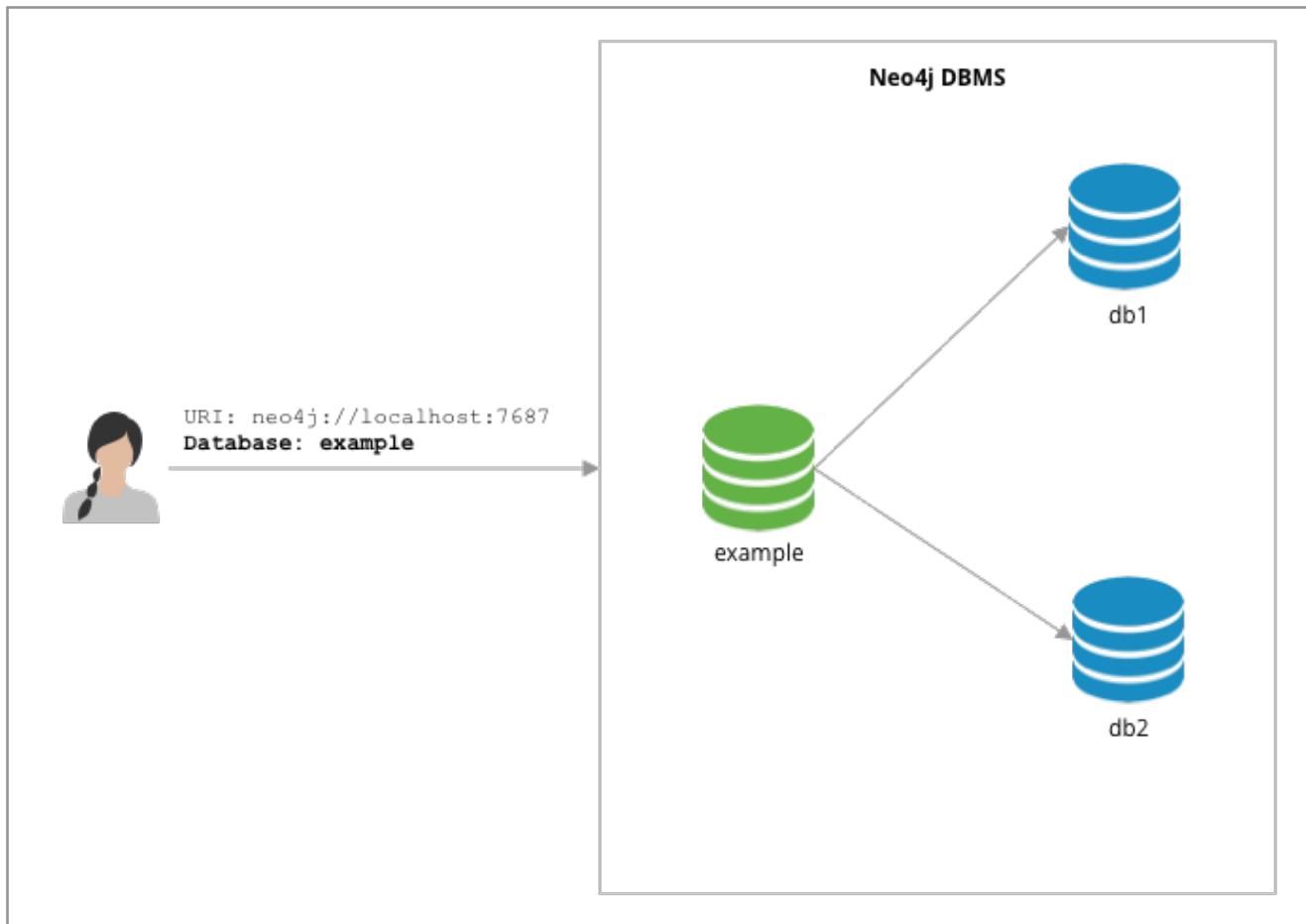


Figure 9. Minimal local Fabric setting in a development setup

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, i.e. `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

```

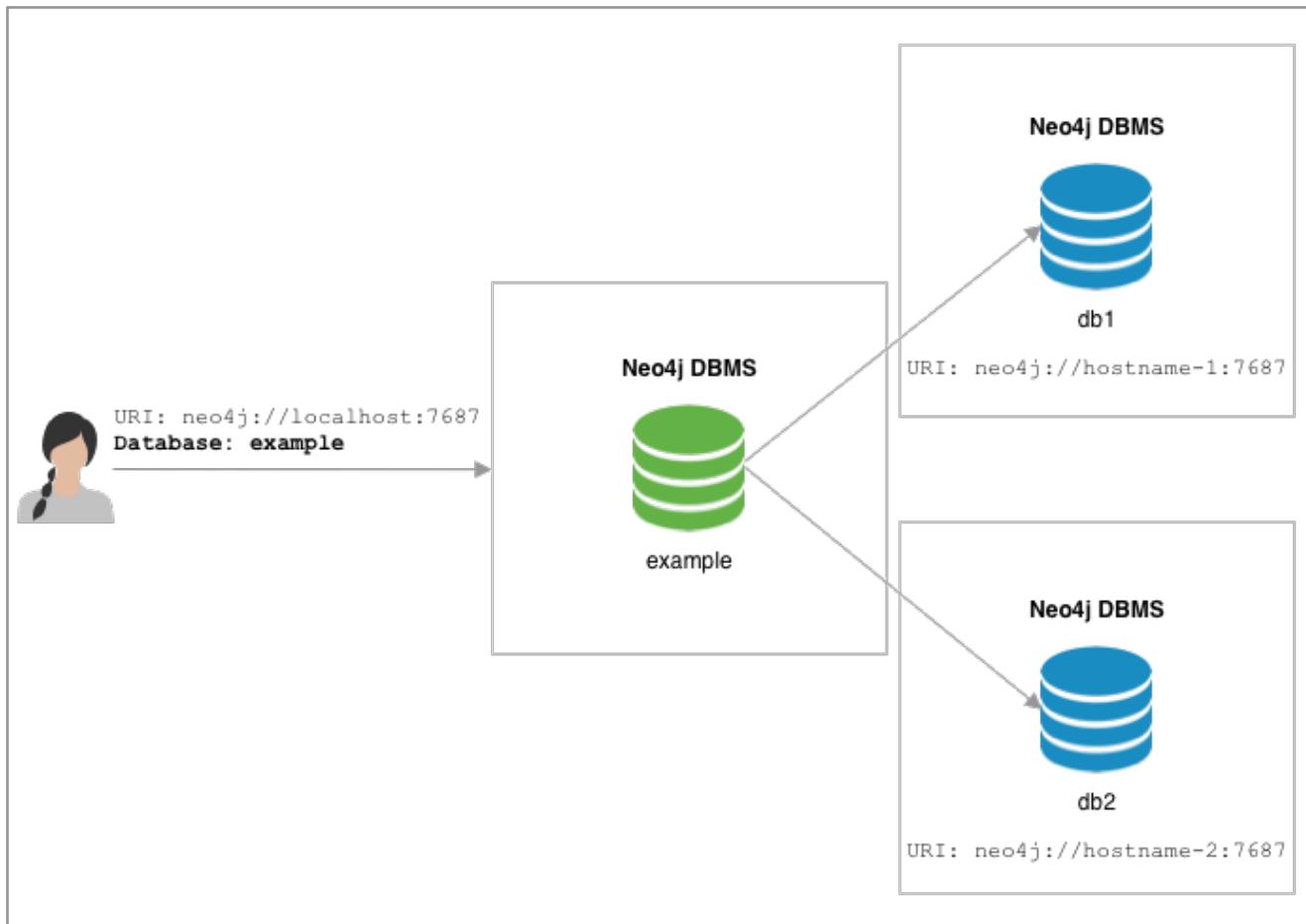


Figure 10. Minimal remote Fabric setting in a development setup

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 we 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, i.e. 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.

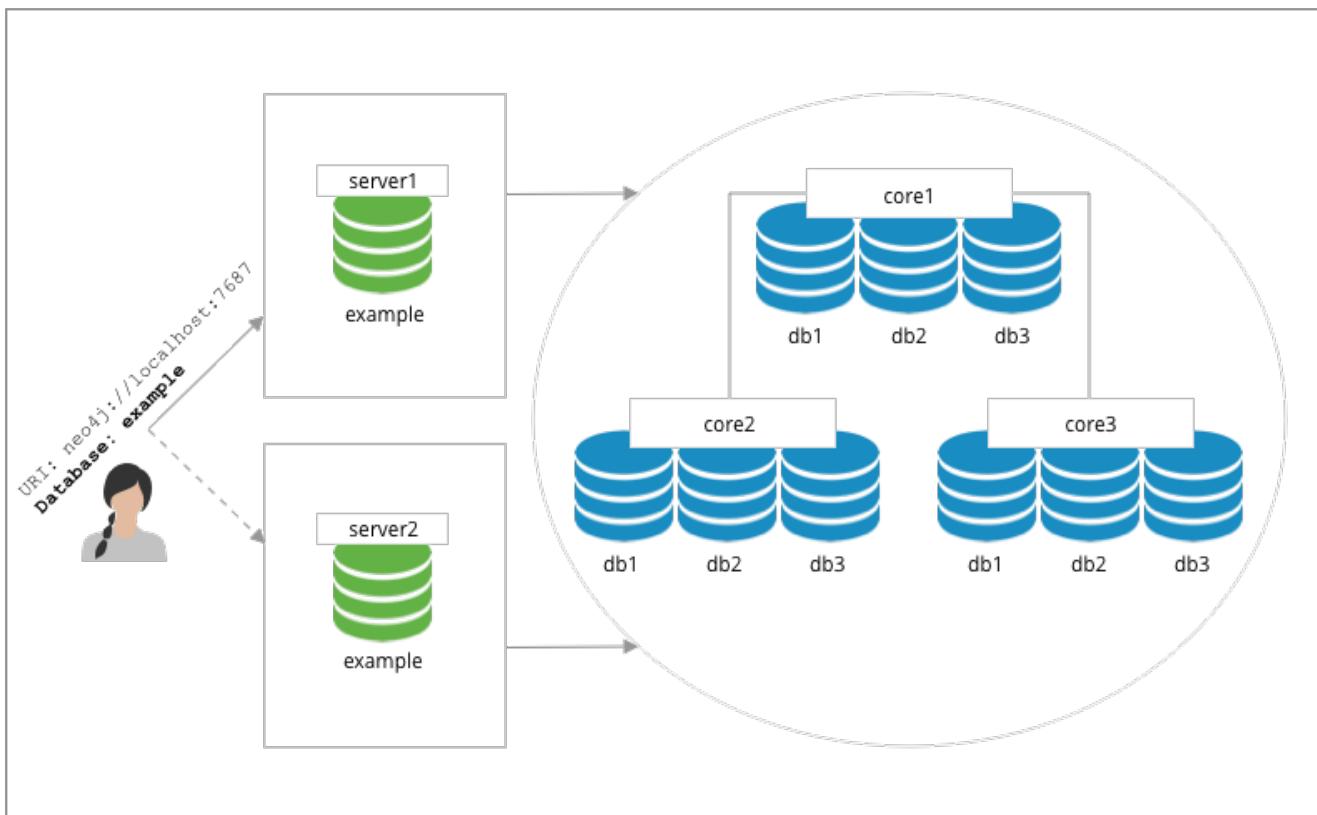


Figure 11. Fabric setting with Causal Cluster and no single point of failure

Cluster routing context

The URIs in the graph settings may include [routing contexts](#). 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 we have a server policy called `read_replicas` defined in the configuration of the cluster we are targeting, we 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.

8.2.2. 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.

8.2.3. Important settings

This section provides general information about Fabric settings and describes the ones important for creating a fabric set-up. Please refer to [\[configuration-settings\]](#) for the full list of Fabric configuration options.

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 31. Fabric system settings

Parameter	Description
<code>fabric.database.name</code>	Name of the Fabric database. Neo4j Fabric currently supports one Fabric database in a standalone Neo4j DBMS.

Parameter	Description
<code>fabric.routing.servers</code>	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: <code>fabric.routing.servers=server1:7687,server2:7687</code> .

Graph settings



The `<ID>` in the following settings is the integer associated to each Fabric graph.

Table 32. Fabric graph settings

Parameter	Description
<code>fabric.graph.<ID>.uri</code>	URI of the Neo4j DBMS hosting the database associated to the Fabric graph. Example: <code>neo4j://somewhere:7687</code>
<code>fabric.graph.<ID>.database</code>	Name of the database associated to the Fabric graph.
<code>fabric.graph.<ID>.name</code>	Name assigned to the Fabric graph. The name can be used in Fabric queries.
<code>fabric.graph.<ID>.driver.*</code>	Any specific driver setting, i.e. 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, please make sure that the remote is configured to advertise its address correctly. This is done through 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 48. 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 33. Fabric drivers setting suffixes

Parameter suffix	Explanation
<code>ssl_enabled</code>	SSL for Fabric drivers is configured using the <code>fabric</code> SSL policy. This setting can be used to instruct the driver not to use SSL even though the <code>fabric</code> SSL policy is configured. The driver will use SSL if the <code>fabric</code> SSL policy is configured, and this setting is set to <code>true</code> . This parameter can only be used in <code>fabric.graph.<graph ID>.driver.ssl_enabled</code> and not <code>fabric.driver.ssl_enabled</code> .
<code>api</code>	Determines which driver API will be used. Supported values are <code>RX</code> and <code>ASYNC</code> .



Most driver options described in [Driver Manual □ Configuration](#) have an equivalent in Fabric configuration.

8.3. Queries

Examples of Cypher queries and commands that can be used with Neo4j Fabric.

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.

8.3.1. Query a single graph

Example 49. 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.

8.3.2. Query multiple graphs

Example 50. 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.

8.3.3. Query all graphs

Example 51. 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.

8.3.4. Query result aggregation

Example 52. 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.

8.3.5. Correlated subquery

Example 53. 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 54. Illegal correlated subquery

```
USE example.graphA  
MATCH (movie:Movie)  
WITH movie.title AS title  
CALL {  
    USE example.graphB // Cannot switch 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.

8.3.6. Updating query

Example 55. 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
```

8.3.7. 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
```

8.3.8. 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 34. Fabric built-in functions

Function	Explanation
<code><fabric database name>.graphIds()</code>	Provides a list of IDs of all remote graph configured for the given Fabric database.
<code><fabric database name>.graph(graphId)</code>	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

8.4. Further considerations

This section presents considerations about Fabric that developers and administrators must be aware of.

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 graphs 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.

Transactions in Fabric

In Fabric, ACID compliance is guaranteed only within a single graph. This means, that the current version of Fabric does not support transactions that span across multiple graphs. To avoid common mistakes that may lead to data corruption, Fabric does not allow write operations on more than one graph within the same transaction. Transactions with queries that read from multiple graphs, or read from multiple graphs and write to a single graph, are allowed.

Administration commands

Fabric does not support issuing 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.

Chapter 9. Backup

This chapter covers how to perform and restore backups of a Neo4j DBMS deployed as a Causal Cluster or a single instance.

This chapter describes the following:

- [Backup planning](#)
 - [Introduction](#)
 - [Online and offline backups](#)
 - [Server configuration](#)
 - [Databases to backup](#)
 - [Storage considerations](#)
 - [Cluster considerations](#)
 - [Using SSL/TLS for backups](#)
 - [Additional files to back up](#)
- [Perform a backup](#)
 - [Backup command](#)
 - [Backup process](#)
 - [Memory configuration](#)
- [Restore a backup](#)
 - [Restore commands](#)
 - [Restore a single database](#)
 - [Restore a cluster](#)

9.1. Backup planning

How to prepare for backing up a Neo4j deployment.

9.1.1. Introduction

Designing an appropriate backup strategy for your Neo4j DBMS is a fundamental part of operations. The backup strategy should take into account elements such as:

- Demands on performance during backup actions.
- Tolerance for data loss in case of failure.
- Tolerance for downtime in case of failure.
- Data volumes.

The backup strategy will answer question such as:

- What type of backup method used; online or offline backups?
- What physical setup meets our demands?
- What backup media — offline or remote storage, cloud storage etc. — should we use?

- How long do we archive backups for?
- With what frequency should we perform backups;

If using online backups:

- How often should we perform full backups?
- How often should we perform incremental backups?

- How do we test recovery routines, and how often?

9.1.2. Online and offline backups

Online backups are typically required for production environments, but it is also possible to perform offline backups.

Offline backups are a more limited method for backing up a database. For example:

- Online backups run against a database in online mode on a live Neo4j instance, while offline backups require that the database is shut down and Neo4j could be either live or shutdown.
- Online backups can be full or incremental, but there is no support for backing up incrementally with offline backups.

For more details about offline backups, see [Dump and load databases](#).

The remainder of this chapter is dedicated to describing *online* backups.

9.1.3. 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) and this needs to be changed if backups are to be taken from another machine.

Table 35. Server parameters for backups

Parameter name	Default value	Description
<code>dbms.backup.enabled</code>	<code>true</code>	Enable support for running online backups.
<code>dbms.backup.listen_address</code>	<code>127.0.0.1:6362</code>	Listening server for online backups.

9.1.4. Databases to backup

Since a Neo4j DBMS can host multiple databases and they are backed up independently of one another, it is important to plan a backup strategy for every database and to not forget any databases. In a new deployment there are two databases by default, `neo4j` and `system`. The system database contains configuration, e.g. operational states of databases, security configuration, etc.

9.1.5. 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 could for example 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.

You may also want to override the settings used for pruning and rotation of transaction log files. The

transaction log files are files that keep track of recent changes. Please note that removing transaction logs manually can result in a broken backup.

Recovered servers do not need all of the transaction log files that have already been applied, so it is possible to reduce storage size even further 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. Alternatively you can use the `--additional-config` override.

9.1.6. Cluster considerations

In a cluster 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 Servers 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 Cluster. If a Read Replica is not available, then a Core can be picked based on factors such as its physical proximity, bandwidth, performance, and liveness.

Note that both Read Replicas and Cores can fall behind the leader and be out-of-date. We can look at transaction IDs in order to avoid taking a backup from a server that has lagged too far behind. The latest transaction ID can be found by [exposing Neo4j metrics](#) or via Neo4j Browser. To view the latest processed transaction ID (and other metrics) in Neo4j Browser, type `:sysinfo` at the prompt.

9.1.7. Using SSL/TLS for backups

The backup server can be configured to require SSL/TLS. If that is the case then the backup client must also be configured to use it with a compatible policy. Refer to [SSL framework](#) to learn how SSL is configured in general. See below table for more details about how configured SSL policies map to the configured ports.

Table 36. Mapping backup configuraton to SSL policies

Backup target address on database server	SSL policy setting on database server	SSL policy setting on backup client	Default port
<code>dbms.backup.listen_address</code>	<code>dbms.ssl.policy.backup</code>	<code>dbms.ssl.policy.backup</code>	6362
<code>causal_clustering.transaction_listen_address</code>	<code>dbms.ssl.policy.cluster</code>	<code>dbms.ssl.policy.backup</code>	6000

9.1.8. Additional files to back up

The files listed below are not included in online nor offline backups. Make sure to back them up separately.

- If you have a cluster, it may be relevant to back up `neo4j.conf` on each server.
- Back up all the files used for SSL/TLS, i.e. private keys, public certificates, 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 on each server in the cluster.

9.2. Perform a backup

How to perform an online backup of a Neo4j database.



Remember to back up all of your created databases, including the `system` database.

9.2.1. Backup command

A Neo4j database can be backed up in online mode using the `backup` command of `neo4j-admin`. The machine that runs the backup command must have Neo4j installed, but does not need to run a Neo4j server. Note that it is **not** recommended to use an NFS mount for backup purposes as this is likely to corrupt and slow down the backup.

Syntax

```
neo4j-admin backup --backup-dir=<path>
  [--verbose=<true/false>]
  [--from=<host:port>]
  [--database=<database>]
  [--fallback-to-full=<true/false>]
  [--pagecache=<size>]
  [--check-consistency=<true/false>]
  [--check-graph=<true/false>]
  [--check-indexes=<true/false>]
  [--check-label-scan-store=<true/false>]
  [--check-property-owners=<true/false>]
  [--report-dir=<path>]
  [--additional-config=<path>]
  [--include-metadata=<all/users/roles>]
```

Options

Option	Default	Description
<code>--backup-dir</code>		Directory to place backup in.
<code>--verbose</code>	<code>false</code>	Enable verbose output.
<code>--from</code>	<code>localhost:6362</code>	Host and port of Neo4j.
<code>--database</code>	<code>neo4j</code>	Name of the database to back up. Name can contain <code>*</code> or <code>?</code> for name globbing. If a backup of the specified database exists in the target directory, then an incremental backup will be attempted.
<code>--fallback-to-full</code>	<code>true</code>	If an incremental backup fails backup will move the old backup to <code><name>.err.<N></code> and fallback to a full backup instead.

Option	Default	Description
--pagecache	8M	The size of the page cache to use for the backup process.
--check-consistency	true	If a consistency check should be made.
--check-graph	true	Perform checks between nodes, relationships, properties, types and tokens.
--check-indexes	true	Perform checks on indexes.
--check-label-scan-store	true	Perform checks on the label scan store.
--check-property-owners	false	Perform additional checks on property ownership. This check is very expensive in time and memory.
--report-dir	.	Directory where consistency report will be written.
--additional-config		Configuration file to supply additional configuration in.
--include-metadata		Include metadata in the backup. Metadata contains security settings related to the database. Can't be used for backing system database. <ul style="list-style-type: none"> • <code>roles</code> - commands to create the roles and privileges (for both database and graph) that affect the use of the database • <code>users</code> - commands to create the users that can use the database and their role assignments • <code>all</code> - include roles and users

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:

```
neo4j-admin backup --from=192.168.1.34 --backup-dir=/mnt/backups/neo4j --database=n* --pagecache=4G
```

Exit codes

`neo4j-admin backup` will exit with different codes depending on success or error. In the case of error, this includes details of what error was encountered.

Table 37. 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 38. Neo4j Admin backup exit codes when backing several databases

Code	Description
0	All databases are backed up successfully.
1	One or several backup failed.

9.2.2. Backup process

The backup client can operate in two slightly different modes referred to as performing a *full backup* or an *incremental backup*. A full backup is always required initially for the very first backup into a target location. Subsequent backups will attempt to use the incremental mode where just the delta of the transaction logs since the last backup are transferred and applied onto the target location. If the required transaction logs aren't available on the backup server then the backup client will fallback to performing a full backup instead, unless `--fallback-to-full` is disabled.

After the backup has been successfully performed the *consistency checker* will be invoked by default. Checking the consistency of the backup is a major operation which can consume significant computational resources, e.g. memory, CPU, I/O.



It is strongly discouraged to run the backup client on a live Neo4j server, especially together with a consistency check. Doing so can adversely affect the server.

To avoid adversely affecting a running server with the resource demands of the backup client it is recommended to take the backup and perform the consistency check on a dedicated machine which has sufficient free resources to perform the consistency check. Another alternative is to decouple the backup operation from the consistency checking and schedule that part of the workflow to happen at a later point in time on a dedicated machine. The value of consistency checking a backup should not be underestimated as it is vital for safe guarding and ensuring the quality of your data.

The transaction log files in the backup are rotated and pruned based on the provided configuration. For example, setting `dbms.tx_log.rotation.retention_policy=3 files` will keep 3 transaction log files in the backup. You can use the `--additional-config` parameter to override this configuration.

Example 56. Full backup

In this example, set environment variables in order to [control memory usage](#).

The page cache is defined by using the command line option `--pagecache`. Further, the `HEAP_SIZE` environment variable will specify the maximum heap size allocated to the backup process.

Now you can perform a full backup:

```
$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
...
...
...
```

If you do a directory listing of `/mnt/backups` you will now see that you have a backup in a directory called `neo4j`.

Example 57. Incremental backup

This example assumes that you have performed a full backup as per the previous example. In the same way as before, make sure to control the memory usage.

To perform an incremental backup you need to specify the location of your previous backup:

```
$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.
```

9.2.3. Memory configuration

The following options are available for configuring the memory allocated to the backup client:

Configure heap size for the backup

This is done by setting the environment variable `HEAP_SIZE` before starting the backup program. If not specified by `HEAP_SIZE`, the Java Virtual Machine will choose a value based on server resources. `HEAP_SIZE` configures the maximum heap size allocated for the backup process.

Configure page cache for the backup

The page cache size can be determined for the backup program by using the `--pagecache` option to the `neo4j-admin backup` command. If not explicitly defined, the page cache will default to 8MB.

9.3. Restore a backup

How to restore from backups of a Neo4j deployment.

9.3.1. Restore command

A Neo4j database can be restored using the `restore` command of `neo4j-admin`.



Restoring a database does not automatically cause it to be created by the Neo4j DBMS.

That configuration lives in the `system` database and if a backup of the `system` database is restored, then any configuration for other databases in it will also be restored. If the restored database has not been created yet, use the `CREATE DATABASE` command after restoring, to create it. See [Administrative commands](#) for more information.

If you have backed up database which includes `--include-metadata`, you will need to restore users and roles metadata manually on the core member. To do this you need to run `data/scripts/databasename/restore_metadata.cypher` using [Cypher Shell](#):

Using `cat` (UNIX)

```
$neo4j-home> cat data/scripts/databasename/restore_metadata.cypher | ./cypher-shell -u user -p password -a core_member -d system --param "database => 'databasename'"
```

Using `type` (Windows)

```
$neo4j-home> type data\scripts\databasename\restore_metadata.cypher | bin\cypher-shell.bat -u user -p password -a core_member -d system --param "database => 'databasename'"
```

For more information, see [Cypher Shell](#).

Syntax

```
neo4j-admin restore --from=<path,path>
  [--verbose=<true/false>]
  [--database=<database>]
  [--force]
  [--move]
  [--to-data-directory=<path>]
  [--to-data-tx-directory=<path>]
```

Options

Option	Default	Description
<code>--from</code>		Path, or paths, from which to restore. Every path can contain asterisks or question marks in the last subpath. Multiple paths may be separated by a comma, but paths themselves must not contain commas.
<code>--verbose</code>	<code>false</code>	Enable verbose output.

Option	Default	Description
--database		Name of the database after restore. Usage of this option is only allowed if the <code>--from</code> parameter points to exact one directory.
--force		If an existing database should be replaced.
--move		Moves the backup files to the destination, rather than copying.
--to-data-directory		Base directory for databases. Usage of this option is only allowed if the <code>--from</code> parameter points to exact one directory.
--to-data-tx-directory		Base directory for transaction logs. Usage of this option is only allowed if the <code>--from</code> parameter points to exact one directory.

9.3.2. Restore a standalone server

To restore from backups, follow these steps:

1. If the server is running, shut it down.
2. Run `neo4j-admin restore` for every database.
3. Start up the server.

Example 58. Restore a standalone server

Restore the databases `system` and `neo4j` from the backups located in `/mnt/backups`.

```
neo4j-home> bin/neo4j stop
neo4j-home> bin/neo4j-admin restore --from=/mnt/backups/system --database=system --force
neo4j-home> bin/neo4j-admin restore --from=/mnt/backups/neo4j --database=neo4j --force
neo4j-home> bin/neo4j start
```

9.3.3. Restore a cluster

To restore a Causal Cluster from backups, follow these steps:

1. Shut down all server instances in the cluster.
2. Run the `neo4j-admin unbind` command on each of the Core Servers.

3. Restore the backups on each instance, using `neo4j-admin restore`.
4. If you are restoring onto new hardware, please review the *Causal Clustering* settings in `neo4j.conf`.

In particular, check the settings `causal_clustering.initial_discovery_members`, `causal_clustering.minimum_core_cluster_size_atFormation`, and `causal_clustering.minimum_core_cluster_size_atRuntime`, and ensure that they correctly reflect the new setup.

5. Start the server instances.

Refer to [Seed a cluster](#) for more detailed information.



When restoring a backup on a Core Server where a database of the same name already exists, you must execute `DROP DATABASE` first. If you are unable to (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.

Chapter 10. Upgrade

This chapter describes how to upgrade Neo4j from an earlier minor or patch version.

It covers the following topics:

- [Plan your upgrade](#)
 - [Important information](#)
 - [Upgrade checklist](#)
- [Upgrade a single instance deployment](#)
- [Upgrade a Causal Cluster](#)
 - [Offline upgrade](#)
 - [Rolling upgrade](#)

10.1. Plan your upgrade

How to prepare for upgrading your Neo4j deployment.

10.1.1. Important information

Upgrading your Neo4j deployment ensures that you are provided with the latest improvements in performance, security, and bug fixes.

Understanding upgrades and migration

The following are methods of keeping your Neo4j deployment up-to-date:

Upgrade

The process of upgrading an existing Neo4j deployment (single or clustered) to a newer Neo4j version, when such process **does not require** changes to the configuration or to the applications that use Neo4j.

- Between patches, for example, from 4.0.3 to 4.0.4.
- Between minor versions, for example from 4.0.4 to 4.1.0.

What needs to be upgraded

- The Neo4j product to a later version of Neo4j.
- (Optional) The store formats (the file formats on disk, which usually auto-upgrade at server startup, but sometimes an older version may remain on a running server).
- The **system** database schema, which restructures the contents of that database. This is done automatically on standalone server and in static cluster upgrades (where you temporarily set each server as standalone), but need to be run manually on rolling upgrades after the other two upgrades (product and stores) have been done.

Migration

The process of upgrading an existing Neo4j deployment (single or clustered) to a newer Neo4j version, when such process **requires** a review of the configuration(s) and the applications that use Neo4j.

- Between major versions, for example, from 3.5.13 to 4.0.7.

What needs to be upgraded and migrated

- The Neo4j product to a later major version of Neo4j.
- Store formats - same as in upgrade.
- The `system` database schema - same as in upgrade.
- Configuration settings - the configuration changes from the newer version must be applied to the old configuration file.
- Application code - the source code of your application(s) must be updated as per the new version in order to work properly.
- Custom plugins - all custom plugins must be compatible with the new version of Neo4j.



For more information, see [Neo4j Migration Guide](#).

Supported upgrade paths

The following upgrade paths are supported:

- From one version to the next adjacent version, for example 4.0.0 to 4.0.1.
- From one version to a non-adjacent release, for example, from 4.0.1 to 4.0.3, or from 4.0.2 to 4.1.0.
 - If you want to migrate from a 3.5.any to 4.0.any or 4.2 version of Neo4j, follow the instructions in the [Neo4j Migration Guide](#).
 - If you want to upgrade to version 3.5.latest, follow the instructions in the [Neo4j Operations Manual for 3.5](#).



Limitations

- Neo4j does not support downgrades. If the upgrade is not successful, you have to do a full rollback, including restoring a pre-upgrade backup.
- A Neo4j upgrade must be performed as an isolated operation. If you are planning to upgrade from a single-instance installation to a Causal Cluster, this must be performed separately from the upgrade to 4.2.
- In order to further minimize risk, it is recommended that while upgrading, you do not switch from Community Edition to Enterprise Edition, change configuration, perform architectural restructuring, database administration, or similar tasks.

10.1.2. Upgrade checklist

The following upgrade checklist provides a high-level overview of the tasks you have to perform:

- Complete all [prerequisites](#) for the upgrade.
- [Back up your current deployment](#) to avoid losing data in case of a failure.
- Download [the new version of Neo4j](#). Make sure [the upgrade path is supported](#).
- [Prepare the new `neo4j.conf` file](#) to be used by the new installation.
- [Perform a test upgrade](#).
- [Monitor the logs](#).
- Perform the upgrade as per your deployment type ([Single instance](#) or [Causal Cluster](#)).



If you are upgrading a Causal Cluster, do the tasks for each member of the cluster.

- Verify that you have installed Java 11.
- Review the improvements and fixes that have been carried out in the version that you want to upgrade to. See the [Neo4j 4.1 Change log](#).
- Verify that you have [planned your Neo4j upgrade](#).
- Verify that you have done all the tasks from the [Upgrade checklist](#).

Back up your current deployment

It is recommended to perform and verify the following backups, to avoid losing data in case of a failure.

- Back up [*neo4j.conf*](#).
- Back up 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](#).
- Back up your databases by using `neo4j-admin dump` and store it in a safe location.
- (Optional/Recommended) If using custom plugins, make sure that you have the plugins in an accessible location.

Prepare a new *neo4j.conf* file

Even though a Neo4j upgrade does not require changes to the configuration, you still have to prepare the new *neo4j.conf* file to be used by the new deployment.

- Update the new *neo4j.conf* file with any non-default settings from your old installation.
- When upgrading, it is particularly important to note any custom values of the settings `dbms.directories.data` and `dbms.default_database`.
- In cluster installations, pay attention to cluster-specific configuration settings, which might be different for the different cluster members.

Perform a test upgrade

Based on the findings in this chapter, allocate a staging test environment for the upgrade and do a test upgrade. The test upgrade will give you valuable information about the time required for the production upgrade. Follow the steps as per your deployment type. For more information, see [Upgrade a single instance](#) or [Upgrade a Causal Cluster](#).

Monitor the logs

The [*neo4j.log*](#) file contains information on how many steps the upgrade will involve and how far it has progressed. It is a good idea to monitor this log.

Update metrics

The metrics that are enabled by default have been changed in the 4.2 release.

Any specific metrics that you want to be enabled **must** be specified in the `metrics.filter`.

Additionally in the 4.2 release, metrics are no longer exposed via JMX by default. These can be enabled by adding `metrics.jmx.enabled=true` to *neo4j.conf*.

For more information, see [Enable metrics logging](#).

10.2. Upgrade a single instance

This section describes how to upgrade a single Neo4j instance.

This section describes the following:

- [Prerequisites](#)
- [Upgrade Neo4j using the tarball or ZIP file distribution](#)
- [Upgrade Neo4j using the Debian or RPM distribution](#)
- [Post-upgrade tasks](#)

10.2.1. Prerequisites

- Verify that you have installed Java 11.
- Review the improvements and fixes that have been carried out in the version that you want to upgrade to. See the [Neo4j 4.1 Change log](#).
- Verify that you have [planned your Neo4j upgrade](#).
- Verify that you have done all the tasks from the [Upgrade checklist](#).

10.2.2. Upgrade Neo4j using the tarball or ZIP file distribution

1. Untar or unzip Neo4j 4.2.0.
2. Place the new *neo4j.conf* that you have prepared during the [upgrade planning](#) in the *Configuration* directory.
3. Open the *neo4j.conf* file of the new installation and configure the following settings:
 - Uncomment `dbms.allow_upgrade=true` to allow automatic store upgrade. Neo4j will fail to start without this configuration.
 - Configure `dbms.mode` to `SINGLE` and add the following setting `dbms.allow_single_automatic_upgrade=true` to allow automatic upgrade of the system database schema.
4. If using [role-based access control](#), copy the *data/dbms* directory into the new location.
5. Copy the files used for encryption from the old installation to the new one.
6. Use the command `neo4j-admin load --from=<archive-path> --database=<database> [--force]` to load your data dump. For more information, see [Dump and load databases](#).
7. If using custom plugins, place the plugins in the */plugins* directory.
8. Run `./bin/neo4j start` to start Neo4j 4.2.0. The database upgrade takes place during startup.
9. Monitor the *neo4j.log* file for information on how many steps the upgrade involves and how far it has progressed.

10.2.3. Upgrade Neo4j using the Debian or RPM distribution

1. Open the *neo4j.conf* file of the new installation and configure the following settings:
 - Uncomment `dbms.allow_upgrade=true` to allow automatic store upgrade. Neo4j will fail to start without this configuration.
 - Configure `dbms.mode` to `SINGLE` and add the following setting `dbms.allow_single_automatic_upgrade=true` to allow automatic upgrade of the system database schema.

2. Install Neo4j 4.2.0. See [Installation](#).
3. When prompted, review the differences between the `neo4j.conf` files of the previous version and Neo4j 4.2.0. Transfer any custom settings to the 4.2.0 installation that you have prepared during the [upgrade planning](#). Make sure to preserve the settings as set in step 1.
4. If using custom plugins, place the plugins in the `/plugins` directory.
5. Run `./bin/neo4j start` to start Neo4j 4.2.0. The database upgrade takes place during startup.
6. Monitor the `neo4j.log` file for information on how many steps the upgrade involves and how far it has progressed.

10.2.4. Post-upgrade tasks

1. After the upgrade finishes, open the `neo4j.conf` file, and set `dbms.allow_upgrade` to `false`. If, for some reason, you have forgotten to enable the automatic upgrade of the system database schema, use the following commands to manually upgrade it:
 - `CALL dbms.upgradeStatus()` to determine whether an upgrade is necessary or not.
 - `CALL dbms.upgrade()` on the `system` database to perform the upgrade of the system schema.

For more details, see [Procedures](#).

2. Run `./bin/neo4j restart` to restart the database.
3. It is a good practice to make a full backup immediately after the upgrade.
4. Check the default settings on any metrics. In the 4.2 release there are new default settings for metrics. Any specific metrics that you want to be enabled **must** be specified in the `metrics.filter`.
For more information, see [Enable metrics logging](#).

10.3. Upgrade a Causal Cluster

How to upgrade a Neo4j Causal Cluster.

You can upgrade your existing Neo4j Causal Cluster by either performing a rolling upgrade, or by upgrading it offline.



The prerequisites and the upgrade steps must be completed for each cluster member.

10.3.1. Offline upgrade

This variant is suitable for cases where a rolling upgrade is not possible.



It is recommended to perform a test upgrade on a production-like environment to get information on the duration of the downtime.

Prerequisites

- Verify that you have installed Java 11.
- Review the improvements and fixes that have been carried out in the version that you want to upgrade to. See the [Neo4j 4.1 Change log](#).
- Verify that you have [planned your Neo4j upgrade](#).
- Verify that you have done all the tasks from the [Upgrade checklist](#).

- Shut down all the cluster members.
- Install *Neo4j* on each instance using one of the following procedures:

Install Neo4j using a tarball or ZIP file

1. Untar or unzip the version of *Neo4j* that you want to upgrade to.
2. Place the *neo4j.conf* file, that you have prepared for this instance, into the *Configuration* directory.
3. If using [native user and role management](#), copy the *data/dbms* directory into the new location.
4. Copy the files used for encryption from the old installation to the new one.
5. Copy the *data* directory from the old installation to the new one.



The previous step may not be enough if you have configured store files to be situated outside the default configuration of *NEO4J_HOME*. If you have modified configurations starting with *dbms.directories.**, then you need to make sure that the new installation is configured properly to find these directories.

6. If using custom plugins, place the plugins that are adjusted for the new version in the */plugins* directory.

Install Neo4j using a Debian or RPM distribution

1. Install the version of *Neo4j* that you want to upgrade to. See [Installation](#).
2. When prompted, review the differences between the *neo4j.conf* files of the previous version and the new version of *Neo4j*. Transfer any custom settings to the new installation, as prepared during the [upgrade planning](#).

Upgrade steps

On one of the Cores

1. Perform `neo4j-admin unbind`.
2. Open the *neo4j.conf* file of the new installation and configure the following settings:
 - Uncomment `dbms.allow_upgrade=true` to allow automatic store upgrade.
 - Configure `dbms.mode` to `SINGLE` and add the following setting `dbms.allow_single_automatic_upgrade=true` to allow automatic upgrade of the `system` database schema.
3. Start the server. The upgrade takes place during startup.
4. Monitor the *neo4j.log* file for information on how many steps the upgrade will involve and how far it has progressed.
5. When the upgrade finishes, stop the server.
6. Set `dbms.allow_upgrade=false`.
7. Set `dbms.mode=CORE` in *neo4j.conf* to re-enable Causal Clustering in the configuration.
8. Use `neo4j-admin dump` to make a dump of all your databases and transactions, including the `system` database.
9. Do **not** yet start the server.

On each of the other Cores

1. Perform `neo4j-admin unbind`.

2. Delete the contents of the `/databases` and the `/transactions` directories, which are located in the `data` directory. Make sure those directories exist.
3. Use `neo4j-admin load` to restore all your upgraded databases and transactions onto this server, including the `system` database.
4. Repeat the steps on each of the remaining Cores.
5. Start each of the servers and verify that they join the cluster.

On each of the Read Replicas

1. Perform `neo4j-admin unbind`.
2. Delete the contents of the `/databases` and the `/transactions` directories, which are located in the `data` directory. Make sure those directories exist.
3. Load the data by either
 - a. Using `neo4j-admin load` to restore all your upgraded databases and transactions onto this server, including the `system` database.
 - b. Starting the Read Replica and waiting for it to do a complete store copy.
4. Start each of the servers if you have loaded your data with `neo4j-admin load`.
5. Verify that they join the cluster.



It is a good practice to make a full backup immediately after the upgrade.

10.3.2. Rolling upgrade

Rolling upgrade is a zero-downtime method for upgrading a Causal Cluster. You upgrade one member at a time while the rest of the members are running. However, if during a rolling upgrade the cluster loses quorum and can not be recovered, then downtime may be required to do a disaster recovery.

Recommendations

- Replacing cores may reduce the cluster fault tolerance level. It is recommended to add any new members before you shut down the old ones. This way, the cluster fault tolerance level will be preserved.
-
- If the system does not allow adding more members, then the cluster fault tolerance level will be reduced while replacing members.
- The critical point during the upgrades is knowing when it is safe to switch off the original member.
It is highly recommended to monitor the `status endpoint` before each removal to decide which member to switch off, and when it is safe to do so.
 - To reduce the risk of failure during a rolling upgrade it is recommended that the cluster is not under any heavy load during the upgrade. If possible, the safest would be to disable writes entirely.
 - There should be no changes to database administration during a rolling upgrade. For more information, see [Manage databases](#).

Rolling upgrade for fixed servers

This variant is suitable for deployments where the servers are fixed and have to be updated in-place.



When performing a rolling upgrade for fixed servers, it is not possible to increase the cluster size.

Prerequisites

- Verify that you have installed Java 11.
- Review the improvements and fixes that have been carried out in the version that you want to upgrade to. See the [Neo4j 4.1 Change log](#).
- Verify that you have [planned your Neo4j upgrade](#).
- Verify that you have done all the tasks from the [Upgrade checklist](#).
- (Recommended) Ensure all databases are in online mode. Any databases that are in offline mode can be started using `START DATABASE [database-name]`. You can verify the status of available databases using `SHOW DATABASES`.
 - If you need to keep the database inaccessible during the rolling upgrade, you can disable access to it. For example:

```
DENY ACCESS ON DATABASE [database-name] TO PUBLIC
```

When upgrading from 4.0.x you will have to disable access to each role, as the `PUBLIC` will not yet exist:

```
DENY ACCESS ON DATABASE [database-name] TO [role1],[role2]
```

This has to be done for each role; the available roles can be queried with `SHOW ROLES`.

- You may want to deny databases from being stopped, created, or dropped during the rolling upgrade. For example:

```
DENY STOP ON DATABASE * TO PUBLIC  
DENY DATABASE MANAGEMENT ON DBMS TO PUBLIC
```

When upgrading from 4.0.x, you can only disable the ability to stop databases for the `admin` roles:

```
DENY STOP ON DATABASE * TO admin
```

- (Optional/Recommended) Use the process described in the [status endpoint](#) to evaluate whether it is safe to remove the old instance.
- Shut down the instance.
- Install *Neo4j* on each instance using one of the following procedures:

Install Neo4j using a tarball or ZIP file

1. Untar or unzip the version of *Neo4j* that you want to upgrade to.
2. Place the `neo4j.conf` file, that you have prepared for this instance, into the [Configuration](#) directory.
3. If using [native user and role management](#), copy the `data/dbms` directory into the new location.
4. Copy the files used for encryption from the old installation to the new one.
5. Copy the `data` directory from the old installation to the new one.



The previous step may not be enough if you have configured store files to be situated outside the default configuration of `NEO4J_HOME`. If you have modified configurations starting with `dbms.directories.*`, then you need to make sure that the new installation is configured properly to find these directories.

6. If using custom plugins, place the plugins that are adjusted for the new version in the `/plugins` directory.

Install Neo4j using a Debian or RPM distribution

1. Install the version of `Neo4j` that you want to upgrade to. See [Installation](#).
2. When prompted, review the differences between the `neo4j.conf` files of the previous version and the new version of `Neo4j`. Transfer any custom settings to the new installation, as prepared during the [upgrade planning](#).

Upgrade steps

1. Run `./bin/neo4j start` to start the new instance, and let it complete a store copy.



Please note that you can not run a store copy on a stopped database. To ensure that any stopped databases have successfully been transferred to the new server, it is recommended that all databases are in online mode before performing a rolling upgrade.

2. Verify that the new instance joins the cluster by using the [status endpoint](#).
3. Repeat the upgrade steps for each Core server.
4. Repeat the upgrade steps for each Read Replica server.



Because Read Replicas are not part of the cluster consensus group, their replacement during an upgrade does not affect the cluster availability and fault tolerance level. However, it is still recommended to incrementally add Read Replicas for a structured and maintainable upgrade process.

Rolling upgrade for replaceable resources

This variant is suitable for deployments that use replaceable cloud or container resources. With this method, it is possible to increase the cluster size and avoid reducing the cluster fault tolerance.

Prerequisites

- Verify that you have installed Java 11.
- Review the improvements and fixes that have been carried out in the version that you want to upgrade to. See the [Neo4j 4.1 Change log](#).
- Verify that you have [planned your Neo4j upgrade](#).
- Verify that you have done all the tasks from the [Upgrade checklist](#).
- (Recommended) Ensure all databases are in online mode. Any databases that are in offline mode can be started using `START DATABASE [database-name]`. You can verify the status of available databases using `SHOW DATABASES`.
 - If you need to keep the database inaccessible during the rolling upgrade, you can disable access to it. For example:

```
DENY ACCESS ON DATABASE [database-name] TO PUBLIC
```

When upgrading from 4.0.x you will have to disable access to each role, as the `PUBLIC` will not yet exist:

```
DENY ACCESS ON DATABASE [database-name] TO [role1],[role2]
```

This has to be done for each role; the available roles can be queried with `SHOW ROLES`.

- You may want to deny databases from being stopped, created, or dropped during the rolling upgrade. For example:

```
DENY STOP ON DATABASE * TO PUBLIC  
DENY DATABASE MANAGEMENT ON DBMS TO PUBLIC
```

When upgrading from 4.0.x, you can only disable the ability to stop databases for the `admin` roles:

```
DENY STOP ON DATABASE * TO admin
```

- Install *Neo4j* on each instance using one of the following procedures:

Install Neo4j using a tarball or ZIP file

1. Untar or unzip the version of *Neo4j* that you want to upgrade to.
2. Place the `neo4j.conf` file, that you have prepared for this instance, into the *Configuration* directory.
3. If using *native user and role management*, copy the `data/dbms` directory into the new location.
4. Copy the files used for encryption from the old installation to the new one.
5. Copy the `data` directory from the old installation to the new one.

 The previous step may not be enough if you have configured store files to be situated outside the default configuration of `NEO4J_HOME`. If you have modified configurations starting with `dbms.directories.*`, then you need to make sure that the new installation is configured properly to find these directories.

6. If using custom plugins, place the plugins that are adjusted for the new version in the `/plugins` directory.

Install Neo4j using a Debian or RPM distribution

1. Install the version of *Neo4j* that you want to upgrade to. See [Installation](#).
2. When prompted, review the differences between the `neo4j.conf` files of the previous version and the new version of *Neo4j*. Transfer any custom settings to the new installation, as prepared during the [upgrade planning](#).

Upgrade steps

1. Run `./bin/neo4j start` to start the new instance, and let it complete a store copy.



Please note that you can not run a store copy on a stopped database. To ensure that any stopped databases have successfully been transferred to the new server, it is recommended that all databases are in online mode before performing a rolling upgrade.

2. Verify that the new instance joins the cluster by using the [status endpoint](#).
3. (Optional/Recommended) Use the process described in the [status endpoint](#) to evaluate whether it is safe to remove the old instance.
4. Shut down the old instance.
5. Repeat the upgrade steps for each Core server.
6. Repeat the upgrade steps for each Read Replica server.



Because Read Replicas are not part of the cluster consensus group, their replacement during an upgrade does not affect the cluster availability and fault tolerance level. However, it is still recommended to incrementally add Read Replicas for a structured and maintainable upgrade process.

10.3.3. Post-upgrade

In 4.x versions, Neo4j uses a common `system` database with data and schema that must be upgraded as well. In single deployments and offline upgrades, this is done automatically. However, when performing rolling upgrades, you have to run this upgrade manually. This extra complexity requires a level of *compatibility* and *synchronization* among the members of the cluster.

The metrics that are enabled by default have been changed in the 4.2 release.



Any specific metrics that you want to be enabled **must** be specified in the `metrics.filter`.

Additionally in the 4.2 release, metrics are no longer exposed via JMX by default. These can be enabled by adding `metrics.jmx.enabled=true` to [`neo4j.conf`](#).

For more information, see [Enable metrics logging](#).

Restore to the original state

If you performed a rolling upgrade, you may have started databases that had been stopped, and denied some access rights during the preparation phase. You should now restore those to their original state.

- For rolling upgrades from 4.2:

1. To restore the privilege to stop databases:

```
REVOKE DENY STOP ON DATABASE * FROM PUBLIC
```

2. To restore the privilege to create and drop databases:

```
REVOKE DENY DATABASE MANAGEMENT ON DBMS FROM PUBLIC
```

3. To stop the database:

```
STOP DATABASE [database-name]
```

4. To re-enable access to the database:

```
REVOKE DENY ACCESS ON DATABASE [database-name] FROM [role1],[role2]
```

- For rolling upgrades from 4.0.x:

1. To enable the privilege to stop databases:

```
REVOKE DENY STOP ON DATABASE * FROM admin
```

2. To stop the database:

```
STOP DATABASE [database-name]
```

3. To re-enable access to the database:

```
REVOKE DENY ACCESS ON DATABASE [database-name] FROM [role1],[role2]
```

Upgrade the shared state

In Neo4j 4.x, a cluster-wide shared state is maintained in the `system` database. It includes complex information like the security configuration for users, roles, and their privileges. The structure of the shared graph in the `system` database changes with each release as the capabilities of the DBMS grow. For example, Neo4j 4.1 introduces more fine-grained control over `WRITE` and administrative DBMS privileges.

However, a Neo4j 4.2 server can continue to run with a `system` graph that is still structured according to the Neo4j 4.0 schema. The primary point of this is to support rolling upgrades. With a causal cluster of many instances, while upgrading each instance in turn, there will be a period during which the cluster is composed of some old and some new instances. Since the `system` database is transactionally maintained across the entire cluster, it is necessary that it remains structured according to the older structure until the rolling upgrade process has been completed.

When is this needed?

A few conditions lead to the `system` database containing schema from an older version:

- When upgrading a single instance of Neo4j or performing *Offline upgrades* of a cluster, which involves setting `dbms.mode=SINGLE`, the default behaviour is to automatically upgrade the schema of the `system` graph during server startup. However, it is possible to set `dbms.allow_single_automatic_upgrade=false`, which disables the automatic upgrade.
- When running a rolling upgrade, each instance starts with a `dbms.mode` that is not `SINGLE` and the automatic upgrade is forcibly disabled.
- The DBMS investigates the contents of the `system` graph and detects whether the schema is current or old.

When the schema is not current, but represents the schema of the previous minor or patch version, the server can continue to run in a *compatibility mode*. This allows all compatible capabilities shared between the old and new versions to still work. However, newer features that require the newer schema will not work. For example, if you attempt to grant a new privilege not supported in the 4.0

schema, you will receive an error and the grant will fail. It is necessary to upgrade the `system` schema before all new features are supported. One option to reduce risk is to avoid performing updating commands against the `system` database until the schema has been upgraded.

If the schema in the `system` graph is too old to allow *compatibility mode*, the server will not be able to start. Read more about this in [Troubleshooting](#).

How to upgrade the `system` database schema

The DBMS provides two procedures for upgrading the `system` graph:

- `CALL dbms.upgradeStatus()` to determine whether an upgrade is necessary or not.
- `CALL dbms.upgrade()` on the leader of the `system` database to perform the upgrade of the `system` schema.

```
CALL dbms.upgradeStatus();
```

```
+-----+  
| status | description |  
+-----+  
| "REQUIRES_UPGRADE" | "The sub-graph is supported, but is an older version and requires upgrade" |  
"CALL dbms.upgrade()" |  
+-----+
```

When upgrading between two consecutive minor versions of Neo4j, calling `dbms.upgradeStatus()` returns one of the following two status results:

- **CURRENT**
 - The sub-graph is already at the current version.
 - There is no action to take.
- **REQUIRES_UPGRADE**
 - The sub-graph is supported, but is an older version and requires upgrade.
 - The action to take is: `CALL dbms.upgrade()`.

The full list of possible status values is provided in [Status codes for dbms.upgradeStatus](#).

In order to upgrade the `system` graph, call the procedure `dbms.upgrade()`. If the `system` graph is already at the current version this procedure will have no effect. If the `system` graph requires an upgrade, the procedure will perform the upgrade.

```
CALL dbms.upgrade();
```

```
+-----+  
| status | upgradeResult |  
+-----+  
| "CURRENT" | "Success" |  
+-----+
```

If the upgrade fails for some reason, the status will not change, and the `upgradeResult` field will describe which components have failed to upgrade.



It is a good practice to make a full backup immediately after the upgrade.

Troubleshooting

Some possible problem scenarios:

Calling `dbms.upgrade` when not all instances have been upgraded to the new version of Neo4j

- Calling `dbms.upgradeStatus` on any instances still running the older version returns `UNSUPPORTED_FUTURE`.
- These older instances are no longer able to authorize users.
- All queries against these instances result in an error.

Solution

Shut down, upgrade, and restart the instance in order to regain service.

Trying to upgrade between incompatible versions, for example migrating from Neo4j 3.5 to Neo4j 4.1

When the detected version of the `system` database is too old, the status code is one of the following:

- `UNSUPPORTED_BUT_CAN_UPGRADE` - the server shuts down.

Solution

Use an offline upgrade strategy (set `dbms.allow_single_automatic_upgrade=true`)

- `UNSUPPORTED` - the server shuts down.

Solution

No direct upgrade is possible, upgrade first to a supported intermediate version.

Calling `dbms.upgrade` results in a failure

The `upgradeResult` field describes which components have failed to upgrade.

Solution

To troubleshoot the root cause there are two things to try:

- Try the more verbose version of the command `dbms.upgradeDetails`, which returns a message with the underlying exception thrown. This might not be the root cause, but can help narrow it down.
- Look for descriptive warnings and errors in the `security.log` and the `debug.log`. Stack traces should help track down the root cause of the problem.

Status codes for `dbms.upgradeStatus`

Table 39. System component status

Status	Description	Resolution
<code>UNINITIALIZED</code>	No sub-graph detected for this component.	Requires initialization.
<code>CURRENT</code>	The sub-graph is already at the current version.	No action required.
<code>REQUIRES_UPGRADE</code>	The sub-graph is supported but is an older version and requires upgrade.	Run <code>CALL dbms.upgrade()</code> .
<code>UNSUPPORTED_BUT_CAN_UPGRADE</code>	The sub-graph is unsupported; this component cannot function.	Restart Neo4j in a single-instance mode to upgrade this component at startup.

Status	Description	Resolution
UNSUPPORTED	The sub-graph is unsupported because it is too old; this component cannot function.	Downgrade Neo4j and then upgrade this component before upgrading Neo4j again.
UNSUPPORTED_FUTURE	The sub-graph is unsupported because it is a newer version; this component cannot function.	Upgrade Neo4j.

Chapter 11. Authentication and authorization

This chapter describes authentication and authorization in Neo4j.

Ensure that your Neo4j deployment adheres to your company's information security guidelines by setting up the appropriate authentication and authorization rules.

This section describes the following:

- [Introduction](#)
- [Built-in roles](#)
- [Fine-grained access control](#)
- [Integration with LDAP directory 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.

11.1. Introduction

This section provides an overview of authentication and authorization in Neo4j.

Authentication is the process of ensuring that a user is who the user claims to be, while authorization pertains to checking whether the authenticated user is allowed to perform a certain action. 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. This option is controlled by the parameter `dbms.security.auth_enabled`, which is set to `true` by default. The Cypher commands to manage users, roles and permissions are described in detail in [Cypher Manual](#) \square [Administration](#). Various scenarios that describe 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](#).

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](#) \square [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](#).

11.2. Built-in roles

This section describes the roles that come pre-defined with Neo4j.

Neo4j provides the following native roles:

PUBLIC

- Access to the default database.
- Allows executing procedures with the users own privileges.
- Allows executing user-defined functions with the users own privileges.

reader

- Traverse and read access to the data graph (all nodes, relationships, properties).

editor

- Traverse, read, and write access to 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

- Traverse, read, and write access to the data graph.

architect

- Traverse, read, and write access to the data graph.
- Access to create/drop indexes and constraints along with any other future schema constructs.

admin

- Traverse, read, and write access to the data graph.
- Access to create/drop indexes and constraints along with any other future schema constructs.
- 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.

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 40. Native roles overview

Action	reader	editor	publisher	architect	admin	(no role)	Available in Community Edition
Change own password	X	X	X	X	X	X	X
View own details	X	X	X	X	X	X	X
Read data	X	X	X	X	X		X
View own queries	X	X	X	X	X		
Terminate own queries	X	X	X	X	X		
Write/update /delete existing data		X	X	X	X		X
Create new types of properties key			X	X	X		X
Create new types of nodes labels			X	X	X		X
Create new types of relationship types			X	X	X		X
Create/drop index/constraint				X	X		X
Create/delete user					X		X
Change another user's password					X		
Assign/remove role to/from user					X		
Suspend/activate user					X		
View all users					X		X
View all roles					X		
View all roles for a user					X		
View all users for a role					X		
View all queries					X		
Terminate all queries					X		
Dynamically change configuration (see Dynamic settings)					X		

11.3. Fine-grained access control

Describes an example that illustrates various aspects of security and fine-grained access control.

11.3.1. The data model

Consider a *healthcare* database, as could be relevant in a medical clinic or hospital. A simple version of this might contain only three labels, representing three entity types:

(:Patient)

Nodes of this type represent patients that visit the clinic because they have some symptoms. Information specific to the patient can be captured in properties:

- name
- ssn
- address
- dateOfBirth

(:Symptom)

A medical database contains a catalog of known illnesses and associated symptoms, which can be described using properties:

- name
- description

(:Disease)

A medical database contains a catalog of known illnesses and associated symptoms, which can be described using properties:

- name
- description

These entities will be modelled as nodes, and connected using relationships of the following types:

(:Patient)-[:HAS]→(:Symptom)

When a patient reports to the clinic, they will describe their symptoms to the nurse or the doctor. The nurse or doctor will then enter 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)

The graph of known symptoms is part of a graph of diseases and their symptoms. 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, they 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

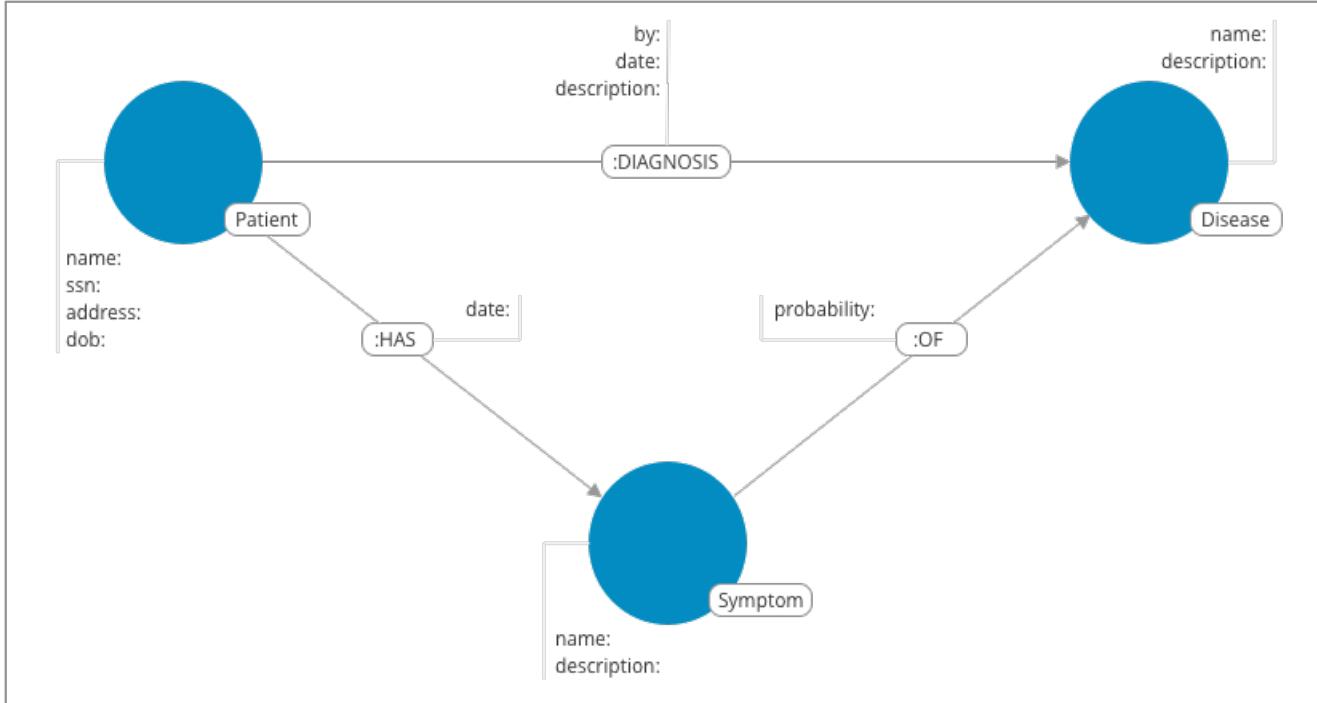


Figure 12. Healthcare use case

The database would be used by a number of different user types, with different needs for access.

- **Doctors** who need to perform diagnosis on 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 administer the database, creating and assigning users.

11.3.2. Security

When building an application for a specific domain, it is common to model the different users within the application itself. However, when working with a database that provides rich user management with roles and privileges, it is possible to model these entirely within the database security model (for more information, see [Cypher Manual](#) □ [Administration](#) □ [Security](#)). This results in separation of concerns for the access control to the data and the data itself. We will show two approaches to using Neo4j security features to support the *healthcare* database application. First, a simple approach using built-in roles, and then a more advanced approach using fine-grained privileges for sub-graph access control.

Our *healthcare* example involves five users of the 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):

Example 59. Creating users

```
CREATE USER charlie SET PASSWORD $secret1 CHANGE NOT REQUIRED;
CREATE USER alice SET PASSWORD $secret2 CHANGE NOT REQUIRED;
CREATE USER daniel SET PASSWORD $secret3 CHANGE NOT REQUIRED;
CREATE USER bob SET PASSWORD $secret4 CHANGE NOT REQUIRED;
CREATE USER tina SET PASSWORD $secret5 CHANGE NOT REQUIRED;
```

At this point the users have no ability to interact with the database, so we need to grant those capabilities 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.

11.3.3. Access control using built-in roles

Neo4j 4.1 comes with a number of built-in roles that cover a number of common needs:

- `PUBLIC` - All users have this role, can by default access the default 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.

Charlie is a researcher and will not need write access to the database, and so he is assigned the `reader` role. 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. We assign them all the `editor` role. Tina is the IT administrator that installs and manages the database. In order to create all other users, Tina is assigned the `admin` role.

Example 60. Granting roles

```
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;
```

A limitation of this approach is that it does allow all users to see all data in the database, and in many real-world scenarios it would be preferable to restrict the users' access. In this example, we would want to restrict the researcher from being able to read any of the patients' personal information, and the receptionist should only be able to see the patient records and nothing more.

These, and more restrictions, could be coded into the application layer. However, it is possible and

more secure to enforce these kinds of fine-grained restrictions directly within the Neo4j security model, by creating custom roles and assigning specific privileges to those roles.

Since we will be creating new custom roles, the first thing to do is revoke the current roles from the users:

Example 61. Revoking roles

```
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 the users are once again unable to do anything, and so we need to start over by building the set of new privileges based on a complete understanding of what we want each user to be able to do.

11.3.4. Sub-graph access control using privileges

With the concept of *privileges*, we can take much more control over what each user is capable of doing. We start by identifying each type of user:

Doctor

Should be able to read and write most of the graph. We would, however, like to prevent the doctor from reading the patient's address. We would also like to make sure the doctor can save *diagnoses* to the database, but not expand the schema of the database 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 on all data, except patients' personal information, and as such should not be able to read most patient properties. To illustrate two different ways of setting up the same effective privileges, we will create two roles and compare them.

Nurse

The nurse should be able to perform all tasks that both the doctor and the receptionist can do. At first one might be tempted to simply grant both those roles, but this does not work as expected. We will demonstrate why below, and instead create a dedicated *nurse* role.

Junior nurse

The senior nurse above is able to save diagnoses just as a doctor can. However, we might wish to have nurses that are not allowed to make that update to the graph. While we could build another role from scratch, this could more easily 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 we wish to restrict access to the patients *SSN*, as well as prevent the administrator from performing the very critical action of saving a diagnosis, something specific to medical professionals. To achieve this, we can create this role by copying the built-in *admin* role and modifying the privileges of that copy.

User manager

It is possible that we would like the IT administrator to be less powerful than described above. We can create a new role from scratch, granting only the specific administrative capabilities we actually

desire.

Before we create the new roles and assign them to Alice, Bob, Daniel, Charlie and Tina, we should define the privileges of each role. Since all users need **ACCESS** privilege to the **healthcare** database, we can add this to 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 all we need to do is deny the two specific actions this role is not supposed to do:

- Should not be able to read any patients social security number.
- Should not be able to perform medical diagnosis.

```
DENY READ {ssn} ON GRAPH healthcare NODES Patient TO itadmin;
DENY CREATE ON GRAPH healthcare RELATIONSHIPS DIAGNOSIS 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;
```

access	action	resource	graph	segment	role
"GRANTED"	"match"	"all_properties"	"*"	"NODE(*)"	
"itadmin"					
"GRANTED"	"write"	"graph"	"*"	"NODE(*)"	
"itadmin"					
"GRANTED"	"match"	"all_properties"	"*"	"RELATIONSHIP(*)"	
"itadmin"					
"GRANTED"	"write"	"graph"	"*"	"RELATIONSHIP(*)"	
"itadmin"					
"GRANTED"	"access"	"database"	"*"	"database"	
"itadmin"					
"GRANTED"	"admin"	"database"	"*"	"database"	
"itadmin"					
"GRANTED"	"constraint"	"database"	"*"	"database"	
"itadmin"					
"GRANTED"	"index"	"database"	"*"	"database"	
"itadmin"					
"GRANTED"	"token"	"database"	"*"	"database"	
"itadmin"					
"DENIED"	"read"	"property(ssn)"	"healthcare"	"NODE(Patient)"	
"itadmin"					
"DENIED"	"create_element"	"graph"	"healthcare"	"RELATIONSHIP(DIAGNOSIS)"	
"itadmin"					



Privileges that were granted or denied earlier can be revoked using the `REVOKE` command. See [the Cypher Manual](#) □ [The REVOKE command](#).

In order for the IT administrator `tina` to be provided these privileges, she 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`, we can login to `healthcare` as `tina` and run the 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, that users cannot tell the difference between data that is not there, and data that is hidden using fine-grained read privileges.

Now remember that we also denied the administrator from saving diagnoses, because that is a critical medical function reserved for only doctors and senior medical staff. We 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].
```



While restrictions on reading data do not result in errors and only make it appear as if the data is not there, restrictions on updating, i.e. writing to the graph will produce an appropriate error when the user attempts to perform an update they are not permitted to do.

Privileges of researcher

Charlie the researcher was previously our only read-only user. We could do something similar to what we did with the `itadmin` role, by copying and modifying the `reader` role. However, we would like to explicitly illustrate how to build a role from scratch. There are various possibilities for building this role using the concepts of either granting or denying a list of privileges:

- Denying privileges:

We could grant the role the ability to find all nodes and read all properties (much like the `reader` role) and then deny read access to the `Patient` properties we want to restrict the researcher from seeing, such as `name`, `SSN` and `address`. This approach is simple but suffers from one problem. If `Patient` nodes are assigned additional properties, *after* we have restricted access, these new properties will automatically be visible to the researcher, which may not be desirable.

Example 62. Denying 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:

An alternative is to only provide specific access to the properties we wish the researcher to see. Then, the addition of new properties will not automatically make them visible to the researcher. In this case, adding new properties to a `Patient` will not mean that the researcher can see them by default. If we wish to have them visible, we need to explicitly grant read access.

Example 63. Granting specific privileges

```
// Create the role first
CREATE ROLE researcherW
// We 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 Charlie now has the privileges we have specified, we assign him to the **researcherB** role (with specifically denied privileges):

```
GRANT ROLE researcherB TO charlie;
```

We can use a version of the **SHOW PRIVILEGES** command to see Charlies access rights:

```
neo4j@system> SHOW USER charlie PRIVILEGES;
```

access	action	resource	graph	segment	role
"GRANTED"	"access"	"database"	"DEFAULT"	"database"	"PUBLIC"
"charlie"					
"GRANTED"	"access"	"database"	"healthcare"	"database"	"PUBLIC"
"charlie"					
"GRANTED"	"match"	"all_properties"	"healthcare"	"NODE(*)"	
"researcherB"	"charlie"				
"DENIED"	"read"	"property(address)"	"healthcare"	"NODE(Patient)"	
"researcherB"	"charlie"				
"DENIED"	"read"	"property(name)"	"healthcare"	"NODE(Patient)"	
"researcherB"	"charlie"				
"DENIED"	"read"	"property(ssn)"	"healthcare"	"NODE(Patient)"	
"researcherB"	"charlie"				
"GRANTED"	"match"	"all_properties"	"healthcare"	"RELATIONSHIP(*)"	
"researcherB"	"charlie"				
"DENIED"	"traverse"	"graph"	"healthcare"	"RELATIONSHIP(DIAGNOSIS)"	
"researcherB"	"charlie"				

Now when Charlie logs into the `healthcare` database and tries to run a command similar to the one used by the `itadmin` above, we 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 Charlie the researcher 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

Now if we revoke the `researcherB` and instead grant the `researcherW` role to Charlie, and re-run these queries, we will see the same results.



Privileges that were granted or denied earlier can be revoked using the `REVOKE` command. See [the Cypher Manual](#) □ [The REVOKE command](#).

Privileges of `doctor`

Doctors should be given the ability to read and write almost everything. We would, however, like to remove the ability to read the patients' `address` property. 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 Alice to have these privileges, we grant her this new role:

```
neo4j@system> GRANT ROLE doctor TO alice;
```

To demonstrate that Alice is not able to see patient addresses, we log in as `alice` to `healthcare` and run the 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 we can see, the doctor has the expected privileges, including being able to see the SSN, but not the address of each patient.

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 this 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

In order to create the **DIAGNOSIS** relationship for the first time, it is required to have the privilege to create new types. This is also true of the property names **doctor**, **created_at** and **updated_at**. This 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. However, this would have the unfortunate side effect of allowing the receptionist 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 for being able to create data you cannot read, that is not desired for this model.

```
neo4j@system> GRANT ROLE receptionist TO bob;
```

With these privileges, if Bob tries to read the entire database, he will still only see the patients:

```
MATCH (n) WITH labels(n) AS labels
RETURN labels, count(*);
```

labels	count(*)
["Patient"]	101

However, Bob is able to see all fields of the Patient 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

We have granted Bob the receptionist the ability to delete patient nodes. This will allow him to delete any new patients he has just created, but will not allow him the ability to delete patients that have already received diagnoses, because those are connected to parts of the graph the receptionist cannot see. Let's demonstrate both these 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 this fails is that Bob can find the (`:Patient`) node, but does not have sufficient traverse rights to find nor delete the outgoing relationships from it. Either he needs to ask Tina the `itadmin` for help for this task, or we can add more privileges 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;
```



Privileges that were granted or denied earlier can be revoked using the `REVOKE` command. See [the Cypher Manual](#) □ [The REVOKE command](#).

Privileges of nurses

As previously described, nurses have the capabilities of both doctors and receptionists. As such it would be tempting to assign them both the `doctor` and the `receptionist` roles. However, this might not have the effect you would expect. If those two roles were created with `GRANT` privileges only, combining them would be simply cumulative. But it turns out the doctor contains some `DENY` privileges, and these always overrule `GRANT`. This means that the nurse will still have the same restrictions as a doctor, which is not what we wanted.

To demonstrate this, let's give it a try:

```
neo4j@system> GRANT ROLE doctor, receptionist TO daniel;
```

Now we can see that the user 'Daniel' has a combined set of privileges:

```
SHOW USER daniel PRIVILEGES;
```

access	action	resource	graph	segment	role
user					
"GRANTED"	"access"	"database"	"DEFAULT"	"database"	"PUBLIC"
"daniel"					
"GRANTED"	"access"	"database"	"healthcare"	"database"	"PUBLIC"
"daniel"					
"GRANTED"	"execute"	"database"	"*"	"FUNCTION(*)"	"PUBLIC"
"daniel"					
"GRANTED"	"execute"	"database"	"*"	"PROCEDURE(*)"	"PUBLIC"
"daniel"					
"GRANTED"	"read"	"all_properties"	"healthcare"	"NODE(*)"	"doctor"
"daniel"					
"GRANTED"	"traverse"	"graph"	"healthcare"	"NODE(*)"	"doctor"
"daniel"					
"GRANTED"	"write"	"graph"	"healthcare"	"NODE(*)"	"doctor"
"daniel"					
"DENIED"	"read"	"property(address)"	"healthcare"	"NODE(Patient)"	"doctor"
"daniel"					
"DENIED"	"set_property"	"property(address)"	"healthcare"	"NODE(Patient)"	"doctor"
"daniel"					
"GRANTED"	"read"	"all_properties"	"healthcare"	"RELATIONSHIP(*)"	"doctor"
"daniel"					
"GRANTED"	"traverse"	"graph"	"healthcare"	"RELATIONSHIP(*)"	"doctor"
"daniel"					
"GRANTED"	"write"	"graph"	"healthcare"	"RELATIONSHIP(*)"	"doctor"
"daniel"					
"GRANTED"	"match"	"all_properties"	"healthcare"	"NODE(Patient)"	
"receptionist"	"daniel"				
"GRANTED"	"set_property"	"all_properties"	"healthcare"	"NODE(Patient)"	
"receptionist"	"daniel"				
"GRANTED"	"create_element"	"graph"	"healthcare"	"NODE(Patient)"	
"receptionist"	"daniel"				
"GRANTED"	"delete_element"	"graph"	"healthcare"	"NODE(Patient)"	
"receptionist"	"daniel"				



Privileges that were granted or denied earlier can be revoked using the `REVOKE` command. See [the Cypher Manual](#) □ [The REVOKE command](#).

Now the intention is that a nurse can perform the actions of a receptionist. This would mean they should be able to read and write the `address` field of the `Patient` nodes.

```
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	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

Clearly the `address` field is invisible. This is due to the `DENIED` privileges we could see in the table earlier. If we tried to write to the address field we would receive an error. This is not the intended behavior. We have two choices to correct otherwise:

- We could redefine the `doctor` role with only whitelisting, requiring that we define each Patient property we wish the doctor to be able to read.
- We can redefine the `nurse` role with the actual intended behavior.

It turns out that the latter choice is by far the simplest. The nurse is essentially the doctor without the `address` restrictions:

```
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 let's assign this role to Daniel and test the new behavior:

```
REVOKE ROLE doctor FROM daniel;
REVOKE ROLE receptionist FROM daniel;
GRANT ROLE nurse TO daniel;
```

When the *improved* nurse Daniel takes another look at the patient records, he 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

Now Daniel can see the previously hidden `address` field. The other main action we want the `nurse` to 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 an action otherwise reserved for the **doctor** role involves more responsibility for the **nurse**. Perhaps it is not desirable to entrust all nurses with this option, which is why we can divide the nurses into *senior* and *junior* nurses. Daniel is currently a senior nurse.

Privileges of junior nurses

When we tried to create the senior nurse by combining the **doctor** and **receptionist** roles, that did not work out. As previously mentioned, it would work to combine two roles if the intention is to increase capabilities and the roles were created with **GRANT** privileges only. It is also possible to combine two roles if the intention is to reduce capabilities and the combination brings in **DENY** privileges.

Consider this case, we would like a junior nurse to be able to perform the same actions as a senior nurse, but not be able to save diagnoses. We could create a special role that contains specifically only the additional restrictions:

```
CREATE ROLE disableDiagnoses;
DENY CREATE ON GRAPH healthcare RELATIONSHIPS DIAGNOSIS TO disableDiagnoses;
```

Now let's assign this role to Daniel and test the new behaviour:

```
GRANT ROLE disableDiagnoses TO daniel;
```

If we look at what privileges Daniel now has, it will be the combination of the two roles **nurse** and **disableDiagnoses**:

```
neo4j@system> SHOW USER daniel PRIVILEGES;
```

access	action	resource	graph	segment	role
"GRANTED"	"access"	"database"	"DEFAULT"	"database"	
"PUBLIC"	"access"	"daniel"			
"GRANTED"	"access"	"database"	"healthcare"	"database"	
"PUBLIC"	"access"	"daniel"			
"GRANTED"	"execute"	"database"	"*"	"FUNCTION(*)"	
"PUBLIC"	"execute"	"daniel"			
"GRANTED"	"execute"	"database"	"*"	"PROCEDURE(*)"	
"PUBLIC"	"execute"	"daniel"			
"DENIED"	"create_element"	"graph"	"healthcare"	"RELATIONSHIP(DIAGNOSIS)"	
"disabledDiagnoses"	"daniel"				
"GRANTED"	"read"	"all_properties"	"healthcare"	"NODE(*)"	
"nurse"	"read"	"daniel"			
"GRANTED"	"traverse"	"graph"	"healthcare"	"NODE(*)"	
"nurse"	"traverse"	"daniel"			
"GRANTED"	"write"	"graph"	"healthcare"	"NODE(*)"	
"nurse"	"write"	"daniel"			
"GRANTED"	"read"	"all_properties"	"healthcare"	"RELATIONSHIP(*)"	
"nurse"	"read"	"daniel"			
"GRANTED"	"traverse"	"graph"	"healthcare"	"RELATIONSHIP(*)"	
"nurse"	"traverse"	"daniel"			
"GRANTED"	"write"	"graph"	"healthcare"	"RELATIONSHIP(*)"	
"nurse"	"write"	"daniel"			

Daniel can still see 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 he tries to save a diagnosis to the database, he will be denied:

```

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].

Promoting Daniel back to senior nurse will be as simple as revoking the role that introduced the restriction:

```
REVOKE ROLE disableDiagnoses FROM daniel;
```

Building a custom administrator role

Originally we created the `itadmin` role by copying the built-in `admin` role and adding restrictions. However, we have also shown cases where using blacklisting can be less convenient than whitelisting. So can we instead build the administrator role from the ground up?

Let's review the purpose of this role. The intention is that Tina, the administrator, can create new users and assign them to the product roles. We can create a new role called `userManager` and grant it 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;

```

We need to revoke the `itadmin` role from Tina and grant her the `userManager` role instead:

```
REVOKE ROLE itadmin FROM tina
GRANT ROLE userManager TO tina
```

The three privileges we've granted will allow:

- `USER MANAGEMENT` allows creating, updating and dropping users
- `ROLE MANAGEMENT` allows assigning roles to users
- `SHOW PRIVILEGE` allows listing the users privileges

Listing Tina's new privileges should show a much shorter list than when she was a more powerful administrator:

```
neo4j@system> SHOW USER tina PRIVILEGES;
```

access	action	resource	graph	segment	role	user
"GRANTED"	"access"	"database"	"DEFAULT"	"database"	"PUBLIC"	"tina"
"GRANTED"	"access"	"database"	"healthcare"	"database"	"PUBLIC"	"tina"
"GRANTED"	"execute"	"database"	"*"	"FUNCTION(*)"	"PUBLIC"	"tina"
"GRANTED"	"execute"	"database"	"*"	"PROCEDURE(*)"	"PUBLIC"	"tina"
"GRANTED"	"role_management"	"database"	"*"	"database"	"userManager"	"tina"
"GRANTED"	"show_privilege"	"database"	"*"	"database"	"userManager"	"tina"
"GRANTED"	"user_management"	"database"	"*"	"database"	"userManager"	"tina"



We have not granted any other privilege management privileges. How much power this role should have would depend on the requirements of the system. Refer to the section [Cypher Manual □ Security of Administration](#) 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 'secret' CHANGE REQUIRED;  
GRANT ROLE receptionist TO sally;  
SHOW USER sally PRIVILEGES;
```

access	action	resource	graph	segment	role	user
"GRANTED"	"access"	"database"	"DEFAULT"	"database"	"PUBLIC"	"sally"
"GRANTED"	"access"	"database"	"healthcare"	"database"	"PUBLIC"	"sally"
"GRANTED"	"execute"	"database"	"*"	"FUNCTION(*)"	"PUBLIC"	"sally"
"GRANTED"	"execute"	"database"	"*"	"PROCEDURE(*)"	"PUBLIC"	"sally"
"GRANTED"	"match"	"all_properties"	"healthcare"	"NODE(Patient)"	"receptionist"	"sally"
"GRANTED"	"set_property"	"all_properties"	"healthcare"	"NODE(Patient)"	"receptionist"	"sally"
"GRANTED"	"create_element"	"graph"	"healthcare"	"NODE(Patient)"	"receptionist"	"sally"
"GRANTED"	"delete_element"	"graph"	"healthcare"	"NODE(Patient)"	"receptionist"	"sally"

11.4. Integration with LDAP directory services

This section describes Neo4j support for integrating with LDAP systems.

- [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 \(Option 1\)](#)
 - [Configure Neo4j to support `sAMAccountName` authentication \(Option 2\)](#)
 - [Configure Neo4j to support `sAMAccountName` authentication \(Option 3\)](#)
- [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](#)

11.4.1. 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 [\[configuration-settings\]](#).

11.4.2. LDAP configuration parameters

Parameter name	Default value	Description
<code>dbms.security.ldap.authentication.user_dn_template</code>	<code>uid={0},ou=users,dc=example,dc=com</code>	Converts usernames into LDAP-specific fully qualified names required for logging in.
<code>dbms.security.ldap.authorization.user_search_base</code>	<code>ou=users,dc=example,dc=com</code>	Sets the base object or named context to search for user objects.
<code>dbms.security.ldap.authorization.user_search_filter</code>	<code>(&(objectClass=*)(uid={0}))</code>	Sets up an LDAP search filter to search for a user principal.
<code>dbms.security.ldap.authorization.group_membership_attributes</code>	<code>memberOf</code>	Lists attribute names on a user object that contains groups to be used for mapping to roles. Common values: <code>memberOf</code> and <code>gidNumber</code> .
<code>dbms.security.ldap.authorization.group_to_role_mapping</code>		Lists an authorization mapping from groups to the pre-defined built-in roles <code>admin</code> , <code>architect</code> , <code>publisher</code> , <code>editor</code> , and <code>reader</code> , or to any other custom-defined roles.

All settings are defined at server startup time in the default configuration file [`neo4j.conf`](#).

11.4.3. 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 will be used as a security provider for both authentication and authorization.

11.4.4. 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 comma-separated list of the LDAP group names and the value is a comma-separated list of the corresponding role names. For example, `group1=role1;group2=role2;group3=role3,role4,role5;group4,group5=role6`.

Example of LDAP groups to Neo4j roles mapping

```
dbms.security.ldap.authorization.group_to_role_mapping=\n    "cn=Neo4j Read Only,cn=users,dc=example,dc=com"      = reader;      \n    "cn=Neo4j Read-Write,cn=users,dc=example,dc=com"     = editor,publisher; \n    "cn=Neo4j Read-Write,cn=users,dc=example,dc=com", "cn=Neo4j Create Data,cn=users,dc=example,dc=com"\n    = publisher; \\ \n    "cn=Neo4j Create Data,cn=users,dc=example,dc=com", "cn=Neo4j Schema Manager,cn=users,dc=example,dc=com"\n    = architect; \\ \n    "cn=Neo4j Administrator,cn=users,dc=example,dc=com" = admin; \\ \n    "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.

11.4.5. 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 (Option 1)

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 `sAMAccountName` authentication (Option 2)

This is an alternative configuration for Active Directory that allows users to log in using `sAMAccountName`. 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:

```
dbms.security.ldap.authorization.user_search_base=cn=Users,dc=example,dc=com  
dbms.security.ldap.authorization.user_search_filter=(&(objectClass*)(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. 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
```



`sAMAccountName` requires `dbms.security.ldap.authorization.system_username` and `dbms.security.ldap.authorization.system_password` to be used, since there is no way to log in through LDAP directly with the `sAMAccountName`. Instead, the login name will be resolved to a DN that will be used to log in with.

- d. Add the following line to the `neo4j.conf` file:

```
dbms.security.ldap.authentication.use_samaccountname=true
```

Configure Neo4j to support `sAMAccountName` authentication (Option 3)

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.use_samaccountname` is not configured.

11.4.6. 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. 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](#).

11.4.7. 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
```

11.4.8. 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 41. Auth cache parameters

Parameter name	Default value	Description
dbms.security.ldap.authentication.cache_enabled	true	Determines whether or not to cache the result of authentication via the LDAP server. Whether authentication caching should be enabled or not must be considered in view of your company's security guidelines.
dbms.security.auth_cache_ttl	600 seconds	Is the time to live (TTL) for cached authentication and authorization info. Setting the TTL to 0 disables all auth caching. A short TTL requires more frequent re-authentication and re-authorization, which can impact performance. A very long TTL means that changes to the users settings on an LDAP server may not be reflected in the Neo4j authorization behaviour in a timely manner. Valid units are ms, s, m; default unit is s.

An administrator can clear the auth cache to force the re-querying of authentication and authorization information from the federated auth provider system. Use Neo4j Browser or Neo4j Cypher Shell to execute this statement:

```
CALL dbms.security.clearAuthCache()
```

11.4.9. 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` will be used.

```
dbms.security.ldap.host=ldaps://myactivedirectory.example.com
dbms.security.ldap.host=ldaps://myactivedirectory.example.com:636
```

11.4.10. 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=mypassword  
dbms.jvm.additional=-Djavax.net.ssl.trustStore=/path/to/MyCert.jks  
dbms.jvm.additional=-Djavax.net.ssl.trustStorePassword=mypassword
```

11.5. Manage procedure and user-defined function permissions

This section describes how access control works with procedures and user-defined functions in Neo4j.

11.5.1. Introduction

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.

11.5.2. 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 64. 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 65. 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.

11.5.3. 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 66. 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 67. 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.

11.5.4. 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 68. 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 69. 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 created 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

11.6. Terminology

This section lists the relevant terminology related to authentication and authorization in Neo4j.

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 12. Security

This chapter covers important security aspects in Neo4j.

Ensure your physical data security by following industry best practices with regard to server and network security.

This chapter includes the following:

- [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.

12.1. Securing extensions

This section describes how to use white listing to ensure the security of custom-written additions in Neo4j.

Neo4j can be extended by writing custom code which can be invoked directly from Cypher, as described in [Java Reference](#) □ [Procedures and functions](#). This section describes how to ensure the security of these additions.

12.1.1. White listing

White listing can be used to allow the loading of only a few extensions from a larger library.

The configuration setting `dbms.security.procedures.allowlist` is used to name certain procedures that should be available from a library. It defines a comma-separated list of procedures that are to be loaded. The list may contain both fully qualified procedure names, and partial names with the wildcard `*`.

Example 70. White listing

In this example we wish 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 white listing
dbms.security.procedures.allowlist=apoc.coll.*,apoc.load.*
```

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.

- 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.

12.2. SSL framework

Describes SSL/TLS integration for securing communication channels in Neo4j.

Neo4j supports the securing of communication channels using standard SSL/TLS technology.

12.2.1. Introduction

The SSL support is enabled per communication channel and requires SSL certificates encoded in the `PEM` format. The process is described in the following sections.

12.2.2. Certificates

The instructions in this section assume that you have already acquired the required [certificates](#).

All certificates must be in the `PEM` format, and they can be combined into one file. The private key is also required to be in the `PEM` format. Multi-host and wildcard certificates are supported.

12.2.3. Configuration

The SSL policies are configured by assigning values to parameters of the following format:

`dbms.ssl.policy.<scope>.<setting-suffix>`

The `scope` is the name of the communication channel, and must be one of `bolt`, `https`, `cluster`, `backup` or `fabric`.

Each policy needs to be explicitly enabled by setting:

`dbms.ssl.policy.<scope>.enabled=true`

Settings

The valid values for `setting-suffix` are described below.

Setting suffix	Description	Default value
Basic		
<code>enabled</code>	Setting this to <code>true</code> will enable this policy.	<code>false</code>
<code>base_directory</code>	The base directory under which cryptographic objects are searched for by default.	<code>certificates/<scope></code>
<code>private_key</code>	The private key used for authenticating and securing this instance.	<code>private.key</code>
<code>private_key_password</code>	The passphrase to decode the private key. Only applicable for encrypted private keys.	

Setting suffix	Description	Default value
<code>public_certificate</code>	A public certificate matching the private key signed by a Certificate Authority (CA).	<code>public.crt</code>
<code>trusted_dir</code>	A directory populated with certificates of trusted parties.	<code>trusted/</code>
<code>revoked_dir</code>	A directory populated with certificate revocation lists (CRLs).	<code>revoked/</code>
Advanced		
<code>verify_hostname</code>	Enabling this setting will turn on client-side hostname verification. After the client has received the servers public certificate, it will compare the address it used against the certificate Common Name (CN) and Subject Alternative Names (SAN) fields. If the address used doesn't match those fields, the client will disconnect.	<code>false</code>
<code>ciphers</code>	A comma-separated list of cipher suites that will be allowed during cipher negotiation. Valid values depend on the current JRE and SSL provider, see note below for examples.	Java platform default allowed cipher suites
<code>tls_versions</code>	A comma-separated list of allowed TLS versions.	<code>TLSv1.2</code>
<code>client_auth</code>	Whether or not clients must be authenticated. Setting this to <code>REQUIRED</code> effectively enables mutual authentication for servers. Available values given to this setting are <code>NONE</code> , <code>OPTIONAL</code> , or <code>REQUIRED</code> .	<code>OPTIONAL</code> for <code>bolt</code> and <code>https</code> and <code>REQUIRED</code> for <code>cluster</code> and <code>backup</code> .
<code>trust_all</code>	Setting this to <code>true</code> will result in all clients and servers being trusted. The content of the <code>trusted_dir</code> directory will be ignored. Use of this is discouraged, since it will not offer security. It is provided as a mean of debugging.	<code>false</code>



Ciphers supported by the Oracle JRE can be found here: <https://docs.oracle.com/en/java/javase/11/docs/specs/security/standard-names.html#jsse-cipher-suite-names>

For security reasons, Neo4j will not attempt to automatically create any of these directories. The creation of an SSL policy therefore requires the appropriate file system structure to be set up manually. Note that the existence of the directories is mandatory, as well as the presence of the certificate file and the private key. Ensure correct permissions are set on the private key, such that only the Neo4j user can read it.

Example 71. Enable Bolt SSL

In this example we will configure Bolt to use SSL/TLS. As the simplest configuration possible, we will just enable it in `neo4j.conf` and rely on the default values:

```
dbms.ssl.policy.bolt.enabled=true
```

Then create the mandatory directories:

```
$neo4j-home> mkdir certificates/bolt  
$neo4j-home> mkdir certificates/bolt/trusted  
$neo4j-home> mkdir certificates/bolt/revoked
```

Finally, place the files `private.key` and `public.crt` into the base directory:

```
$neo4j-home> cp /path/to/certs/private.key certificates/bolt  
$neo4j-home> cp /path/to/certs/public.crt certificates/bolt
```

The base directory should now show the following listings:

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

12.2.4. Choosing an SSL provider

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 utilize OpenSSL:

1. Install a suitable dependency into the `plugins/` folder of Neo4j. Dependencies can be downloaded from <https://netty.io/wiki/forked-tomcat-native.html>.
2. Set `dbms.netty.ssl.provider=OPENSSL`.



Using OpenSSL can significantly improve performance, especially for AES-GCM-cryptos, e.g. `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`.

12.2.5. 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](#).

12.3. Browser credentials handling

This section explains how to control how credentials are handled in Neo4j Browser.

Neo4j Browser has two mechanisms for avoiding users having to repeatedly enter their Neo4j credentials.

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.

12.4. Security checklist

This section provides a summary of recommendations regarding security in Neo4j.

Below is a simple checklist highlighting the specific areas within Neo4j that may need some extra attention in order to ensure the appropriate level of security for your application.

1. Deploy Neo4j on safe servers in safe networks:

- a. Use subnets and firewalls.
- b. Only open up for the necessary ports. 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 leave a security hole open 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 (refer to [Dump and load databases](#)) and backups (refer to [Perform a backup](#)).
- c. Manage access to 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 open up for encrypted Bolt or HTTPS.

- b. Use SSL certificates issued from a trusted Certificate Authority.
 - i. For configuring your Neo4j installation to use encrypted communication, refer to [SSL framework](#).
 - ii. If using Causal Clustering, configure and use encryption for intra-cluster communication. For details, see [Intra-cluster encryption](#).
 - iii. If using Causal Clustering, configure and use encryption for backups. This ensures that only servers with the specified SSL policy and SSL certificates will be able to access the server and perform the backup.
 - iv. For configuring your Bolt and/or HTTPS connectors, refer to [Configure connectors](#).
 - v. If using LDAP, configure your LDAP system with encryption via StartTLS; see [Use LDAP with encryption via StartTLS](#).
4. Be on top of the security for custom extensions:
 - a. Validate any custom code that you deploy (procedures and unmanaged extensions) and ensure that they do not expose any parts of the product or data unintentionally.
 - b. Survey the settings `dbms.security.procedures.unrestricted` and `dbms.security.procedures.allowlist` to ensure that they exclusively contain intentionally exposed extensions.
5. Ensure 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. If `LOAD CSV` is 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`, which enables the native auth provider.
9. Survey your `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.

Chapter 13. Monitoring

This chapter describes the tools that are available for monitoring Neo4j.

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:

- [Metrics](#)
 - [Expose metrics](#)
 - [Metrics reference](#)
- [Logging](#)
 - [General logging](#)
 - [Security events logging](#)
 - [Query logging](#)
 - [Transaction logging](#)
- [Query management](#)
 - [List all running queries](#)
 - [List all active locks for a query](#)
 - [Terminate multiple queries](#)
 - [Terminate a single query](#)
- [Transaction management](#)
 - [Configure transaction timeout](#)
 - [Configure lock acquisition timeout](#)
 - [List all running transactions](#)
- [Connection management](#)
 - [List all network connections](#)
 - [Terminate multiple network connections](#)
 - [Terminate a single network connection](#)
- [Background job management](#)
 - [Listing active background jobs](#)
 - [Listing failed job executions](#)
- [Monitoring a Causal Cluster](#)
 - [Procedures for monitoring a Causal Cluster](#)
 - [Endpoints for status information](#)
- [Monitoring the state of individual databases](#)

13.1. Metrics

This section describes the Neo4j metrics output facilities.

This section describes the following:

- [Types of metrics](#)
 - [Global metrics](#)
 - [Database metrics](#)
- [Expose metrics](#)
 - [Enable metrics logging](#)
 - [Graphite](#)
 - [Prometheus](#)
 - [CSV files](#)
 - [JMX MBeans](#)
- [Metrics reference](#)
 - [General-purpose metrics](#)
 - [Metrics specific to Causal Clustering](#)
 - [Java Virtual Machine metrics](#)

13.1.1. Types of metrics

This section describes the types of metrics available in Neo4j.

Neo4j provides a built-in metrics subsystem. Reported metrics can be queried via JMX, retrieved from CSV files, or consumed by third-party monitoring tools.

Neo4j has the following types of metrics:

- [Global](#) - covers the whole Neo4j DBMS.
- [Per database](#) - covers an individual database.

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`.

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:

- Page cache metrics
- GC metrics

- Thread metrics
- Memory pool metrics
- Memory buffers metrics
- File descriptor metrics
- Database operation metrics
- Bolt metrics
- Web Server 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`.

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 it 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

13.1.2. Expose metrics

This section describes how to log and display various metrics by using the Neo4j metrics output facilities.

This section describes the following:

- [Enable metrics logging](#)
- [Graphite](#)
- [Prometheus](#)
- [CSV files](#)
- [JMX MBeans](#)

Enable metrics logging

The metrics that are enabled by default have been changed in the 4.2 release.



Any specific metrics that you want to be enabled **must** be specified in the `metrics.filter`.

Additionally in the 4.2 release, metrics are no longer exposed via JMX by default. These can be enabled by adding `metrics.jmx.enabled=true` to `neo4j.conf`.

By default, metrics logging into CSV files is enabled. A subset of metrics are enabled once `metrics.enabled=true` is set, and you can use the `metrics.filter` setting to select the specific metrics you want to enable.

The `metrics.filter` should be specified as a comma separated list of globbing patterns. For example, `*check_point*,neo4j.dbms.page_cache.evictions` will enable all checkpoint metrics and the pagecache eviction metric. When specifying a complete metric name, you should take into account whether `metrics.namespaces.enabled` is set:

```
# Setting for enabling all supported metrics.  
metrics.enabled=true  
  
# Setting for enabling clear separation between global and database metrics.  
metrics.namespaces.enabled=true  
  
# Setting for exposing metrics. Should be specified as a comma separated list of globbing patterns.  
metrics.filter=*causal_clustering*,*check_point*,neo4j.dbms.page_cache.evictions
```

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 <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.

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

Expose metrics over JMX MBeans.

In order to enable metrics exposure via JMX, add the following setting to `neo4j.conf`:

```
# Enable settings export via JMX. Default is 'false'.
metrics.jmx.enabled=true
```

13.1.3. Metrics reference

This section provides a listing of available metrics.

This section describes the following:

- [General-purpose metrics](#)
- [Metrics specific to Causal Clustering](#)
- [Java Virtual Machine metrics](#)

General-purpose metrics

Table 42. Database store size metrics

Name	Description
<prefix>.store.size.total	The total size of the database and transaction logs, in bytes. (gauge)
<prefix>.store.size.database	The size of the database, in bytes. (gauge)

Table 43. 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. (gauge)

Table 44. Database data metrics

Name	Description
<prefix>.ids_in_use.relationship_type	The total number of different relationship types stored in the database. (gauge)
<prefix>.ids_in_use.property	The total number of different property names used in the database. (gauge)
<prefix>.ids_in_use.relationship	The total number of relationships stored in the database. (gauge)
<prefix>.ids_in_use.node	The total number of nodes stored in the database. (gauge)

Table 45. Database page cache metrics

Name	Description
<prefix>.page_cache.eviction_exceptions	The total number of exceptions seen during the eviction process in the page cache. (counter)
<prefix>.page_cache.flushes	The total number of page flushes executed by the page cache. (counter)
<prefix>.page_cache.merges	The total number of page merges executed by the page cache. (counter)
<prefix>.page_cache.unpins	The total number of page unpins executed by the page cache. (counter)
<prefix>.page_cache.pins	The total number of page pins executed by the page cache. (counter)
<prefix>.page_cache.evictions	The total number of page evictions executed by the page cache. (counter)
<prefix>.page_cache.page_faults	The total number of page faults happened in the page cache. (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. (gauge)
<prefix>.page_cache.usage_ratio	The ratio of number of used pages to total number of available pages. (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)

Table 46. 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. (counter)
<prefix>.transaction.active	The number of currently active transactions. (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. (counter)
<prefix>.transaction.committed_read	The total number of committed read transactions. (counter)
<prefix>.transaction.committed_write	The total number of committed write transactions. (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. (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. (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 47. Cypher metrics

Name	Description
<prefix>.cypher.replan_events	The total number of times Cypher has decided to re-plan a query. (counter)
<prefix>.cypher.replan_wait_time	The total number of seconds waited between query replans. (counter)

Table 48. 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)

Name	Description
<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)

Table 49. Bolt metrics

Name	Description
<prefix>.bolt.sessions_started	The total number of Bolt sessions started since this instance started. This includes both succeeded and failed sessions (deprecated, use connections_opened instead). (counter)
<prefix>.bolt.connections_opened	The total number of Bolt connections opened since this instance started. This includes both succeeded and failed connections. (counter)
<prefix>.bolt.connections_closed	The total number of Bolt connections closed since this instance started. This includes both properly and abnormally ended connections. (counter)
<prefix>.bolt.connections_running	The total number of Bolt connections currently being executed. (gauge)
<prefix>.bolt.connections_idle	The total number of Bolt connections sitting idle. (gauge)
<prefix>.bolt.messages_received	The total number of messages received via Bolt since this instance started. (counter)
<prefix>.bolt.messages_started	The total number of messages that began processing since this instance started. This is different from messages received in that this counter tracks how many of the received messages have been taken on by a worker thread. (counter)
<prefix>.bolt.messages_done	The total number of messages that completed processing since this instance started. This includes successful, failed and ignored Bolt messages. (counter)
<prefix>.bolt.messages_failed	The total number of messages that failed processing since this instance started. (counter)
<prefix>.bolt.accumulated_queue_time	The accumulated time messages have spent waiting for a worker thread. (counter)
<prefix>.bolt.accumulated_processing_time	The accumulated time worker threads have spent processing messages. (counter)

Table 50. 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. (gauge)

Table 51. 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 52. Server metrics

Name	Description
<prefix>.server.threads.jetty.idle	The total number of idle threads in the jetty pool. (gauge)
<prefix>.server.threads.jetty.all	The total number of threads (both idle and busy) in the jetty pool. (gauge)

Metrics specific to Causal Clustering

Table 53. Raft core metrics

Name	Description
<prefix>.causal_clustering.core.append_index	Append index of the RAFT log. (gauge)
<prefix>.causal_clustering.core.commit_index	Commit index of the RAFT log. (gauge)
<prefix>.causal_clustering.core.applied_index	Applied index of the RAFT log. (gauge)
<prefix>.causal_clustering.core.term	RAFT Term of this server. (gauge)
<prefix>.causal_clustering.core.tx_retries	Transaction retries. (counter)
<prefix>.causal_clustering.core.is_leader	Is this server the leader? (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.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	Raft replication new request count. (counter)
<prefix>.causal_clustering.core.replication_attempt	Raft replication attempt count. (counter)

Name	Description
<prefix>.causal_clustering.core.replication_fail	Raft Replication fail count. (counter)
<prefix>.causal_clustering.core.replication_maybe	Raft Replication maybe count. (counter)
<prefix>.causal_clustering.core.replication_success	Raft Replication success count. (counter)
<prefix>.causal_clustering.core.last_leader_message	Time elapsed since last message from leader in milliseconds. (gauge)

Table 54. 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)

Table 55. 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)

Java Virtual Machine Metrics

These metrics are environment dependent and they may vary on different hardware and with JVM configurations. Typically these metrics will show information about garbage collections (for example the number of events and time spent collecting), memory pools and buffers, and finally the number of active threads running.

Table 56. GC metrics.

Name	Description
<prefix>.vm.gc.time.%s	Accumulated garbage collection time in milliseconds. (counter)
<prefix>.vm.gc.count.%s	Total number of garbage collections. (counter)

Table 57. JVM memory buffers metrics.

Name	Description
<code><prefix>.vm.memory.buffer.%s.count</code>	Estimated number of buffers in the pool. (gauge)
<code><prefix>.vm.memory.buffer.%s.used</code>	Estimated amount of memory used by the pool. (gauge)
<code><prefix>.vm.memory.buffer.%s.capacity</code>	Estimated total capacity of buffers in the pool. (gauge)

Table 58. JVM memory pools metrics.

Name	Description
<code><prefix>.vm.memory.pool.%s</code>	Estimated number of buffers in the pool. (gauge)

Table 59. JVM threads metrics.

Name	Description
<code><prefix>.vm.thread.count</code>	Estimated number of active threads in the current thread group. (gauge)
<code><prefix>.vm.thread.total</code>	The total number of live threads including daemon and non-daemon threads. (gauge)

13.2. Logging

This section describes the logging mechanisms in Neo4j, including general log files, error messages, and severity levels.

Neo4j provides the following log types:

- General Neo4j logs
- Inspection of queries that are run in the database
- Security events that have occurred
- Transaction logs

The section describes the following:

- [General logging](#)
- [User log file configuration](#)
- [Debug log file configuration](#)
- [Garbage collection log file configuration](#)
- [HTTP request log file configuration](#)
- [Security events logging](#)
- [Query logging](#)
- [Transaction logging](#)

13.2.1. General logging

The root directory where the general log files are located is configured by `dbms.directories.logs`.

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`.

For more information on where files are located, see [File locations](#).

Log files

The following table describes the Neo4j general log files and the information they contain.

Filename	Description
<code>neo4j.log</code>	The standard log, where general information about Neo4j is written. Not written for Debian and RPM packages.
<code>debug.log</code>	Information useful when debugging problems with Neo4j.
<code>http.log</code>	Request log for the HTTP API.
<code>gc.log</code>	Garbage collection logging provided by the JVM.
<code>query.log</code>	Log of executed queries that takes longer than a specified threshold.
<code>security.log</code>	Log of security events.
<code>service-error.log</code>	(Windows) Log of errors encountered when installing or running the Windows service.

Configuration setting	Default value	Description
<code>dbms.directories.logs</code>	<code>logs</code>	Path of the logs directory.
<code>dbms.logs.user.path</code>	<code>neo4j.log</code>	Path to the user log file.
<code>dbms.logs.debug.path</code>	<code>debug.log</code>	Path to the debug log file.
<code>dbms.logs.http.path</code>	<code>http.log</code>	Path to HTTP request log.
<code>dbms.logs.query.path</code>	<code>query.log</code>	Path to the query log file.
<code>dbms.logs.security.path</code>	<code>security.log</code>	Path to the security log file.

13.2.2. User log file configuration

User log file configuration	Default value	Description
<code>dbms.logs.user.rotation.delay</code>	<code>5m</code>	Minimum time interval after last rotation of the user log before it may be rotated again.
<code>dbms.logs.user.rotation.keep_number</code>	<code>7</code>	Maximum number of history files for the user log.
<code>dbms.logs.user.rotation.size</code>	<code>0B</code>	Threshold size for rotation of the user log. If set to 0 log rotation is disabled.
<code>dbms.logs.user.stdout_enabled</code>	<code>true</code>	Send user logs to the process stdout. If this is disabled then logs will instead be sent to the file <code>neo4j.log</code> .

13.2.3. Debug log file configuration

Debug log file configuration	Default value	Description
<code>dbms.logs.debug.level</code>	<code>INFO</code>	Debug log level threshold.
<code>dbms.logs.debug.rotation.delay</code>	<code>5m</code>	Minimum time interval after last rotation of the debug log before it may be rotated again.

Debug log file configuration	Default value	Description
<code>dbms.logs.debug.rotation.keep_number</code>	7	Maximum number of history files for the debug log.
<code>dbms.logs.debug.rotation.size</code>	20M	Threshold size for rotation of the debug log.

The following table lists all message types raised by Neo4j and their severity level:

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 debug log level threshold use the configuration setting `dbms.logs.debug.level`.

13.2.4. Garbage collection log file configuration

Garbage collection log file configuration	Default value	Description
<code>dbms.logs.gc.enabled</code>	false	Enable garbage collection logging.
<code>dbms.logs.gc.options</code>		Garbage collection logging options.
<code>dbms.logs.gc.rotation.keep_number</code>	0	Maximum number of history files for the garbage collection log.
<code>dbms.logs.gc.rotation.size</code>		Threshold size for rotation of the garbage collection log.

13.2.5. HTTP request log file configuration

HTTP request log file configuration	Default value	Description
<code>dbms.logs.http.enabled</code>	false	Enable HTTP request logging.
<code>dbms.logs.http.rotation.keep_number</code>	5	Maximum number of history files for the HTTP request log.
<code>dbms.logs.http.rotation.size</code>	20M	Threshold size for rotation of the HTTP request log.

13.2.6. Security events logging

Neo4j provides security event logging that records all security events.

For native user management, the following actions are recorded:

- Login attempts - per default both successful and unsuccessful logins are recorded.
- All [administration commands](#) run towards the system database.
- All [security procedures](#) run towards the system database.

Security log file configuration

The name of the security log file is `security.log` by default, (see [dbms.logs.security.path](#)).

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 file:

Security log file configuration	Default value	Description
<code>dbms.logs.security.level</code>	<code>INFO</code>	Security log level threshold.
<code>dbms.logs.security.rotation.size</code>	<code>20M</code>	Sets the file size at which the security event log will auto-rotate.
<code>dbms.logs.security.rotation.delay</code>	<code>300s</code>	Sets the minimum time interval after the last log rotation occurred, before the log may be rotated again.
<code>dbms.logs.security.rotation.keep_number</code>	<code>7</code>	Sets number of historical log files kept.

If using LDAP as the authentication method, some cases of LDAP misconfiguration will also be 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. Logging of successful logins is disabled by setting the `dbms.security.log_successful_authentication` parameter in the `neo4j.conf` file:

```
dbms.security.log_successful_authentication=false
```

Below is an example of the security log:

```
2019-12-09 13:45:00.796+0000 INFO [AsyncLog @ 2019-12-09 ...] [johnsmith]: logged in
2019-12-09 13:47:53.443+0000 ERROR [AsyncLog @ 2019-12-09 ...] [johndoe]: failed to log in: invalid
principal or credentials
2019-12-09 13:48:28.566+0000 INFO [AsyncLog @ 2019-12-09 ...] [johnsmith]: CREATE USER janedoe SET
PASSWORD '*****' CHANGE REQUIRED
2019-12-09 13:48:32.753+0000 INFO [AsyncLog @ 2019-12-09 ...] [johnsmith]: CREATE ROLE custom
2019-12-09 13:49:11.880+0000 INFO [AsyncLog @ 2019-12-09 ...] [johnsmith]: GRANT ROLE custom TO janedoe
2019-12-09 13:49:34.979+0000 INFO [AsyncLog @ 2019-12-09 ...] [johnsmith]: GRANT TRAVERSE ON GRAPH *
NODES A, B (*) TO custom
2019-12-09 13:49:37.053+0000 INFO [AsyncLog @ 2019-12-09 ...] [johnsmith]: DROP USER janedoe
```

13.2.7. Query logging

Neo4j can be configured to log queries executed in the database.

Query logging is enabled by default and is controlled by the setting `dbms.logs.query.enabled`.

Configuration options are:

Option	Default	Description
<code>OFF</code>		Will completely disable logging.

Option	Default	Description
INFO		Will log at the end of queries that have either succeeded or failed. The <code>dbms.logs.query.threshold</code> parameter is used to determine the threshold for logging a query. If the execution of a query takes a longer time than this threshold, it will be logged. Setting the threshold to <code>0s</code> will result in all queries being logged.
VERBOSE	Yes	Will log all queries at both start and finish, regardless of <code>dbms.logs.query.threshold</code> .

Query log file configuration

The name of the query log file is `query.log` by default, (see `dbms.logs.query.path`).

Rotation of the query log can be configured in the `neo4j.conf` configuration file.

The following configuration settings are available for the query log file:

Query log file configuration	Default value	Description
<code>dbms.logs.query.allocation_logging_enabled</code>	<code>true</code>	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 <code>dbms.track_query_allocation=true</code> .
<code>dbms.logs.query.early_raw_logging_enabled</code>	<code>false</code>	Log query text and parameters without obfuscating passwords. This allows queries to be logged earlier before parsing starts.
<code>dbms.logs.query.enabled</code>	<code>VERBOSE</code>	Log executed queries.
<code>dbms.logs.query.page_logging_enabled</code>	<code>false</code>	Log page hits and page faults for the executed queries being logged.
<code>dbms.logs.query.parameter_full_entities</code>	<code>false</code>	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 <code>dbms.logs.query.parameter_logging_enabled=true</code> .
<code>dbms.logs.query.parameter_logging_enabled</code>	<code>true</code>	Log parameters for the executed queries being logged.
<code>dbms.logs.query.rotation.keep_number</code>	<code>7</code>	Maximum number of history files for the query log.
<code>dbms.logs.query.rotation.size</code>	<code>20M</code>	The file size in bytes at which the query log will auto-rotate.
<code>dbms.logs.query.runtime_logging_enabled</code>	<code>true</code>	Logs which runtime that was used to run the query.
<code>dbms.logs.query.threshold</code>	<code>0s</code>	If the execution of query takes a longer time than this threshold, the query is logged once completed (provided query logging is set to <code>INFO</code>). A threshold of 0 seconds, will log all queries.

Query log file configuration	Default value	Description
<code>dbms.logs.query.time_logging_enabled</code>	<code>false</code>	Log detailed time information for the executed queries being logged. Requires <code>dbms.track_query_cpu_time` =true`</code> .

Example 72. Configure for simple query logging

In this example we set query logging to `INFO`, but leave all other query log parameters 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 73. Configure for query logging with more details

In this example we turn query logging on, and also enable some additional logging.

```
dbms.logs.query.parameter_logging_enabled=true
dbms.logs.query.time_logging_enabled=true
dbms.logs.query.allocation_logging_enabled=true
dbms.logs.query.page_logging_enabled=true
```

Below is an example of the query log with these configuration parameters enabled:

```
2017-11-22 12:38 ... INFO 3 ms: bolt-session bolt johndoe neo4j-javascript/1.4.1
...
2017-11-22 22:38 ... INFO 61 ms: (planning: 0, cpu: 58, waiting: 0) - 6164496 B - 0 page hits, 1
page faults ...
2017-11-22 12:38 ... INFO 78 ms: (planning: 40, cpu: 74, waiting: 0) - 6347592 B - 0 page hits, 0
page faults ...
2017-11-22 12:38 ... INFO 44 ms: (planning: 9, cpu: 25, waiting: 0) - 1311384 B - 0 page hits, 0
page faults ...
2017-11-22 12:38 ... INFO 6 ms: (planning: 2, cpu: 6, waiting: 0) - 420872 B - 0 page hits, 0 page
faults - ...
```

Attach metadata to a query

It is possible to attach metadata to a query and have it printed in the query log, using the built-in procedure `tx.setMetaData`. This is typically done programmatically, but can be illustrated as follows, using `cypher-shell`.

Example 74. Attach metadata to a query

Start a transaction and call `tx.setMetaData` with a list of meta data.

```
neo4j> :begin
neo4j# CALL tx.setMetaData({ User: 'jsmith', AppServer: 'app03.dc01.company.com'});
neo4j# CALL dbms.procedures() YIELD name RETURN COUNT(name);
COUNT(name)
39
neo4j# :commit
```

Below are the corresponding results in the query log:

```
... CALL tx.setMetaData({ User: 'jsmith', AppServer: 'app03.dc01.company.com'}); - {} - {}
... CALL dbms.procedures() YIELD name RETURN COUNT(name); - {} - {User: 'jsmith', AppServer:
'app03.dc01.company.com'}
```

13.2.8. 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:

Transaction log configuration	Default value	Description
<code>dbms.directories.transaction.logs.root</code>	<code>transactions</code>	Root location where Neo4j will store transaction logs for configured databases.
<code>dbms.tx_log.preallocate</code>	<code>true</code>	Specify if Neo4j should try to preallocate logical log file in advance.
<code>dbms.tx_log.rotation.retention_policy</code>	<code>7 days</code>	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.
<code>dbms.tx_log.rotation.size</code>	<code>250M</code>	Specifies at which file size the logical log will auto-rotate. Minimum accepted value is <code>128K</code> (128 KiB).

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

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.

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.

Log retention

There are several different means of controlling the amount of transaction logs that are kept, using the parameter `dbms.tx_log.rotation.retention_policy`.

This parameter can be configured in different ways:

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

If this parameter is set to `true`, transaction logs will be kept indefinitely. 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.

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

If this parameter is set to `false`, only the most recent non-empty log will be kept. 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.

- `dbms.tx_log.rotation.retention_policy=<amount> <type>`

Type	Description	Example
files	Number of 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	Number of transactions to keep	"250k txs" or "5M txs"
hours	Keep logs which contains any transaction committed within N hours from current time	"10 hours"
days	Keep logs which contains any transaction committed within N days from current time	"50 days"

Example 75. 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
```

- Keep only the most recent non-empty log:

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

- 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
```

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
<code>dbms.checkpoint.interval.time</code>	<code>15m</code>	Configures the time interval between check-points.
<code>dbms.checkpoint.interval.tx</code>	<code>100000</code>	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 that disabling the IOPS limit can cause transaction processing to slow down a bit.

13.3. Query management

This section describes facilities for query management.

This section describes the following:

- [List all running queries](#)
- [List all active locks for a query](#)
- [Terminate multiple queries](#)
- [Terminate a single query](#)

13.3.1. List all running queries

An [administrator](#) is able to view all queries that are currently executing within the instance. Alternatively, the [current user](#) may view all of their own currently-executing queries.

Syntax:

```
CALL dbms.listQueries()
```

Returns:

Name	Type	Description
queryId	String	This is the ID of the query.
username	String	This is the username of the user who is executing the query.
metaData	Map	This is any metadata associated with the transaction.
query	String	This is the query itself.
parameters	Map	This is a map containing all the parameters used by the query.
planner	String	Planner used by the query
runtime	String	Runtime used by the query
indexes	List	Indexes used by the query
startTime	String	This is the time at which the query was started.
elapsedTime	String	Deprecated: Use elapsedTimeMillis instead. This is the time that has elapsed since the query was started.
connectionDetails	String	Deprecated: Use requestScheme , clientAddress , requestUri . These are the connection details pertaining to the query.
protocol	String	The protocol used by connection issuing the query.
connectionId	String	The ID of the connection issuing the query. This field will be null if the query was issued using embedded API.
clientAddress	String	The client address of the connection issuing the query.

Name	Type	Description
requestUri	String	The request URI used by the client connection issuing the query.
status	String	Status of the executing query.
resourceInformation	Map	Status of the executing query.
activeLockCount	Integer	Count of active locks held by transaction executing the query.
elapsedTimeMillis	Integer	This is the time in milliseconds that has elapsed since the query was started.
cpuTimeMillis	Integer	CPU time in milliseconds that has been actively spent executing the query. This field will be null unless the config parameter <code>dbms.track_query_cpu_time</code> is set to true.
waitTimeMillis	Integer	Wait time in milliseconds that has been spent waiting to acquire locks.
idleTimeMillis	Integer	Idle time in milliseconds. This field will be null unless the config parameter <code>dbms.track_query_cpu_time</code> is set to true.
allocatedBytes	Integer	Estimated bytes allocated for the executing query. For memory-intense or long-running queries the value may be larger than the current memory usage. This field will be null unless the config parameter <code>dbms.track_query_allocation</code> is set to true.
pageHits	Integer	Page hits occurred during the execution.
pageFaults	Integer	Page faults occurred during the execution.
database	String	This is the name of the database the query is executing against.

Example 76. Viewing queries that are currently executing

The following example shows that the user `alwood` is currently running `dbms.listQueries()` yielding specific variables, namely `queryId`, `username`, `query`, `elapsedTimeMillis`, `requestUri`, `status`, and `database`.

```
CALL dbms.listQueries() YIELD queryId, username, query, elapsedTimeMillis, requestUri, status, database
```

queryId	username	query	elapsedTimeMillis	requestUri
status	database			
"query-33"	"alwood"	"CALL dbms.listQueries() YIELD 1 127.0.0.1:7687" "running" "myDb" queryId, username, query, ela psedTime, requestUri, status, database"	1	
1 row				

13.3.2. 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
<code>mode</code>	String	Lock mode corresponding to the transaction.
<code>resourceType</code>	String	Resource type of the locked resource
<code>resourceId</code>	Integer	Resource id of the locked resource .

Example 77. 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

13.3.3. Terminate multiple queries

An [administrator](#) is able to terminate within the instance all transactions executing a query with any of the given query IDs. Alternatively, the [current user](#) may terminate all of their own transactions executing a query with any of the given query IDs.

Syntax:

```
CALL dbms.killQueries(queryIds)
```

Arguments:

Name	Type	Description
<code>ids</code>	List<String>	This is a list of the IDs of all the queries to be terminated.

Returns:

Name	Type	Description
<code>queryId</code>	String	This is the ID of the terminated query.
<code>username</code>	String	This is the username of the user who was executing the (now terminated) query.

Name	Type	Description
message	String	A message stating whether the query was successfully found.

Example 78. Terminating multiple queries

The following example shows that the administrator has terminated the queries with IDs `query-378` and `query-765`, started by the users `joesmith` and `annebrown`, respectively.

This command can target queries from multiple databases at the same time. In this example, `joesmith` ran his query against `joeDb` and `annebrown` ran hers against `anneDb`.

```
CALL dbms.killQueries(['query-378', 'query-765'])
```

```
+-----+
| queryId | username   | message      |
+-----+
| "query-378" | "joesmith" | "Query found" |
| "query-765" | "annebrown" | "Query found" |
+-----+
2 rows
```

13.3.4. Terminate a single query

An `administrator` is able to terminate within the instance any transaction executing the query whose ID is provided. Alternatively, the `current user` may terminate their own transaction executing the query whose ID is provided.

Syntax:

```
CALL dbms.killQuery(queryId)
```

Arguments:

Name	Type	Description
<code>id</code>	String	This is the ID of the query to be terminated.

Returns:

Name	Type	Description
<code>queryId</code>	String	This is the ID of the terminated query.
<code>username</code>	String	This is the username of the user who was executing the (now terminated) query.
<code>message</code>	String	A message stating whether the query was successfully found.

Example 79. Terminating a single query

The following example shows that the user `joesmith` has terminated his query with the ID `query-502`.

```
CALL dbms.killQuery('query-502')
```

```
+-----+  
| queryId | username | message |  
+-----+  
| "query-502" | "joesmith" | "Query found" |  
+-----+  
1 row
```

The following example shows the output when trying to kill a query with an ID that does not exist.

```
CALL dbms.killQuery('query-502')
```

```
+-----+  
| queryId | username | message |  
+-----+  
| "query-502" | "n/a" | "No Query found with this id" |  
+-----+  
1 row
```

13.4. Transaction management

This section describes facilities for transaction management.

This section describes the following:

- [Configure transaction timeout](#)
- [Configure lock acquisition timeout](#)
- [List all running transactions](#)

13.4.1. Configure transaction timeout

It is possible to configure Neo4j to terminate transactions whose execution time has exceeded the configured timeout. To enable this feature, set `dbms.transaction.timeout` to some positive time interval value denoting the default transaction timeout. Setting `dbms.transaction.timeout` to `0` — which is the default value — disables the feature.

Example 80. Configure transaction timeout

Set the timeout to ten seconds.

```
dbms.transaction.timeout=10s
```

Configuring transaction timeout will have no effect on transactions executed with custom timeouts (via the Java API), as a custom timeout will override the value set for `dbms.transaction.timeout`.

The *transaction timeout* feature is also known as the *transaction guard*.

13.4.2. Configure lock acquisition timeout

An executing transaction may get stuck while waiting for some lock to be released by another transaction. A transaction in such state is not desirable, and in some cases it is better for the transaction to instead give up and fail.

To enable this feature, set `dbms.lock.acquisition.timeout` to some positive time interval value 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.

Example 81. Configure lock acquisition timeout

Set the timeout to ten seconds.

```
dbms.lock.acquisition.timeout=10s
```

13.4.3. List all running transactions

An [administrator](#) is able to view all transactions that are currently executing within the instance. Alternatively, the [current user](#) may view all of their own currently-executing transactions.

Syntax:

```
CALL dbms.listTransactions()
```

Returns:

Name	Type	Description
<code>transactionId</code>	String	This is the ID of the transaction.
<code>username</code>	String	This is the username of the user who is executing the transaction.
<code>metaData</code>	Map	This is any metadata associated with the transaction.
<code>startTime</code>	String	This is the time at which the transaction was started.
<code>protocol</code>	String	The protocol used by connection issuing the transaction.
<code>connectionId</code>	String	The ID of the connection issuing the transaction. This field will be null if the transaction was issued using embedded API.
<code>clientAddress</code>	String	The client address of the connection issuing the transaction.
<code>requestUri</code>	String	The request URI used by the client connection issuing the transaction.
<code>currentQueryId</code>	String	This is the ID of the current query executed by transaction.
<code>currentQuery</code>	String	This is the current query executed by transaction.

Name	Type	Description
activeLockCount	Integer	Count of active locks held by transaction.
status	String	Status of the executing transaction.
resourceInformation	Map	Information about what transaction is waiting for when it is blocked.
elapsedTimeMillis	Integer	This is the time in milliseconds that has elapsed since the transaction was started.
cpuTimeMillis	Integer	CPU time in milliseconds that has been actively spent executing the transaction.
waitTimeMillis	Integer	Wait time in milliseconds that has been spent waiting to acquire locks.
idleTimeMillis	Integer	Idle time in milliseconds.
allocatedBytes	Integer	Number of bytes allocated so far by the transaction. This column is deprecated in favor of estimatedUsedHeapMemory .
allocatedDirectBytes	Integer	Direct bytes used by the executing transaction.
pageHits	Integer	Page hits occurred during the execution.
pageFaults	Integer	Page faults occurred during the execution.
database	String	This is the name of the database the transaction is executing against.
estimatedUsedHeapMemory	Integer	This is the current estimated heap usage of the transaction, in bytes.

Example 82. Viewing transactions that are currently executing

The following example shows that the user '**alwood**' is currently running `dbms.listTransactions()`. The procedure call yields specific information about the running transaction, namely `transactionId`, `username`, `currentQuery`, `elapsedTimeMillis`, `requestUri`, and `status`.

```
CALL dbms.listTransactions() YIELD transactionId, username, currentQuery, elapsedTimeMillis,
requestUri, status
```

transactionId	username	currentQuery	elapsedTimeMillis
requestUri	status		
"myDb-transaction-22" "127.0.0.1:7687"	"alwood" "Running"	"CALL dbms.listTransactions() YIELD transactionId, username, currentQuery elapsedTime, requestUri, status"	"1"

1 row

13.5. Connection management

This section describes facilities for connection management.

This section describes the following:

- [List all network connections](#)
- [Terminate multiple network connections](#)
- [Terminate a single network connection](#)

13.5.1. 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.

Syntax:

```
CALL dbms.listConnections()
```

Returns:

Name	Type	Description
<code>connectionId</code>	String	This is the ID of the network connection.
<code>connectTime</code>	String	This is the time at which the connection was started.
<code>connector</code>	String	Name of the connector that accepted the connection.
<code>username</code>	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.
<code>userAgent</code>	String	Name of the software that is connected. This information is extracted from the User-Agent request header for HTTP and HTTPS connections. It is available natively for Bolt connections which supply the agent string in an initialization message.
<code>serverAddress</code>	String	The server address this connection is connected to.
<code>clientAddress</code>	String	The client address of the connection.

Example 83. 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"	"clientAddress"	"status"	"connector"	"username"	"userAgent"
"bolt-21"	"2018-10-10T12:11:42.276Z"	"127.0.0.1:53929"	"Running"	"bolt"	"alwood"	"neo4j-java/1.6.3"
"http-11"	"2018-10-10T12:37:19.014Z"	"macOS 10.13; rv:62.0) Gecko/20100101 Firefox/62.0"	"Running"	"http"	null	"Mozilla/5.0 (Macintosh; Intel
						"127.0.0.1:54118"

2 rows

13.5.2. 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
<code>ids</code>	List<String>	This is a list of the IDs of all the connections to be terminated.

Returns:

Name	Type	Description
<code>connectionId</code>	String	This is the ID of the terminated connection.
<code>username</code>	String	This is the username of the user who initiated the (now terminated) connection.
<code>message</code>	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 84. 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

13.5.3. 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
<code>id</code>	String	This is the ID of the connection to be terminated.

Returns:

Name	Type	Description
<code>connectionId</code>	String	This is the ID of the terminated connection.
<code>username</code>	String	This is the username of the user who initiated the (now terminated) connection.
<code>message</code>	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 85. 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

13.6. Background job management

This section describes facilities for listing both active and failed 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.

13.6.1. Listing active background jobs

An [administrator](#) can list background jobs active on an instance:

Syntax:

```
CALL dbms.scheduler.jobs()
```

Returns:

Name	Type
jobId	String
group	String
submitted	String
database	String
submitter	String
description	String
type	String
scheduledAt	String

Name	Type
<code>period</code>	String
A period of a <code>PERIODIC</code> job, in format <code>hh:mm:ss.sss</code> .	
<code>state</code>	String
A state of the job. Since this procedure lists only active jobs, they can be either in <code>SCHEDULED</code> or <code>EXECUTING</code> state. <code>SCHEDULED</code> state is applicable only to <code>DELAYED</code> or <code>PERIODIC</code> jobs, and means that the job is scheduled for a given time in the future.	
<code>currentStateDescription</code>	String
If a job supports reposting its progress, the progress will be reported in this column in a free-form format, specific for each job.	

13.6.2. Listing failed job executions

An [administrator](#) can list job executions failed on an instance:

Syntax:

```
CALL calling dbms.scheduler.failedJobs()
```

Returns:

Name	Type
<code>jobId</code>	String
ID of the failed job.	
<code>group</code>	String
A job is a member of a job group. For example, <code>INDEX_POPULATION</code> , <code>LOG_ROTATION</code> or <code>RAFT_SERVER</code> .	
<code>database</code>	String
Jobs can have either a database or a DBMS scope:	
<ul style="list-style-type: none"> For database, this column will display the name of the database. For DBMS, this column will be blank. 	
<code>submitter</code>	String
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.	

Name	Type
<code>description</code>	String
A short description of a job that, unlike <code>currentStateDescription</code> , does not change during the running of the job.	
<code>type</code>	String
Type of the job. The values can be <code>IMMEDIATE</code> , <code>DELAYED</code> or <code>PERIODIC</code> .	
<code>submitted</code>	String
A timestamp for when the job was submitted, in ISO-8601 format.	
<code>executionStart</code>	String
A timestamp for when the failed execution started, in ISO-8601 format.	
<code>failureTime</code>	String
A timestamp for when the execution failed, in ISO-8601 format.	
<code>failureDescription</code>	String
A short description of the failure. If the failure description is insufficient, more information can be found in logs.	

13.7. Monitoring a Causal Cluster

This section covers additional facilities available for monitoring a Neo4j 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

13.7.1. Procedures for monitoring a Causal Cluster

This section covers procedures for monitoring a Neo4j Causal Cluster.

A number of procedures are available that provide information about a cluster. This section describes the following:

- [Find out the role of a cluster member](#)
- [Gain an overview over the instances in the cluster](#)
- [Get routing recommendations](#)

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
<code>databaseName</code>	String	The name of the database to get the cluster role for.

Returns:

Name	Type	Description
<code>role</code>	String	This is the role of the current instance, which can be <code>LEADER</code> , <code>FOLLOWER</code> , or <code>READ_REPLICA</code> .

Considerations:

- While this procedure is useful in and of itself, it serves as basis for more powerful monitoring procedures.

Example 86. 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
<code>id</code>	String	This is id of the instance.
<code>addresses</code>	List	This is a list of all the addresses for the instance.
<code>groups</code>	List	This is a list of all the server groups which an instance is part of.
<code>databases</code>	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 87. Get an overview of the cluster

The following example shows how to explore the cluster topology.

```
CALL dbms.cluster.overview()
```

<code>id</code>	<code>addresses</code>	<code>groups</code>	<code>databases</code>
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}
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 88. 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"
    }
  ]
}
```

13.7.2. Endpoints for status information

This section describes HTTP endpoints for monitoring the health of a Neo4j Causal Cluster.

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.

The section includes:

- [Adjusting security settings for Causal Clustering endpoints](#)
- [Unified endpoints](#)

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/<dbname>/cluster/writable` — Used to direct `write` traffic to specific instances.
- `/db/<dbname>/cluster/read-only` — Used to direct `read` traffic to specific instances.
- `/db/<dbname>/cluster/available` — Available for the general case of directing arbitrary request types to instances that are available for processing read transactions.
- `/db/<dbname>/cluster/status` — Gives a detailed description of this instance's view of its own status within the cluster. See [Status endpoint](#) for further details.

Every endpoint targets a specific database with its own Raft-group. The `databaseName` path parameter represents the name of the database. By default, a fresh Neo4j installation will have endpoints for two databases `system` and `neo4j`:

```
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 60. Unified HTTP endpoint responses

Endpoint	Instance state	Returned code	Body text
<code>/db/<dbname>/cluster/writable</code>	Leader	<code>200 OK</code>	<code>true</code>
	Follower	<code>404 Not Found</code>	<code>false</code>
	Read Replica	<code>404 Not Found</code>	<code>false</code>
<code>/db/<dbname>/cluster/read-only</code>	Leader	<code>404 Not Found</code>	<code>false</code>
	Follower	<code>200 OK</code>	<code>true</code>
	Read Replica	<code>200 OK</code>	<code>true</code>

Endpoint	Instance state	Returned code	Body text
<code>/db/<dbname>/cluster/available</code>	Leader	<code>200 OK</code>	<code>true</code>
	Follower	<code>200 OK</code>	<code>true</code>
	Read Replica	<code>200 OK</code>	<code>true</code>
<code>/db/<dbname>/cluster/status</code>	Leader	<code>200 OK</code>	JSON - See Status endpoint for details.
	Follower	<code>200 OK</code>	JSON - See Status endpoint for details.
	Read Replica	<code>200 OK</code>	JSON - See Status endpoint for details.

Example 89. 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 endpoint

The status endpoint, available at `/db/<dbname>/cluster/status`, is to be used to assist with [rolling upgrades](#).

Typically, you will want to have some guarantee that a Core is safe to shutdown before removing it from a cluster. The status endpoint provides the following information in order to help resolve such issues:

Example 90. 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 61. 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.
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	Reflects that the local database of this member has not encountered a critical error preventing it from writing locally.
memberId	string	no	30edc1c4-519c-4030-8348-7cb7af44f591	Every member in a cluster has its 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 window 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 62. Measured values, accessed via the status endpoint

Name of check	Method of calculation	Description
<code>allServersAreHealthy</code>	Every Core's status endpoint indicates <code>isHealthy==true</code> .	We want to make sure the data across the entire cluster is healthy. Whenever any Cores are false that indicates a larger problem.
<code>allVotingSetsAreEqual</code>	For any 2 Cores (A and B), status endpoint A's <code>votingMembers==</code> status endpoint B's <code>votingMembers</code> .	When the voting begins, all the Cores are equal to each other, and you know all members agree on membership.
<code>allVotingSetsContainAtLeastTargetCluster</code>	For all Cores (<code>S</code>), excluding Core Z (to be switched off), every member in <code>S</code> contains <code>S</code> in their voting set. Membership is determined by using the <code>memberId</code> and <code>votingMembers</code> 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).
<code>hasOneLeader</code>	For any 2 Cores (A and B), <code>A.leader == B.leader && leader!=null</code> .	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.
<code>noMembersLagging</code>	For Core A with <code>lastAppliedRaftIndex = min</code> , and Core B with <code>lastAppliedRaftIndex = max</code> , <code>B.lastAppliedRaftIndex - A.lastAppliedRaftIndex < raftIndexLagThreshold</code> .	If there is a large difference in the applied indexes between Cores, then it could be dangerous to switch off a Core.

For more information on rolling upgrades for causal clusters, see [Rolling upgrade](#).

13.8. Monitoring individual database states

This section covers the use of `SHOW DATABASES`, and other related Cypher commands.

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.

13.8.1. 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
<code>name</code>	String	The human-readable name of the database.
<code>address</code>	String	The bolt address of the Neo4j instance hosting the database.

Name	Type	Description
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 <code>requestedStatus</code> , if any.
default	String	Whether this database is the default for this DBMS.

Example 91. Listing databases in standalone Neo4j

When executing `SHOW DATABASES` against a standalone instance of Neo4j, you should see output like the following:

name	address	role	requestedStatus	currentStatus	error	default
"neo4j"	"localhost:7687"	"standalone"	"online"	"online"	""	true
"system"	"localhost:7687"	"standalone"	"online"	"online"	""	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	address	role	requestedStatus	currentStatus	error	default
"neo4j"	"localhost:7687"	"standalone"	"online"	"online"	""	true
"system"	"localhost:7687"	"standalone"	"online"	"online"	""	false
"foo"	"localhost:7687"	"standalone"	"online"	"offline"	"An error occurred! Unable to start database ..."	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 92. Listing databases in a Neo4j Causal Cluster

When running `SHOW DATABASES` against a Neo4j Causal Cluster you might see output like the following:

name	address	role	requestedStatus	currentStatus	error	default
"neo4j"	"localhost:20031"	"follower"	"online"	"online"	""	true
"neo4j"	"localhost:20010"	"follower"	"online"	"online"	""	true
"neo4j"	"localhost:20005"	"leader"	"online"	"online"	""	true

name	address	role	requestedStatus	currentStatus	error	default
"neo4j"	"localhost:20034"	"read_replica"	"online"	"online"	""	true
"system"	"localhost:20031"	"follower"	"online"	"online"	""	false
"system"	"localhost:20010"	"follower"	"online"	"online"	""	false
"system"	"localhost:20005"	"leader"	"online"	"online"	""	false
"system"	"localhost:20034"	"read_replica"	"online"	"online"	""	false
"foo"	"localhost:20031"	"leader"	"online"	"online"	""	false
"foo"	"localhost:20010"	"follower"	"online"	"online"	""	false
"foo"	"localhost:20005"	"follower"	"online"	"online"	""	false
"foo"	"localhost:20034"	"read_replica"	"online"	"online"	""	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	address	role	requestedStatus	currentStatus	error	default
"neo4j"	"localhost:20031"	"follower"	"online"	"online"	""	true
"neo4j"	"localhost:20010"	"follower"	"online"	"online"	""	true
"neo4j"	"localhost:20005"	"leader"	"online"	"online"	""	true
"neo4j"	"localhost:20034"	"read_replica"	"online"	"online"	""	true
"system"	"localhost:20031"	"follower"	"online"	"online"	""	false
"system"	"localhost:20010"	"follower"	"online"	"online"	""	false
"system"	"localhost:20005"	"leader"	"online"	"online"	""	false
"system"	"localhost:20034"	"read_replica"	"online"	"online"	""	false
"foo"	"localhost:20031"	"unknown"	"online"	"initial"	"An error occurred! Unable to start database ..."	false
"foo"	"localhost:20010"	"leader"	"online"	"online"	""	false
"foo"	"localhost:20005"	"follower"	"online"	"online"	""	false

<code>name</code>	<code>address</code>	<code>role</code>	<code>requestedStatus</code>	<code>currentStatus</code>	<code>error</code>	<code>default</code>
"foo"	"localhost:20034"	"unknown"	"online"	"initial"	"An error occurred! Unable to start database ..."	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".

13.8.2. 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
<code>databaseName</code>	String	The name of the database whose status to report

Returns:

Name	Type	Description
<code>name</code>	String	The human-readable name of the database.
<code>address</code>	String	The bolt address of the Neo4j instance hosting the database.
<code>role</code>	String	The cluster role which the Neo4j instance fulfills for this database.
<code>requestedStatus</code>	String	The state that an operator has requested the database to be in.

Name	Type	Description
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 <code>requestedStatus</code> , if any.
default	String	Whether this database is the default for this DBMS.

Example 93. Listing statuses for database foo

When running `SHOW DATABASE foo` in a Neo4j Causal Cluster, you should see output like the following:

name	address	role	requestedStatus	currentStatus	error	default
"foo"	"localhost:20031"	"unknown"	"online"	"initial"	"An error occurred! Unable to start database ..."	false
"foo"	"localhost:20010"	"leader"	"online"	"online"	""	false
"foo"	"localhost:20005"	"follower"	"online"	"online"	""	false
"foo"	"localhost:20034"	"unknown"	"online"	"initial"	"An error occurred! Unable to start database ..."	false

Chapter 14. Performance

This chapter describes factors that affect operational performance, and how to tune Neo4j for optimal throughput.

The topics described in this chapter are:

- [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 section describes the different aspects of Neo4j memory configuration and use.

14.1.1. Overview

The RAM of the Neo4j server has a number of usage areas, with some sub-areas:

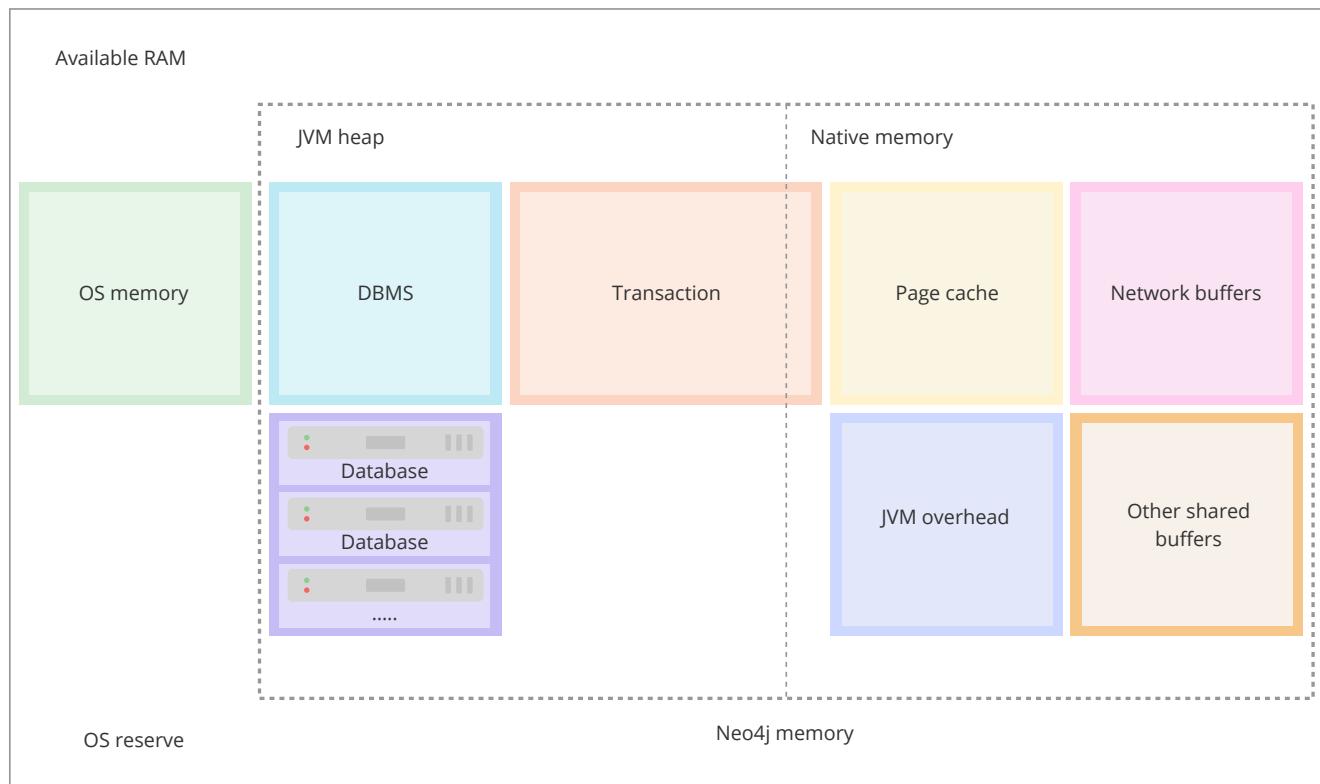


Figure 13. Neo4j memory management

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 has a heap that is the runtime data area from which memory for all class instances and arrays is allocated. Heap storage for objects is reclaimed by an automatic storage management system (known as a garbage collector or GC).

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.

14.1.2. 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 94. 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.

14.1.3. 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 95. 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 96. 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.

$$35.4\text{GB} - 35\text{GB} = 0.4\text{GB} * 100 = 40\text{GB}$$

4. Multiply that number by 1.2 to size up the result, and allow for 20% growth.

$$\text{dbms.memory.pagecache.size} = 1.2 * (40\text{GB}) = 48\text{GB}$$

5. Configure the page cache by adding the following to `neo4j.conf`:

```
dbms.memory.pagecache.size=48G
```

14.1.4. 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`.

14.2. Index configuration

How to configure indexes to enhance performance in search, and to enable full-text search.

14.2.1. Introduction

In Neo4j there are two different index types: [b-tree](#) and [full-text](#).

B-tree indexes can be created and dropped using Cypher. Users typically do not have to know about the index in order to use it, since Cypher's query planner decides which index to use in which situation. B-tree indexes are good at exact look-ups on all types of values, and range scans, full scans, and prefix searches.

For details on the configuration aspects of b-tree indexes, see [B-tree indexes](#).

Full-text indexes differ from b-tree indexes, in that they are optimised for indexing and searching text. They are used for queries that demand an understanding of language, and they only index string data. They must also be queried explicitly via procedures, as Cypher will not make plans that rely on them.

An example of a use case for full-text indexes is parsing a book for a certain term, and taking advantage of the knowledge that the book is written in a certain language. The use of an *analyzer* for that language will, among other things, enable you to exclude words that are not relevant for the search (for example "if" and "and"), and include 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 would enable someone to find all emails that are sent from, or to, or mentions, an email account.

In contrast to b-tree indexes, full-text indexes are created, queried, and dropped using built-in procedures. The use of full-text indexes do require a familiarity with how the indexes operate.

For details on the configuration aspects of full-text indexes, see [Full-text indexes](#).

For details on creating, querying and dropping full-text indexes, see [Cypher Manual □ Indexes to support full-text search](#).

The type of an index can be identified according to the table below:

Index type	Procedure	Core API
B-tree index	<code>db.indexes#BTREE</code>	<code>org.neo4j.graphdb.schema.IndexType#BTREE</code>
Full-text index	<code>db.indexes#FULLTEXT</code>	<code>org.neo4j.graphdb.schema.IndexType#FULLTEXT</code>

14.2.2. B-tree indexes

B-tree indexes 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` will be used.

For more information on the different index types, refer to [Cypher Manual □ Indexes for search performance](#) and [Cypher Manual □ Indexes to support full-text search](#).

Limitations

In this section a few limitations for b-tree indexes are described, together with suggested workarounds.

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 will not be able to do an optimized search as per queries that use `STARTS WITH`, `=`, and `<>`. Instead, the index result will be a stream of an index scan with filtering.

In the future, `ENDS WITH` and `CONTAINS` queries will be supported with full-text indexes, but for now the index provider `lucene+native-3.0` can be used instead. Please note that `lucene+native-3.0` only has support for `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 around 8kB.

If a transaction reaches the key size limit for one or more of its changes, that transaction will fail before committing any changes. If the limit is reached during index population, the resulting index will be in a failed state, and as such will not be usable for any queries.

If this is an issue, you can use the index provider `lucene+native-3.0` instead. This provider has a key size limit for single property strings of around 32kB.

Workarounds to address limitations

To workaround problems with key size, or performance issues related to `ENDS WITH` or `CONTAINS`, you can use the index provider `lucene+native-3.0`. This only works for single-property string indexes.

This can be done using either of the following methods:

Option 1. Use `OPTIONS` clause in create command (recommended)

The Cypher commands for index creation, unique property constraint creation, and node key creation contains an optional `OPTIONS` clause. This clause can be used to specify index provider.

For details on indexes, see [Cypher Manual □ Indexes for search performance](#). For details on constraints, see [Cypher manual □ Constraints](#).

Option 2. Use a built-in procedure

Please note that this option uses built-in procedures that have been deprecated, and will be removed in a future release. These have been replaced with the Cypher commands in Option 1.

The built-in procedures `db.createIndex`, `db.createUniquePropertyConstraint`, and `db.createNodeKey` can be used to specify index provider on index creation, unique property constraint creation, and

node key creation.

For details on constraints, see [Cypher manual](#) [Constraints](#), and for more information on built-in procedures, see [\[neo4j-procedures\]](#).

Option 3. Change the config

Please 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 is using.

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 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](#) [Indexes for search performance](#) and [Cypher manual](#) [Constraints](#).

Index migration

When upgrading a 3.5 store to 4.2.0, all indexes will be upgraded to the latest index version, and rebuilt automatically, with the exception for the indexes that were previously using Lucene for single-property strings. They will be upgraded to a fallback version which still use Lucene for those properties. Please note that they will still need to be rebuilt.

The table below shows the migration mapping:

Index provider in 3.5	Index provider in 4.2.0
<code>native-btree-1.0</code>	<code>native-btree-1.0</code>
<code>lucene+native-2.0</code>	<code>native-btree-1.0</code>
<code>lucene+native-1.0</code>	<code>lucene+native-3.0</code>
<code>lucene-1.0</code>	<code>lucene+native-3.0</code>

The caching of indexes takes place in different memory areas for different index providers. See [Memory configuration](#). It can be useful to run `neo4j-admin memrec --database` before and after the rebuilding of indexes, and adjust memory settings in accordance with the findings.

Between 3.5 and 4.2.0, some changes have been made in how large a property that a b-tree index can handle. These changes are only relevant for indexes that use the index providers `lucene-1.0` or `lucene+native-1.0` in 3.5, and hold large strings or large arrays.

Property size limits for index providers in 3.5:

Index and property type	<code>lucene-1.0</code>	<code>lucene+native-1.0</code>	<code>lucene+native-2.0</code>	<code>native-btree-1.0</code>
Single property strings	~32kB	~32kB	~4kB	~4kB

Index and property type	lucene-1.0	lucene+native-1.0	lucene+native-2.0	native-btree-1.0
Single property arrays	~32kB	~32kB	~4kB	~4kB
Multiple properties	~32kB per property	~32kB per property	~4kB for properties combined	~4kB for properties combined

Property size limits for index providers in 4.2.0:

Index and property type	lucene+native-3.0	native-btree-1.0
Single property strings	~32kB	~8kB
Single property arrays	~8kB	~8kB
Multiple properties	~8kB for properties combined	~8kB for properties combined

There is no workaround in 4.2.0 to index arrays or multiple properties larger than ~8kB. For more information about what happens when this property size limit is reached, please see [Limitations on key size](#).

Procedures to create index and index backed constraint

Indexes and constraints are best created through [Cypher](#), but when these indexes or constraints need to be more specifically configured than what is possible through Cypher, then you can use the procedures described in the example below.

Example 97. Example of procedures to create index and index backed constraint

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.

Use `db.createIndex` procedure to create an index:

```
CALL db.createIndex("MyIndex", ["Person"], ["name"], "native-btree-1.0", {`spatial.cartesian.max`: [100.0,100.0], `spatial.cartesian.min`: [-100.0,-100.0]})
```

If a settings map is not provided, the settings will be picked up from the [Neo4j config file](#), the same way as when creating an index or constraint through Cypher.

```
CALL db.createIndex("MyIndex", ["Person"], ["name"], "native-btree-1.0")
```

Use `db.createUniquePropertyConstraint` to create a node property uniqueness constraint (the example is without settings map, left out for abbreviation):

```
CALL db.createUniquePropertyConstraint("MyIndex", ["Person"], ["name"], "native-btree-1.0")
```

Use `db.createNodeKey` to create node key constraint (the example is without settings map, left out for abbreviation):

```
CALL db.createNodeKey("MyIndex", ["Person"], ["name"], "native-btree-1.0")
```

14.2.3. Full-text indexes

Full-text indexes are powered by the [Apache Lucene](#) indexing and search library. A full-text index enables you to write queries that matches within the *contents* of indexed string properties. A full description on 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:

`dbms.index.fulltext.default_analyzer`

The name of the analyzer that the full-text indexes should use by default. This setting only has effect when a full-text index is created, and will be remembered as an index-specific setting from then on.

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`, which is 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 will be remembered as an index-specific setting from then on.

Indexes are normally fully consistent, and the committing of a transaction does not return until both the store and the indexes have been updated. Eventually consistent full-text indexes, on the other hand, are not updated as part of commit, but instead have their updates queued up and applied in a background thread. This means that there can be a short delay between committing a change, and that change becoming visible via any eventually consistent full-text indexes. This delay is just an artifact of the queueing, and will usually be quite small since eventually consistent indexes are updated "as soon as possible".

By default, this is turned off, 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, then committing transactions will block and wait until there is more room in the queue, before adding more updates to it.

This setting applies to all eventually consistent full-text indexes, and they all use the same queue. The maximum queue length must be at least 1 index update, and must be no more than 50 million due to heap space usage considerations.

The default maximum queue length is 10.000 index updates.

14.3. Tuning of the garbage collector

This section discusses the effect of the Java Virtual Machine's garbage collector with regards to Neo4j performance.

The heap is separated into an *old generation* and a *young generation*. 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???` 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 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 63. *neo4j.conf* JVM tuning properties

Property Name	Meaning
<code>dbms.memory.heap.initial_size</code>	initial heap size (in MB)
<code>dbms.memory.heap.max_size</code>	maximum heap size (in MB)
<code>dbms.jvm.additional</code>	additional literal JVM parameter

14.4. Bolt thread pool configuration

This section discusses the thread pool infrastructure built into Bolt connectors, and how it can be configured.

The Bolt connector is backed by a thread pool on the server side. The thread pool is constructed as part of the server startup process.

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 Manual](#).

14.4.2. Configuration options

The following configuration options are available for configuring the Bolt connector:

Table 64. Thread pool options

Option name	Default	Description
<code>dbms.connector.bolt.thread_pool_min_size</code>	5	The minimum number of threads that will always be up even if they are idle.
<code>dbms.connector.bolt.thread_pool_max_size</code>	400	The maximum number of threads that will be created by the thread pool.
<code>dbms.connector.bolt.thread_pool_keep_alive</code>	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 <code>dbms.connector.bolt.thread_pool_min_size</code> .

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 98. 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

This section covers Neo4j I/O behavior, and how to optimize for operations on disk.

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.

Often, recommended practice is to disable file and directory access time updates. This way, the file system won't 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 recommended as a first choice.

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.

To be able to achieve optimum performance, we do not recommend the use of NFS or NAS as database storage.

For more detailed and precise configuration for your operating system and hardware, please consult corresponding manuals.

14.6. Disks, RAM and other tips

This section provides an overview of performance considerations for disk and RAM when running Neo4j.

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.

14.6.1. Storage

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.

To achieve maximum performance, it is recommended to provide Neo4j with as much RAM as possible in order to avoid hitting the disk.

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 don't quite fit in memory. In this case, database access can have high latencies.

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.

Active page cache warmup

The 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 99. 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.

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 300, for instance, would thus only allow the checkpoint process to write at a rate of roughly 2.5 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

This section describes the configuration options that affect the gathering of statistics, and the replanning of query plans in the Cypher query engine.

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 available information about the database, such as schema information about which indexes and constraints exist in the database. Neo4j also uses statistical information about the database to optimize the execution plan.

For further details, please see [Cypher manual □ Query tuning](#) and [Cypher manual □ Execution plans](#).

The frequency of statistics gathering and the replanning of execution plans are described in the sections below.

14.7.1. Statistics

The statistical information that Neo4j keeps is:

1. The number of nodes having a certain label.
2. The number of relationships by type.
3. The number of relationships by type, ending or starting from a node with a specific label.
4. Selectivity per index.

Neo4j keeps the statistics up to date in two different ways. For label and relationship counts, the number is updated whenever you set or remove a label from a node. For indexes, however, Neo4j needs to scan the full index to produce the selectivity number. Since this is potentially a very time-consuming operation, these numbers are collected in the background when enough data on the index has been changed.

The following settings allow you to control whether statistics are collected automatically, and at which rate:

Parameter name	Default value	Description
dbms.index_sampling.background_enabled	true	Controls whether indexes will automatically be re-sampled when they have been updated enough. The Cypher query planner depends on accurate statistics to create efficient plans, so it is important it is kept up to date as the database evolves.
dbms.index_sampling.update_percentage	5	Controls the percentage of the index that has to have been updated before a new sampling run is triggered.

It is possible to manually trigger index sampling, using the following built-in procedures:

`db.resampleIndex()`

Trigger resampling of an index.

`db.resampleOutdatedIndexes()`

Trigger resampling of all outdated indexes.

Example 100. Manually trigger index resampling

The following example illustrates how to trigger a resampling of the index on the label `Person` and property `name`, by calling `db.resampleIndex()`:

```
CALL db.resampleIndex(":Person(name)");
```

The following example illustrates how to call `db.resampleOutdatedIndexes()` in order to trigger a resampling of all outdated indexes:

```
CALL db.resampleOutdatedIndexes();
```

14.7.2. Execution plans

Execution plans are cached and by default, will not be replanned until the statistical information used to produce the plan has changed. The following setting enables you to control how sensitive replanning should be to updates of the database:

Parameter name	Default value	Description
<code>cypher.statistics_divergence_threshold</code>	<code>0.75</code>	The threshold when a plan is considered stale. If any of the underlying statistics used to create the plan have changed more than this value, the plan will be considered stale and will be replanned. Change is calculated as <code>abs(a-b)/max(a,b)</code> . This means that a value of <code>0.75</code> requires the database to approximately quadruple in size before replanning occurs. A value of 0 means replan as soon as possible, with the soonest being defined by the parameter <code>cypher.min_replan_interval</code> , which 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.

It is possible to manually force the database to replan execution plans that are already in the cache, using the following built-in procedures:

`db.clearQueryCaches()`

Clears out all query caches, but does not change the database statistics.

```
CALL db.clearQueryCaches();
```

`db.prepareForReplanning()`

Completely recalculates all database statistics, so that they can be used for any subsequent query planning.

It triggers an index resampling and waits for it to complete, and after that it also clears all query caches. After this procedure has finished queries will be planned on empty caches using the latest database statistics.

```
CALL db.prepareForReplanning();
```

You can use Cypher replanning to specify whether you want to force a replan, even if the plan is valid according to the planning rules, or skip replanning entirely should you wish to use a valid plan that already exists.

For more information, see:

- [Cypher manual](#) □ [Cypher replanning](#)
- [Cypher manual](#) □ [Execution plans](#)
- [Procedures](#)

14.8. Space reuse

This section describes how Neo4j handles data deletion and storage space.

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.



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.



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` tool to create a defragmented copy of your database. The `copy` command creates an 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 101. 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      | address          | role       | requestedStatus | currentStatus | error |
| default   |                  |            |                |              |         |
+-----+
| "neo4j"    | "localhost:7687" | "standalone" | "offline"     | "offline"    | ""      |
| "neo4jcopy1" | "localhost:7687" | "standalone" | "online"      | "online"     | ""      |
| "system"    | "localhost:7687" | "standalone" | "online"      | "online"     | ""      |
+-----+
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. Tools

This chapter describes the Neo4j tools Neo4j Admin and Cypher Shell, and explains how to use commands to import data into a new database, dump and load an existing database, and how to check consistency.

This chapter comprises the following topics:

- [A description of the Neo4j Admin tool](#)
- [How to check the consistency of a Neo4j database using Neo4j Admin](#)
- [How to collect the most common information needed for remote assessments](#)
- [How to display information about a database store](#)
- [How to get an initial recommendation for Neo4j memory settings](#)
- [How to import data into Neo4j using the import tool
 - \[Syntax\]\(#\)
 - \[CSV header format\]\(#\)
 - \[Options\]\(#\)](#)
- [How to dump and load Neo4j databases using Neo4j Admin](#)
- [How to remove cluster state data from a Neo4j server.](#)
- [How to copy data from an existing database to a new database.](#)
- [How to use the Cypher Shell](#)

15.1. Neo4j Admin

Describes the Neo4j Admin tool.

15.1.1. Introduction

Neo4j Admin is the primary tool for managing your Neo4j instance. 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.

15.1.2. Syntax and commands

Syntax

Neo4j Admin is located in the `bin` directory and is invoked as:

```
neo4j-admin <command>
```

Commands

Functionality area	Command	Description
General	<code>help</code> <code>help <command></code>	Display help text. Display help text for <code><command></code>
	<code>check-consistency</code>	Check the consistency of a database. For details, see Consistency checker .
	<code>report</code>	Collect the most common information needed for remote assessments. For details, see Neo4j Admin report .
	<code>store-info</code>	Print information about a Neo4j database store. For details, see Display store information .
	<code>memrec</code>	Print Neo4j heap and pagecache memory settings recommendations. For details, see Memory recommendations .
	<code>import</code>	Import from a collection of CSV files or a pre-3.0 database. For details, see Import .
Authentication	<code>copy</code>	Copy data from an existing database to a new database. For details, see Copy a database .
	<code>set-default-admin</code>	Sets the default admin user when no roles are present.
Offline backup For details see Dump and load databases .	<code>set-initial-password</code>	Sets the initial password of the initial admin user (<code>neo4j</code>). For details, see Set an initial password .
	<code>dump</code>	Dump a database into a single-file archive.
Online backup For details see Backup .	<code>load</code>	Load a database from an archive created with the dump command.
	<code>backup</code>	Perform an online backup from a running Neo4j server.
Clustering	<code>restore</code>	Restore a backed-up database.
	<code>unbind</code>	Removes cluster state data from a stopped Neo4j server. For details, see Bind a Core Server .

Limitations

`neo4j-admin` must be invoked as the `neo4j` user in order to ensure the appropriate file permissions.

15.1.3. Environment variables

Neo4j Admin can utilize the following environment variables:

`NEO4J_DEBUG`

Set to anything to enable debug output.

`NEO4J_HOME`

Neo4j home directory.

`NEO4J_CONF`

Path to directory which 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.

15.1.4. Exit codes

If `neo4j-admin` finishes as expected it will return 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](#).

15.2. Consistency checker

Describes the Neo4j 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>]`
```

Options

Option	Default	Description
<code>--database</code>	<code>neo4j</code>	Name of database.
<code>--backup</code>		Path to backup to check consistency of. Cannot be used together with <code>--database</code> .
<code>--additional-config</code>		Configuration file to supply additional configuration in.
<code>--verbose</code>	<code>false</code>	Enable verbose output.
<code>--report-dir</code>	<code>.</code>	Directory to write report file in.
<code>--check-graph</code>	<code>true</code>	Perform checks between nodes, relationships, properties, types and tokens.
<code>--check-indexes</code>	<code>true</code>	Perform checks on indexes by comparing content with the store.
<code>--check-index-structure</code>	<code>true</code>	Perform physical structure check on indexes. No comparison with the store takes place.

Option	Default	Description
--check-label-scan-store	true	Perform checks on the label scan store.
--check-property-owners	false	Perform additional checks on property ownership. This check is very expensive in time and memory.

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 102. 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
```

15.3. Neo4j Admin report

This chapter describes the `report` command of Neo4j Admin.

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] [--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 will have to send it manually.

Options

Option	Default	Description
--to	reports/	Specifies to target directory where the report should be written to.
--list		Will list available classifiers.
--verbose		Will instruct the tool to print more verbose output.
--force		Will disable the available disk space check.
--pid		Specify process id of running Neo4j instance. Only applicable when used together with classifiers indicated as <i>Online</i> in the table below.

By default, the tool tries to estimate the final size of the report and use that to assert that there is enough disk space available for it. If there is not enough available space the tool will abort. However, this estimation is pessimistic and does not take the compression into consideration, so if you are confident that you do have enough disk space, you can disable this check with the option `--force`.

Classifiers

Classifier	Online	Description
all		Include all of the available classifiers listed below.
ccstate		Include the current cluster state.
config		Include the <code>neo4j.conf</code> file.
heap	X	Include a heap dump.
logs		Include log files, e.g <code>debug.log</code> , <code>neo4j.log</code> 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	X	Include a list of Java system properties.
threads	X	Include a thread dump of the running instance.

Classifier	Online	Description
tree		Include a text view of the folder structure of the data directory (no files are collected).
tx		Include transaction logs.

The classifiers indicated as *Online* only work when you have a running Neo4j instance that the tool can find.

If no classifiers are specified, the following classifiers will be used: `logs`, `config`, `plugins`, `tree`, `metrics`, `threads`, `sysprop` and `ps`.

The report tool does not read any data from your database. However, the heap, the raft logs, and the transaction logs may contain data. Additionally while the standard `neo4j.conf` file does not contain password information, for certain 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`.



This tool uses the [Java Attach API](#) to gather data from a running Neo4j instance. Therefore, it requires the Java JDK in order to execute properly.

15.4. Display store information

This chapter describes the `store-info` command of Neo4j Admin.

The `store-info` command outputs various pieces of useful information about sets of Neo4j store files. Concretely, the command provides information about the `record format` of the database store (`standard`, `aligned`, or `high_limit`), the store format `version`, when the version was introduced, and whether it needs to be migrated to a newer version. It also outputs the ID of the last transaction applied to the store, as well as whether the store will require recovery on next startup (for example if it was not shut down correctly).

15.4.1. Syntax and examples

The `neo4j-admin store-info` command is located in the `bin` directory. It can be invoked against a single database or backup as follows:

```
neo4j-admin store-info <path-to-store>
```

Example 103. Invoke `neo4j-admin store-info` against a database

```
$neo4j-home> /bin/neo4j-admin store-info data/databases/foo
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
```



When invoked against a single database, that database should be `offline`, otherwise the command will result in an error.

The command can also be invoked against a root directory containing several databases, as follows:

```
neo4j-admin store-info <root> --all
```

Example 104. Invoke `neo4j-admin store-info` against a root containing several databases

```
$neo4j-home> /bin/neo4j-admin store-info data/databases --all
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
```



When the command is invoked against several databases, if some are **online** they will simply report as **in use** and exclude all other information.

If you are parsing the results of this command you may use the `--structured` option to receive the output as a json object (or an array of json objects if combined with `-\all`). All the same fields are included and all values are strings.

Example 105. Invoke `neo4j-admin store-info` against a database and get json results

```
$neo4j-home> /bin/neo4j-admin store-info data/databases/foo --structured
{"databaseName":"foo"
 "inUse":false
 "storeFormat":"SF4.0.0"
 "storeFormatIntroduced":"4.0.0"
 "lastCommittedTransaction":"2"
 "recoveryRequired":true}
```

You can run the `store-info` command to see if the store format of the backups that you want to restore, are 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:

Example 106. Invoke `neo4j-admin store-info` against database backups

```
$neo4j-home> bin/neo4j-admin store-info /tmp/3518/mygraph.db
Database name:          mygraph.db
Database in use:        false
Store format version:   vE.H.4 \
Store format introduced in: 3.4.0 \
Store format superseded in: 4.0.0 \
Last committed transaction id:7263
Store needs recovery:   +false
```

The store format version reveals that the database is configured to use the [high-limit 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.



For more information on how to migrate a single database, see [Migration Guide □ Tutorial: Migrate a single database](#).

15.4.2. Store format versions and limits

Neo4j supports standard, aligned, and high limit store versions.

The following is the mapping of store format versions to Neo4j Server versions, and the maximum numbers of nodes, relationships, and properties that each of the formats supports.

Standard store format

Table 65. Standard store versions

Store Format Name	Store Format Version	Neo4j Server Version
STANDARD_V3_4	v0.A.9	3.4.0
STANDARD_V4_0	SF4.0.0	4.0.0

Table 66. Standard store limits

Store Format Name	Limit
Property keys	$2^{24} = 16M$
Nodes	$2^{35} = 34B$
Relationships	$2^{35} = 34B$
Properties	$2^{36} = 68B$
Labels	$2^{32} = 4B$
Relationship types	$2^{16} = 65k$
Relationship groups	$2^{35} = 34B$

Aligned store format

Table 67. Aligned store versions

Store Format Name	Store Format Version	Neo4j Server Version
ALIGNED_V4_1	AF4.1.a	4.1.0



The Aligned store limits are the same as the [Standard store limits](#).

High limit store format

Table 68. High limit store versions

Store Format Name	Store Format Version	Neo4j Server Version
HIGH_LIMIT_V3_0_0	vE.H.0	3.0.0
HIGH_LIMIT_V3_0_6	vE.H.0b	3.0.6
HIGH_LIMIT_V3_1_0	vE.H.2	3.1.0
HIGH_LIMIT_V3_2_0	vE.H.3	3.2.0
HIGH_LIMIT_V3_4_0	vE.H.4	3.4.0
HIGH_LIMIT_V4_0_0	HL4.0.0	4.0.0

Table 69. High limit store limits

Store Format Name	Limit
Property keys	$2^{24} = 16M$

Store Format Name	Limit
Nodes	$2^{50} = 1Q$ (Quadrillion)
Relationships	$2^{50} = 1Q$
Properties	$2^{50} = 10^{15} = 1Q$
Labels	$2^{32} = 4B$
Relationship types	$2^{24} = 16M$
Relationship groups	$2^{50} = 1Q$

15.5. Memory recommendations

This chapter describes the `memrec` command of Neo4j Admin.

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
<code>--memory=<size></code>	The memory capacity of the machine.	The amount of memory to allocate to Neo4j. Valid units are: <code>k</code> , <code>K</code> , <code>m</code> , <code>M</code> , <code>g</code> , <code>G</code> .
<code>--verbose</code>		Enable verbose output.
<code>--docker</code>		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 107. 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](#).

15.6. Import

This section describes how to perform batch imports of data into Neo4j.

You can do batch imports of large amounts of data into a Neo4j database from CSV files, using the `import` command of `neo4j-admin`. This tool can only be used to load data into a previously unused database. By default, this database is set to `neo4j` but can be configured to other names and locations. If you wish to import small to medium-sized CSV files into an existing database, use `LOAD CSV` (see [Cypher Manual](#) □ `LOAD CSV`).

These are some things you will 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 will trim extra whitespace at the beginning and end of strings. Quote your data to preserve leading and trailing whitespaces.



Indexes and constraints

Indexes and constraints are not created during the import. Instead, you will need to add these afterwards (see [Cypher Manual](#) □ [Indexes](#)).

This chapter explains how to format the input data and use the import tool. If you wish to see in-depth examples of using the import tool, refer to the [Import tutorial](#).

The following sections describe how to use the import tool:

- [Syntax](#) - The syntax of the `neo4j-admin import` command.

- [CSV header format](#) - How to construct the header row of each CSV file.
- [Options](#) - The options available for use with `neo4j-admin import`.

15.6.1. Syntax

The syntax for importing a set of CSV files is:

```
neo4j-admin import [--verbose]
  [--high-io[=<true/false>]]
  [--cache-on-heap]
  [--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=<idType>]
  [--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 108. Import data from CSV files

Assume that we have formatted our data as per [CSV file header format](#) so that we have it in six different files: `movies_header.csv`, `movies.csv`, `actors_header.csv`, `actors.csv`, `roles_header.csv`, and `roles.csv`. The following command will import 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
```



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.

15.6.2. CSV file header format

Explains the header format of CSV files when using the Neo4j import tool.

Header files

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.

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 109. 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 110. 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.

We are going to import the name and location coordinates for cities. First, we define the header as:

```
:ID,name,location:point{crs:WGS-84}
```

We 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 □ Temporal instants syntax](#) and [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 therefore possible to specify a default time zone for `Time` and `DateTime` values in the header, for example: `time{timezone:+02:00}` and: `datetime{timezone:Europe/Stockholm}`. If no default time zone is specified, the default timezone is determined by the `db.temporal.timezone` configuration setting. The default time zone can be explicitly overridden in the values in the data file.

Example 111. 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, we 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
```

We 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]
```

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.

`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 112. Define nodes files

We define the headers for movies in the `movies_header.csv` file. Movies have the properties `movieId`, `year` and `title`. We also specify a field for labels.

```
movieId:ID,title,year:int,:LABEL
```

We 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, we 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 takes a name, e.g. if `<name>:START_ID` or `<name>:END_ID` is defined, the `<name>` part will be ignored.

Example 113. Define relationships files

In this example we assume that the two nodes files from the previous example are used together with the following relationships file.

We 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 we 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
```

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 our 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 we 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, we use the syntax `START_ID(<ID space identifier>)` and `END_ID(<ID space identifier>)`.

Example 114. 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 we wish to ignore completely, this can be done using the **IGNORE** keyword in the header file. **IGNORE** must be prepended with a **:**.

Example 115. Skip a column

In this example, we 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 116. 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
```

15.6.3. Options

This section describes in details the options available when using the Neo4j import tool to import data from CSV files.



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.

`--verbose`

Enable verbose output.

`--database=<name>`

Name of database.

Default: `neo4j`

`--additional-config=<config-file-path>`

Configuration file with which to supply additional configuration.

`--report-file=<filename>`

File in which to store the report of the csv-import.

Default: `import.report`

`--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`: (Advanced) actual node IDs.

Default: `STRING`

`--input-encoding=<character-set>`

Character set that input data is encoded in.

Default: `UTF-8`

`--ignore-extra-columns[=<true/false>]`

If unspecified columns should be ignored during the import.

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`

`--ignore-empty-strings[=<true/false>]`

Determines whether or not empty string fields, such as "", from input source are ignored (treated as null).

Default: `false`

`--trim-strings[=<true/false>]`

Determines whether or not strings should be trimmed for whitespaces.

Default: `false`

`--legacy-style-quoting[=<true/false>]`

Determines whether or not backslash-escaped quote e.g. \" is interpreted as inner quote.

Default: `false`

`--delimiter=<char>`

Delimiter character between values in CSV data.

Unicode character encoding can be used if prepended by \. For example, \44 is equivalent to ,.

Default: ,

`--array-delimiter=<char>`

Delimiter character between array elements within a value in CSV data.

Unicode character encoding can be used if prepended by \. For example, \59 is equivalent to ;.

Default: ;

`--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 \.

`--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`

`--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%`

`--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.

`--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`

`--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.

`--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`

`--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 wasn't specified by the node input data.

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`

`--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`

`--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`

`--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 [Using regular expressions for specifying multiple input files](#).

`--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 [Using regular expressions for specifying multiple input files](#).

`@<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.

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.

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.

Output

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.

15.7. Dump and load databases

This section describes the dump and load commands of Neo4j Admin.

A Neo4j database can be dumped and loaded using the following commands:

```
neo4j-admin dump --database=<database> --to=<destination-path>
```

```
neo4j-admin load --from=<archive-path> --database=<database> [--force]
```

Limitations

- The database must be stopped before running the `dump` and `load` commands.
- `neo4j-admin` must be invoked as the `neo4j` user to ensure the appropriate file permissions.
- If the database used to load into doesn't exist prior to loading, it must be created subsequently using `CREATE DATABASE`. See [Administrative commands](#) for more information.

Usage

Using the `dump` and `load` commands is the recommended and safe way of transferring databases between environments. These commands are capable of discerning which files need to be exported and imported and which should not.

By contrast, file system copy-and-paste of databases is not supported.

Examples

Example 117. Use the `dump` command of `neo4j-admin`

Dump the database called `neo4j` into a file called `/backups/neo4j/2016-10-02.dump`. The destination directory for the dump file — in this case `/backups/neo4j` — must exist before calling the command.

```
$neo4j-home> bin/neo4j-admin dump --database=neo4j --to=/backups/neo4j/2016-10-02.dump  
$neo4j-home> ls /backups/neo4j  
$neo4j-home> 2016-10-02.dump
```

Example 118. Use the `load` command of `neo4j-admin`

Load the backed-up database contained in the file `/backups/neo4j/2016-10-02.dump` into database `neo4j`. If you have a database running, it needs to be shutdown first. When we use the `--force` option, any existing database gets overwritten.

```
$neo4j-home> bin/neo4j-admin load --from=/backups/neo4j/2016-10-02.dump --database=neo4j --force
```

Note that if using the load command to seed a Causal Cluster, you may need to first drop pre-existing databases or perform `neo4j-admin unbind` on each of the cluster instances. The procedure is described in [Seed from backups](#).

When loading database dumps, if a previous version of the database already exists, you must drop it (using `DROP DATABASE`) first. Alternatively, you can stop the Neo4j instance and unbind it from the cluster using `neo4j-admin unbind`. 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

15.8. Unbind a Core Server

This section describes how to remove cluster state data for a Neo4j server.

The cluster state of a Causal Cluster member can be removed by using the following command:

```
neo4j-admin unbind
```

Limitations

The Neo4j server process should be shutdown before running the `unbind` command.

Examples of usage

- When transforming a Causal Cluster member to a standalone server:

The `unbind` command can be used to turn a Causal Cluster server into a standalone server. To start the database in single (standalone) mode after unbinding it from the cluster, first set `dbms.mode=SINGLE` in [neo4j.conf](#).

- When seeding a Causal Cluster with existing store files:

If you wish to seed a new Causal Cluster using the store files of a previous cluster, you must first run `neo4j-admin unbind` on each server. For more information about seeding Causal Clusters, see [Seed a cluster](#).



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.

- When recovering 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 backup](#).



Unlike versions of Neo4j prior to v4.0.0, you **must** run the `unbind` command on both Read Replicas and Core members.

15.9. Copy a database

This chapter describes the `copy` command of Neo4j Admin.

The `copy` command of `neo4j-admin` is used to copy data from an existing database to a new database.

The syntax follows:

```
neo4j-admin copy --to-database=<database>
  [--neo4j-home-directory=<path>]
  [--from-database=<database> | --from-path=<path>)
  [--from-pagecache=<size>]
  [--from-path-tx=<path>]
  [--to-format=<format>]
  [--to-pagecache=<size>]
  [--delete-nodes-with-labels=<label>[,<label>...]]
  [--keep-only-nodes-with-labels=<label>[,<label>...]]
  [--skip-labels=<label>[,<label>...]]
  [--skip-properties=<property>[,<property>...]]
  [--skip-node-properties=<label.property>[,<label.property>...]]
  [--keep-only-node-properties=<label.property>[,<label.property>...]]
  [--skip-relationship-properties=<relationship.property>[,<relationship.property>...]]
  [--keep-only-relationship-properties=<relationship.property>[,<relationship.property>...]]
  [--skip-relationships=<relationship>[,<relationship>...]]'
  [--force]
  [--verbose]
```

The existing database must be stopped before copying from it, and the destination database must not yet exist.

The `copy` command can process an optional set of filters. This can be used to remove data that is unwanted in the destination database.

The schema definitions, i.e. index and constraint, are not automatically transferred. However, they will be extracted and presented as Cypher statements so you can recreate the ones you want.

Options

Option	Description
--from-database	The database name to copy from. Will assume the database is in the configured location.
--from-pagecache	The size of the page cache to use for reading.
--from-path	The path to the database to copy from. It can be used to target databases outside of the installation, e.g. backups.
--from-path-tx	The path to the transaction log files. You only need to use this if the command is unable to determine where they are located.
--to-database	The destination database name.
--neo4j-home-directory	Path to the home directory for the copied database. Default is the same as the database copied from.
--to-format	<p>The store format of the destination database. Valid values are <code>same</code>, <code>standard</code>, <code>high_limit</code>, <code>aligned</code>.</p> <p>The default value for this option is the format of the source database.</p>
--to-pagecache	The size of the page cache to use for writing.
--delete-nodes-with-labels	<p>A list of labels. Any node matching any of the labels will be ignored during copy.</p> <p>Cannot be combined with <code>--keep-only-nodes-with-labels</code>.</p>
--keep-only-nodes-with-labels	<p>A list of labels. All nodes that have any of the specified labels will be kept.</p> <p>Cannot be combined with <code>--delete-nodes-with-labels</code>.</p>
--skip-properties	<p>A list of property keys to ignore during the copy.</p> <p>Cannot be combined with <code>--skip-node-properties</code>, <code>--keep-only-node-properties</code>, <code>--skip-relationship-properties</code>, or <code>--keep-only-relationship-properties</code>.</p>

Option	Description
--skip-node-properties	A list of property keys to ignore for nodes with the specified label. Cannot be combined with <code>--skip-properties</code> or <code>--keep-only-node-properties</code> .
--keep-only-node-properties	A list of property keys to keep for nodes with the specified label. Any labels not explicitly mentioned will keep their properties. Cannot be combined with <code>--skip-properties</code> or <code>--skip-node-properties</code> .
--skip-relationship-properties	A list of property keys to ignore for relationships with the specified type. Cannot be combined with <code>--skip-properties</code> or <code>--keep-only-relationship-properties</code> .
--keep-only-relationship-properties	A list of property keys to keep for relationships with the specified type. Any relationship types not explicitly mentioned will keep their properties. Cannot be combined with <code>--skip-properties</code> or <code>--skip-relationship-properties</code> .
--skip-labels	A list of labels to ignore during the copy.
--skip-relationships	A list of relationship types to ignore during the copy.
--verbose	Will instruct the tool to print more verbose output.
--force	Will force the command to proceed even if the integrity of the database cannot be verified.



Due to the way filters are processed, the `id` of the node might change. This is true even if no filters are specified.

Examples

Example 119. Use the `copy` command to take a copy of the database `neo4j`.

To begin, you must stop the database named `neo4j`; this can be done by issuing the following Cypher statement:

```
STOP DATABASE neo4j
```

You can now copy the data from `neo4j`, to a new database called `copy`:

```
$neo4j-home> bin/neo4j-admin copy --from-database=neo4j --to-database=copy
```

A new database with the name `copy` now exists on the server, but it is not automatically picked up by Neo4j. To start the new database you have to insert it into Neo4j with the following Cypher query:

```
CREATE DATABASE copy
```

Neo4j will then detect the copied database and begin to use that.

Remember to start the database you copied from, if you still want it, using:

```
START DATABASE neo4j
```

The console output is saved to `logs/neo4j-admin-copy-<date>.log`.

Example 120. Use the `copy` command with filters.

The command can perform some basic forms of processing. You can remove nodes, labels, properties, and/or relationships.

The difference with `--skip-labels` and `--delete-nodes-with-labels` is that `--skip-labels` will just remove the labels, potentially leaving nodes without any labels.

```
$neo4j-home> bin/neo4j-admin copy --from-database=neo4j --to-database=copy --delete-nodes-with-labels="Cat,Dog"
```

After this command, you will have a copy of the database `neo4j`, without nodes with the labels `:Cat` and `:Dogs`.



Labels are processed independently, i.e. the filter described above will delete any node with either `:Cat` or `:Dogs`, and not only nodes that have both of the labels.

15.9.1. Sharding data with the `copy` command

The `copy` command can be used to filter out data for a [Fabric installation](#).

In the following example we will go through a sample database that will be separated into 3 shards.

Example 121. 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* will contain the team data. *Shard 2* and *Shard 3* will contain person data.

1. We start by creating *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 doesn't have the label `:S1` or `:SAll`.

The `--skip-labels` property is also used so as to not include the temporary labels we created for the sharding process. The resulting shard will contain 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. Next we 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* we want to keep the `:Team` nodes as proxy nodes in order to be able to link together information from the separate shards. The nodes will be included since they have the label `:SAll`, but we 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.

- Finally, we do exactly the same thing for *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
```

This will produce:

```
(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, we 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`, since only one property should be kept.

The relationship property filters works in the same way.

15.10. Cypher Shell

Describes Neo4j Cypher Shell command-line interface (CLI) and how to use it.

15.10.1. About Cypher Shell CLI

Cypher Shell is a command-line tool that comes with the Neo4j installation. It can also be downloaded from [Neo4j Download 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.

15.10.2. Syntax

The Cypher Shell CLI is located in the `bin` directory if installed as part of the product.

The syntax is:

```

cypher-shell [-u USERNAME, --username USERNAME]
  [cypher]
  [-h, --help]
  [--fail-fast]
  [--fail-at-end]
  [--format]
  [--debug]
  [--non-interactive]
  [-v, --version]
  [-a ADDRESS, --address ADDRESS]
  [-p PASSWORD, --password PASSWORD]
  [--encryption]
  [-d DATABASE, --database DATABASE]
  [--P PARAM, --param PARAM]
  [--sample-rows SAMPLE-ROWS]
  [--wrap]
  [--driver-version]
  [-f FILE, --file FILE]

```

Arguments

Argument	Type	Description	Default value
<code>-u USERNAME, --username USERNAME</code>	Connection argument	Username to connect as. It can also be specified by the environment variable <code>NEO4J_USERNAME</code> .	
<code>cypher</code>	Positional argument	An optional string of cypher to execute and then exit.	
<code>-h, --help</code>	Optional argument	Show help message and exit.	
<code>--fail-fast</code>	Optional argument	Exit and report failure on first error when reading from file.	This is the default behavior.
<code>--fail-at-end</code>	Optional argument	Exit and report failures at end of input when reading from file.	
<code>--format {auto,verbose,plain}</code>	Optional argument	Desired output format.	<p><code>auto</code> (default) displays results in tabular format if you use the shell interactively and with minimal formatting if you use it for scripting.</p> <p><code>verbose</code> displays results in tabular format and prints statistics.</p> <p><code>plain</code> displays data with minimal formatting.</p>
<code>--debug</code>	Optional argument	Print additional debug information.	<code>false</code>
<code>--non-interactive</code>	Optional argument	Force non-interactive mode; only useful if auto-detection fails (e.g. Windows).	<code>false</code>
<code>-v, --version</code>	Optional argument	Print version of cypher-shell and exit.	<code>false</code>
<code>-a ADDRESS, --address ADDRESS</code>	Connection argument	Address and port to connect to.	<code>neo4j://localhost:7687</code>
<code>-p PASSWORD, --password PASSWORD</code>	Connection argument	Password to connect with. It can also be specified by the environment variable <code>NEO4J_PASSWORD</code> .	

Argument	Type	Description	Default value
--encryption {true, false, default}	Connection argument	Whether the connection to Neo4j should be encrypted; must be consistent with Neo4j's configuration.	default - the encryption setting is deduced from the specified address. For example, the neo4j+ssc protocol would use encryption.
-d DATABASE, --database DATABASE	Connection argument	Database to connect to. It can also be specified by the environment variable NEO4J_DATABASE.	
--P PARAM, --param PARAM	Optional argument	Add a parameter to this session. For example, -P "number 3". This argument can be specified multiple times.	
--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
--driver-version	Optional argument	Print version of the Neo4j Driver used and exit.	false
-f FILE, --file FILE	Optional argument	Pass a file with cypher statements to be executed. After the statements have been executed cypher-shell shuts down.	

Example 122. Invoke cypher-shell with username and password

```
$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.  
neo4j>
```

Example 123. Invoke help

```
neo4j> :help
```

```
Available commands:  
:begin Open a transaction  
:commit Commit the currently open transaction  
: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 Prints 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
```

Example 124. Execute a query

```
neo4j> MATCH (n) RETURN n;
```

```
+-----+  
| n |  
+-----+  
| (:Person {name: "Bruce Wayne", alias: "Batman"}) |  
| (:Person {name: "Selina Kyle", alias: ["Catwoman", "The Cat"]}) |  
+-----+
```

Example 125. Invoke `cypher-shell` with a Cypher script

The contents of a file called `examples.cypher`:

```
MATCH (n) RETURN n;  
  
MATCH (batman:Person {name: 'Bruce Wayne'}) RETURN batman;
```

Invoke the `examples.cypher` script from the command-line. All the examples in the remainder of this section use the `--format plain` flag for a simple output.

Using `cat` (UNIX)

```
$neo4j-home> cat examples.cypher | bin/cypher-shell -u neo4j -p <password> --format plain
```

Using `type` (Windows)

```
$neo4j-home> type examples.cypher | bin/cypher-shell.bat -u neo4j -p <password> --format plain
```

Result

```
n  
(:Person {name: "Bruce Wayne", alias: "Batman"})  
(:Person {name: "Selina Kyle", alias: ["Catwoman", "The Cat"]})  
batman  
(:Person {name: "Bruce Wayne", alias: "Batman"})
```

15.10.3. Query parameters

Cypher Shell CLI supports querying based on parameters. This is often used while scripting.

Example 126. Use parameters within Cypher Shell

Set the parameter `thisAlias` to `Robin` using the `:param` keyword. Check the parameter using the `:params` keyword.

```
neo4j> :param thisAlias => 'Robin'  
neo4j> :params  
:param thisAlias => 'Robin'
```

Now use the parameter `thisAlias` in a Cypher query. Verify the result.

```
neo4j> CREATE (:Person {name : 'Dick Grayson', alias : $thisAlias});  
Added 1 nodes, Set 2 properties, Added 1 labels  
neo4j> 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
```

15.10.4. Transactions

Cypher Shell supports explicit transactions. Transaction states are controlled using the keywords `:begin`, `:commit`, and `:rollback`.

Example 127. Use fine-grained transaction control

Start a transaction in your first Cypher Shell session:

```
neo4j> 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
neo4j> :begin
neo4j# CREATE (:Person {name : 'Edward Mygma', alias : 'The Riddler' });
```

If you open a second Cypher Shell session, you will notice no changes from the latest `CREATE` statement.

```
neo4j> 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
```

Go back to the first session and commit the transaction.

```
neo4j# :commit
0 rows available after 1 ms, consumed after another 0 ms
Added 1 nodes, Set 2 properties, Added 1 labels
neo4j> 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"}) |
| (:Person {name: "Edward Mygma", alias: "The Riddler"}) |
+-----+
4 rows available after 1 ms, consumed after another 1 ms
neo4j>
```

15.10.5. Procedures

Cypher Shell supports running any procedures for which the current user is authorized.

Example 128. Call the dbms.showCurrentUser procedure

```
neo4j> CALL dbms.showCurrentUser();
+-----+
| username | roles      | flags |
+-----+
| "neo4j"  | ["admin"]  | []    |
+-----+
1 row available after 66 ms, consumed after another 2 ms
neo4j> :exit
```

15.10.6. 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.

Appendix A: Reference

This appendix contains the following:

- [Configuration settings](#)
- [Procedures](#)

Appendix B: Tutorials

This appendix contains examples and tutorials that further describe usages of Neo4j.

The following step-by-step tutorials cover common operational tasks or otherwise exemplify working with Neo4j.

- [Set up a local Causal Cluster](#)
- [Use the Import tool to import data into Neo4j](#)
- [Set up and use Fabric](#)

B.1. Set up a local Causal Cluster

This tutorial walks through the basics of setting up a Neo4j Causal Cluster. The result is a local cluster of six instances: three Cores and three Read Replicas.

This tutorial describes the following:

- [Introduction](#)
- [Download Neo4j](#)
- [Set up the Cores servers](#)
 - [Configure and start the first Core](#)
 - [Configure and start the second Core](#)
 - [Configure and start the third Core](#)
- [Check the status of the cluster](#)
- [Set up the Read Replicas](#)
 - [Configure and start the first Read Replica](#)
 - [Configure and start the second Read Replica](#)
 - [Configure and start the third Read Replica](#)
- [Check the status of the cluster](#)

Introduction

In this tutorial, you will learn how to deploy a Causal Cluster locally on a single machine.



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 [Core 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-4.2.0` 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:



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_atFormation=3`.
 - c. Locate and uncomment the setting
`causal_clustering.minimum_core_cluster_size_atRuntime=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-4.2.0` 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-4.2.0` 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
```

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 129. 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-4.2.0` 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-4.2.0` 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-4.2.0` 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 130. 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. Use the Import tool

This tutorial provides detailed examples of using the Neo4j import tool.

This tutorial walks us through a series of examples to illustrate the capabilities of the Import tool.

When using CSV files for loading a database, each node must have a unique ID to be able to be referenced when creating relationships between nodes in the same import. In cases where the node ID is only unique within files, using *ID spaces* is a way to ensure uniqueness across all nodes files. Please see below and [Using ID spaces](#) for more information.

Relationships are created by connecting the node IDs. In the examples below, the node IDs are stored as properties on the nodes. Node IDs may be of interest later for cross-reference to other systems, traceability etc., but they are not mandatory. If you do not want the IDs to persist after a completed import, then do not specify a property name in the `:ID` field.

It is possible to import only nodes using the import tool by omitting a relationships file when calling `neo4j-admin import`. Any relationships between the imported nodes will have to be created later by another method, since the import tool works for initial graph population only.

For this tutorial we will use a data set containing movies, actors and roles. If not stated otherwise, the examples assume that the name of the database is `neo4j` (the default name) and that all CSV files are located in the `import` directory. Note that if you wish to run one example after another you have to

remove the database in between.

Header files



For the basic examples we will use the first row of the data file for the header. This is fine when experimenting, but when working with anything but the smallest datasets we recommend keeping the header in a separate file.

B.2.1. Basic example

First we will look at the movies. Each movie has an ID, which is used for referring to it from other data sources. Moreover, each movie also has a title and a year. Along with these properties we also add the node labels **Movie** and **Sequel**:

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
```

Next up are the actors. They have an ID - in this case a shorthand of their name - and a name. All the actors have the node label **Actor**:

actors.csv

```
personId:ID,name,:LABEL
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
```



The node ID is needed to create relationships, but the node labels are optional.

Finally we have the roles that an actor plays in a movie, which will be represented by relationships in the database. There are three mandatory fields for relationships; **START_ID**, **END_ID** and **:TYPE**.

In order to create a relationship between nodes we use the IDs defined in **actors.csv** and **movies.csv** for the **START_ID** and **END_ID** fields. We 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
```

The call to **neo4j-admin import** would look like this:

```
neo4j_home$ bin/neo4j-admin import --nodes=import/movies.csv --nodes=import/actors.csv
--relationships=import/roles.csv
```



In Community Edition, the database `neo4j` must be empty before the import, or else the import will fail. If `neo4j` is not empty, the entire directory may be deleted as it is created on import if non-existent. However, in Enterprise Edition, if there is an unfinished import in the `neo4j` folder, the tool will attempt to complete that import but fail if it cannot.

Now start up a database from the target directory:

```
neo4j_home$ ./bin/neo4j start
```

To see your imported data in the graph, try a simple command:

```
MATCH (n)
RETURN count(n) as count
```

B.2.2. Customizing configuration options

We can customize the configuration options that the import tool uses (see [Options](#)) if our data does not fit the default format. The following CSV files are delimited by `,`, use `|` as the array delimiter and use `'` for quotes:

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
```

The call to `neo4j-admin import` would look like this:

```
neo4j_home$ bin/neo4j-admin import --nodes=import/movies2.csv --nodes=import/actors2.csv
--relationships=import/roles2.csv --delimiter="," --array-delimiter="|" --quote=""
```

B.2.3. 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 nodes.csv.gz
. --relationships rels.zip`

We will use the same data as in the previous example but put 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
```

The call to `neo4j-admin import` would look as follows, note how the file groups are enclosed in quotation marks in the command:

```
neo4j_home$ bin/neo4j-admin import --nodes=import/movies3-header.csv,import/movies3.csv  
--nodes=import/actors3-header.csv,import/actors3.csv --relationships=import/roles3  
-header.csv,import/roles3.csv
```



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.

B.2.4. Multiple input files

In addition to using a separate header file you can also provide multiple nodes or relationships files. This may be useful for example for processing the output from a Hadoop pipeline. 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:

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
```

The call to `neo4j-admin import` would look like this:

```
neo4j_home$ bin/neo4j-admin import --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
```

See also [Using regular expressions for specifying multiple input files](#).

Using regular expressions for specifying multiple input files

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.

As mentioned in a previous section, for the import to work correctly, the header file must be 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.

For example, let's assume that we have the following files:

- *movies4-header.csv*
- *movies4-data1.csv*
- *movies4-data2.csv*
- *movies4-data12.csv*

If we 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.

Using the same data files as in the previous example, the call to `neo4j-admin import` can be simplified to:

```
neo4j_home$ bin/neo4j-admin import --nodes=import/movies4-header.csv,import/movies4-part.*  
--nodes=import/actors4-header.csv,import/actors4-part.* --relationships=import/roles4-  
-header.csv,import/roles4-part.*
```

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`.

B.2.5. Types and labels

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` field in the node file if you pass it as a command line option. If you do then both the label provided in the file and the one provided on the command line will be added to the node.

In this example we put the label `Movie` on every node specified in `movies5a.csv`, and we 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
```

The call to `neo4j-admin import` would look like this:

```
neo4j_home$ bin/neo4j-admin import --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`. If you provide a relationship type both on the command line and in the relationships file, the one in the file will be applied. In this example we put the relationship type `ACTED_IN` 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
```

The call to `neo4j-admin import` would look like this:

```
neo4j_home$ bin/neo4j-admin import --nodes=import/movies5b.csv --nodes=import/actors5b.csv
--relationships=ACTED_IN=import/roles5b.csv
```

B.2.6. Property types

The type for properties specified in nodes and relationships files is defined in the header row. (see [CSV file header format](#))

The following example creates a small graph containing one actor and one movie connected by an `ACTED_IN` 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
```

The call to `neo4j-admin import` would look like this:

```
neo4j_home$ bin/neo4j-admin import --nodes=import/movies6.csv --nodes=import/actors6.csv
--relationships=import/roles6.csv
```

B.2.7. ID handling

A node ID is used to find the correct nodes when creating relationships. Each node processed by `neo4j-admin import` must provide an ID if it is to be connected in any relationships.

Working with sequential or auto incrementing identifiers

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.

For example, if movies and people both use sequential identifiers then we 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
```

We also need to reference the appropriate ID space in our 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
```

The call to `neo4j-admin import` would look like this:

```
neo4j_home$ bin/neo4j-admin import --nodes=import/movies7.csv --nodes=import/actors7.csv
--relationships=ACTED_IN=import/roles7.csv
```

B.2.8. Bad input data

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: bad relationships and bad nodes. We will have a closer look at these in the following examples.

Relationships referring to missing 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.

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
```

The call to `neo4j-admin import` would look like this:

```
neo4j_home$ bin/neo4j-admin import --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 we append the `--skip-bad-relationships` flag:

```
neo4j_home$ bin/neo4j-admin import --nodes=import/movies8a.csv --nodes=import/actors8a.csv
--relationships=import/roles8a.csv --skip-bad-relationships
```

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
```

Multiple nodes with same ID within same ID space

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.

In the following example there is a node ID 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
```

The call to `neo4j-admin import` would look like this:

```
neo4j_home$ bin/neo4j-admin import --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 we append the `--skip-duplicate-nodes` flag:

```
neo4j_home$ bin/neo4j-admin import --nodes=import/actors8b.csv --skip-duplicate-nodes
```

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.3. Set up and use Fabric

This tutorial walks through the basics of setting up and using Neo4j 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](#)

For more information on how to manage multiple active databases in Neo4j, see [Manage databases](#).

For more details on Fabric, see [Fabric](#).

B.3.1. 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.

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

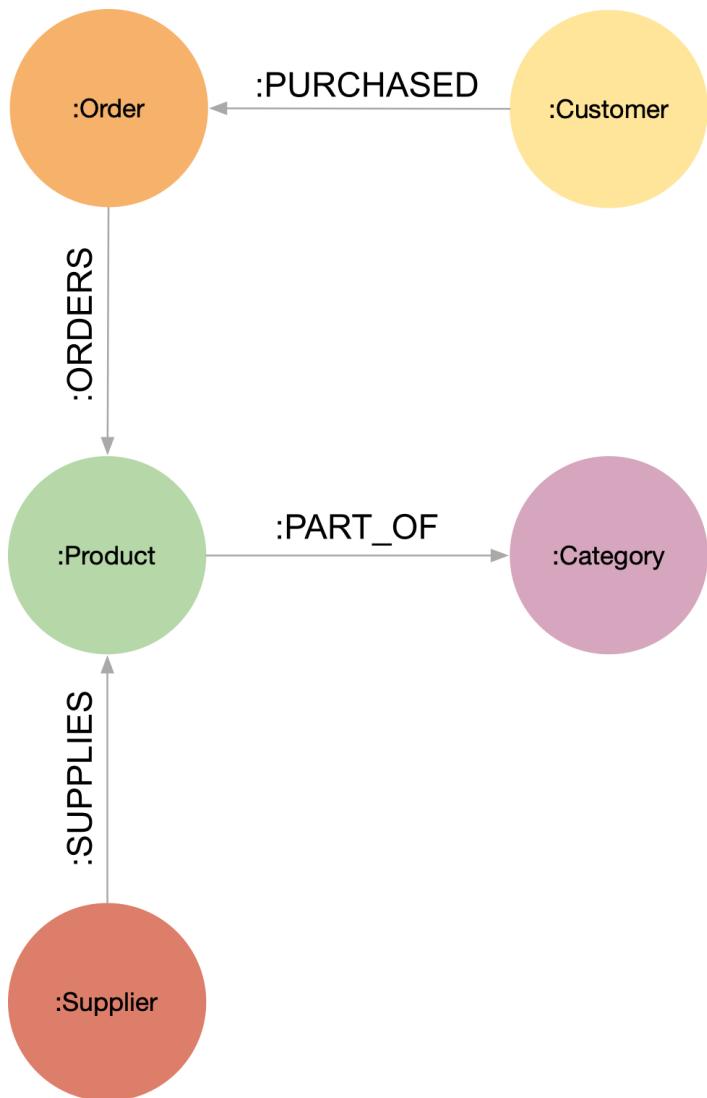


Figure 14. The Northwind data model

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**.

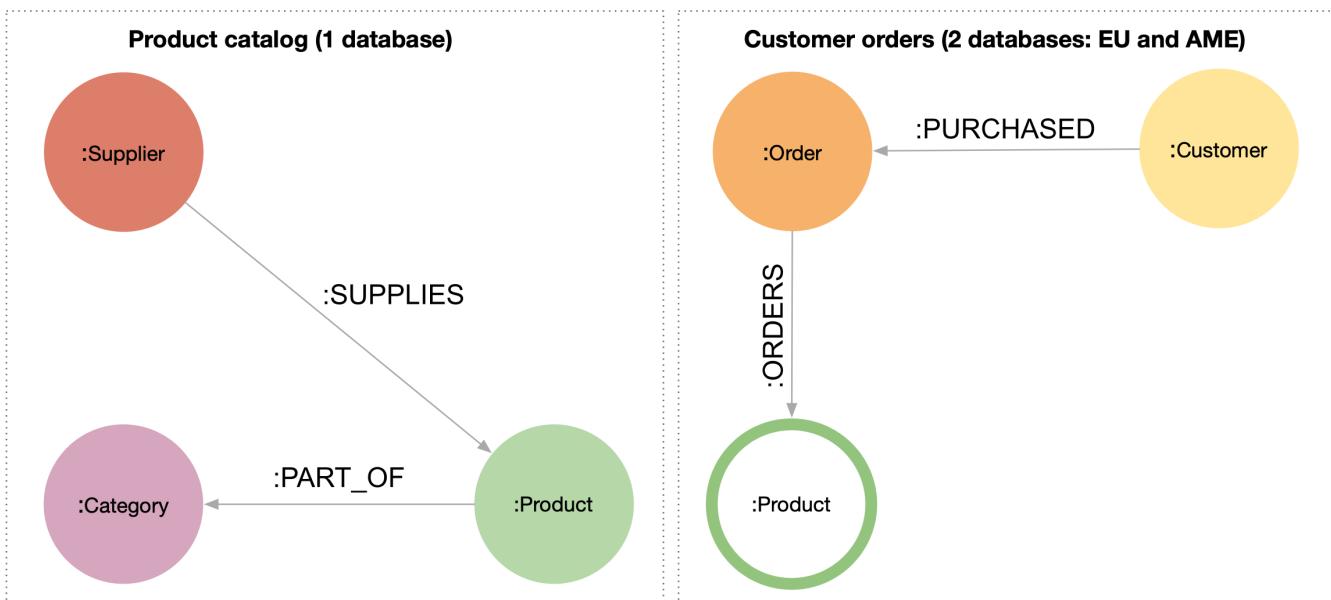


Figure 15. The new data model

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.

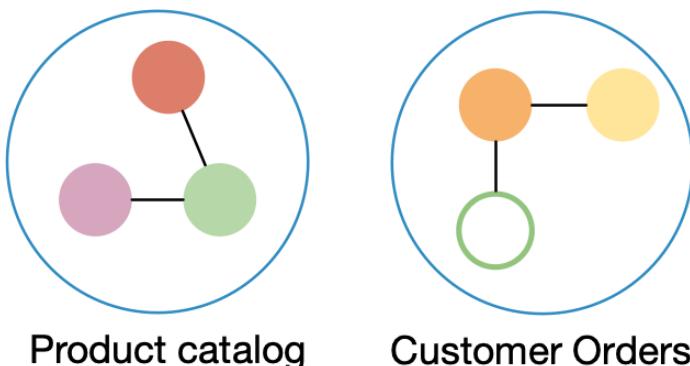


Figure 16. Data Federation

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*.



AME Customer Orders EU Customer Orders

Figure 17. 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.

B.3.2. Configure Fabric with three databases

Now that you have a new multi-database model defined, you can start to configure the Fabric infrastructure.

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.

```
~/neo4j-enterprise-4.1.x/bin$ ./neo4j start
```

2. Check all available databases.

```
~/neo4j-enterprise-4.1.x/bin$ ls -al ../data/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.

```
~/neo4j-enterprise-4.1.x/bin$ ./cypher-shell -u neo4j -p neo4j
```

```
Password change required
new password: *****
Connected to Neo4j 4.1.x 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.

```
neo4j@neo4j> show databases;
```

name	address	role	requestedStatus	currentStatus	error	default
"neo4j"	"localhost:7687"	"standalone"	"online"	"online"	"	TRUE
"system"	"localhost:7687"	"standalone"	"online"	"online"	"	FALSE

```
2 rows available after 102 ms, consumed after another 11 ms
```

5. Run the command `create database <database-name>` to create the databases.

```
neo4j@neo4j> create database db0;
```

```
0 rows available after 137 ms, consumed after another 0 ms
```

```
neo4j@neo4j> create database db1;
```

```
0 rows available after 14 ms, consumed after another 0 ms
```

```
neo4j@neo4j> 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.

```
neo4j@neo4j> show databases;
```

name	address	role	requestedStatus	currentStatus	error	default
"db0"	"localhost:7687"	"standalone"	"online"	"online"	"	FALSE
"db1"	"localhost:7687"	"standalone"	"online"	"online"	"	FALSE
"db2"	"localhost:7687"	"standalone"	"online"	"online"	"	FALSE
"neo4j"	"localhost:7687"	"standalone"	"online"	"online"	"	TRUE
"system"	"localhost:7687"	"standalone"	"online"	"online"	"	FALSE

```
5 rows available after 8 ms, consumed after another 7 ms
```

7. Exit the Cypher Shell command-line tool.

```
neo4j@system> :exit;
```

```
Bye!
```

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 `fabriccnw`.

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=fabriccnw  
  
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>/bin/` folder and restart the Neo4j DBMS.

```
~/neo4j-enterprise-4.1.x/bin$ ./neo4j restart
```

4. Connect to the Neo4j DBMS using `./cypher-shell` and your credentials.

```
~/neo4j-enterprise-4.1.x/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`.

```
neo4j@neo4j:~> show databases;
```

name	address	role	requestedStatus	currentStatus	error	default
"db0"	"localhost:7687"	"standalone"	"online"	"online"	"	FALSE
"db1"	"localhost:7687"	"standalone"	"online"	"online"	"	FALSE
"db2"	"localhost:7687"	"standalone"	"online"	"online"	"	FALSE
"fabriccnw"	"localhost:7687"	"standalone"	"online"	"online"	"	FALSE
"neo4j"	"localhost:7687"	"standalone"	"online"	"online"	"	TRUE
"system"	"localhost:7687"	"standalone"	"online"	"online"	"	FALSE

```
6 rows available after 242 ms, consumed after another 18 ms
```

B.3.3. 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 ON :Product(productID);
CREATE INDEX ON :Category(categoryID);
CREATE INDEX ON :Supplier(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 ON :Customer(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 ON :Order(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 ON :Product(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 ON :Customer(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 ON :Order(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 ON :Product(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

B.3.4. 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.

```
neo4j@db2>> :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`.

```
neo4j@db2>> :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.

B.3.5. 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](#).

Appendix C: Advanced Causal Clustering

This appendix describes advanced features of Neo4j 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 describes the lifecycle of a Neo4j Causal Cluster.

This section includes:

- [Introduction](#)
- [Discovery protocol](#)
- [Core membership](#)
- [Read Replica membership](#)
- [Transacting via the Raft protocol](#)
- [Catchup protocol](#)
- [Read Replica shutdown](#)
- [Core shutdown](#)

C.1.1. 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.

C.1.2. 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.

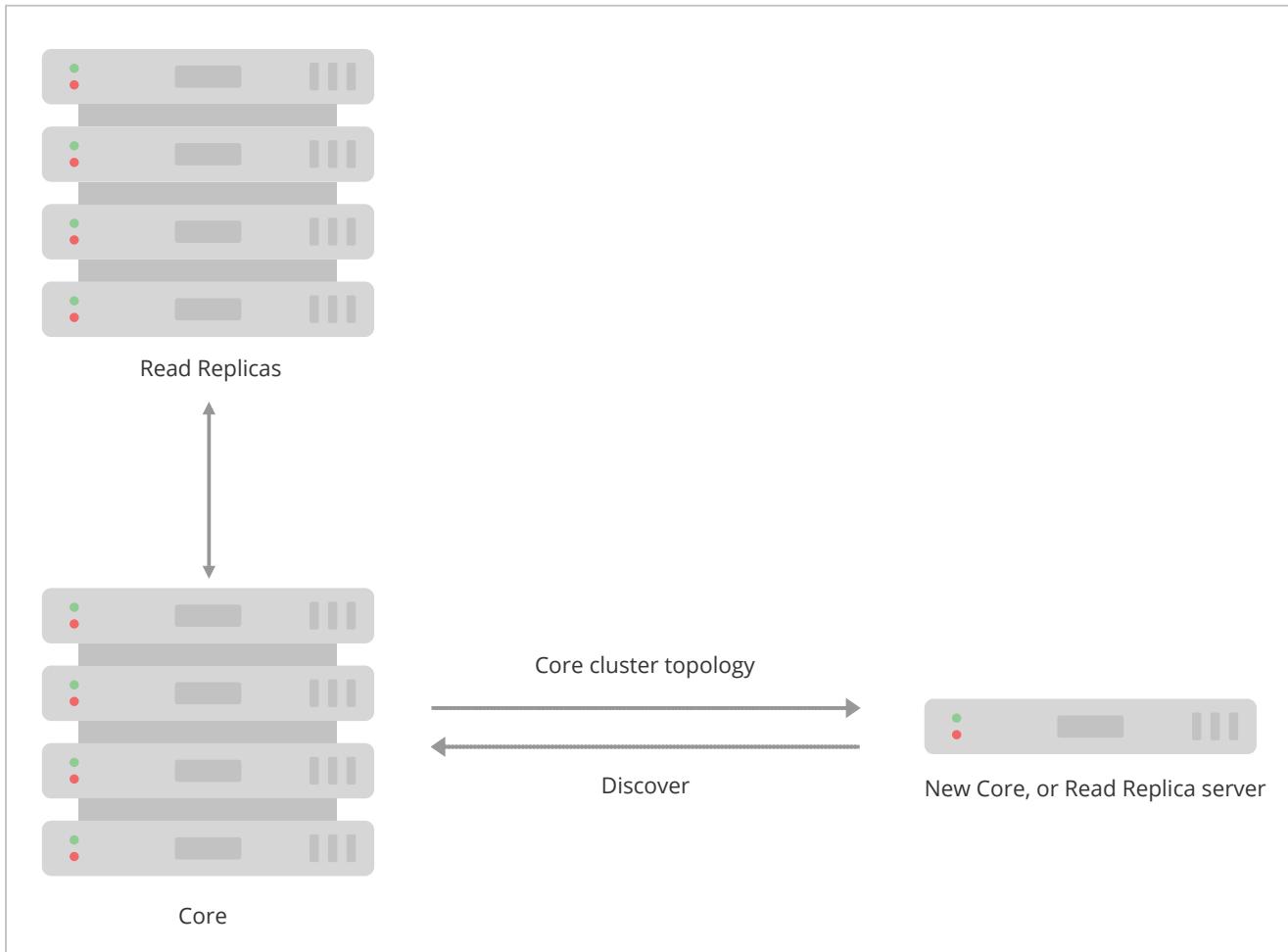


Figure 18. Causal Cluster discovery protocol: Core-to-Core or Read Replica-to-Core only.

Using this information, the server will either join an existing cluster or form one of its own.



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](#).

C.1.3. 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.



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.

C.1.4. 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.

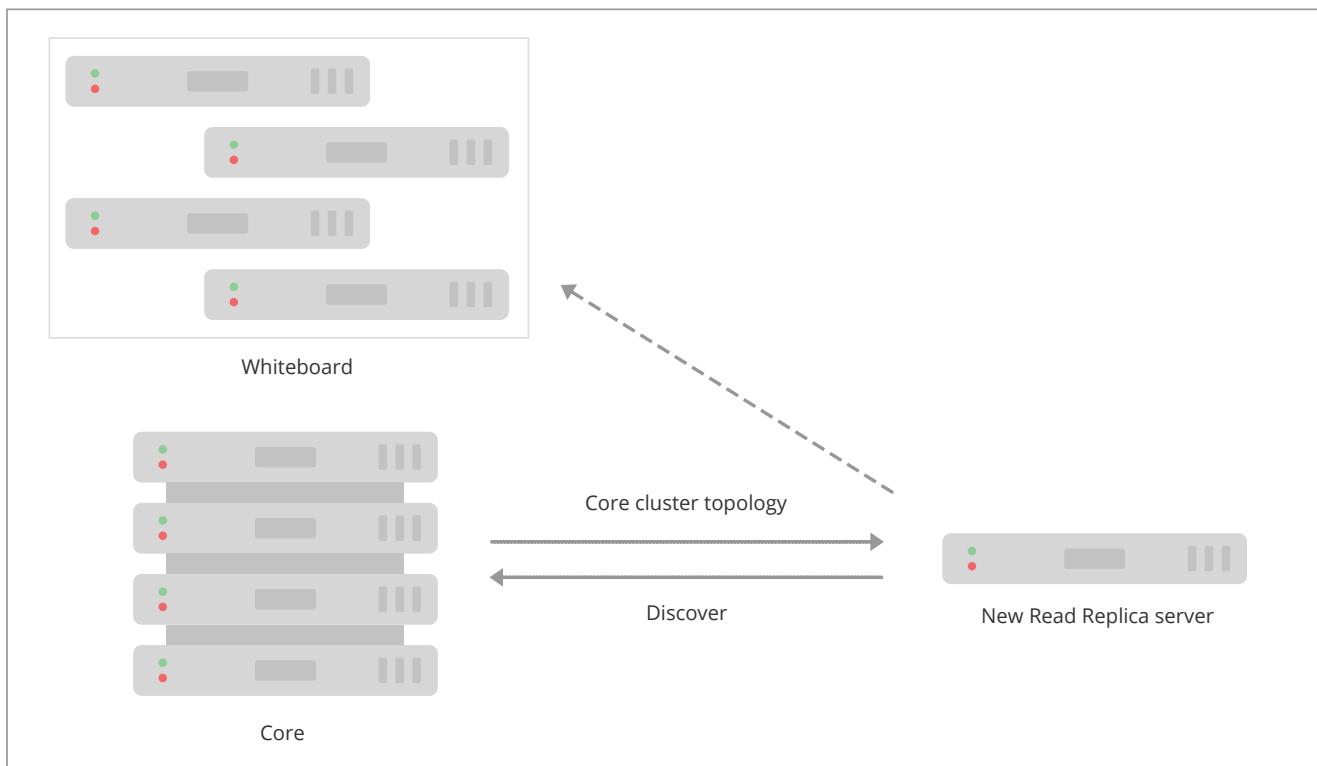


Figure 19. All Read Replicas registered with shared whiteboard.

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.

C.1.5. 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.



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.

i 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.

i 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.

C.1.6. 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.

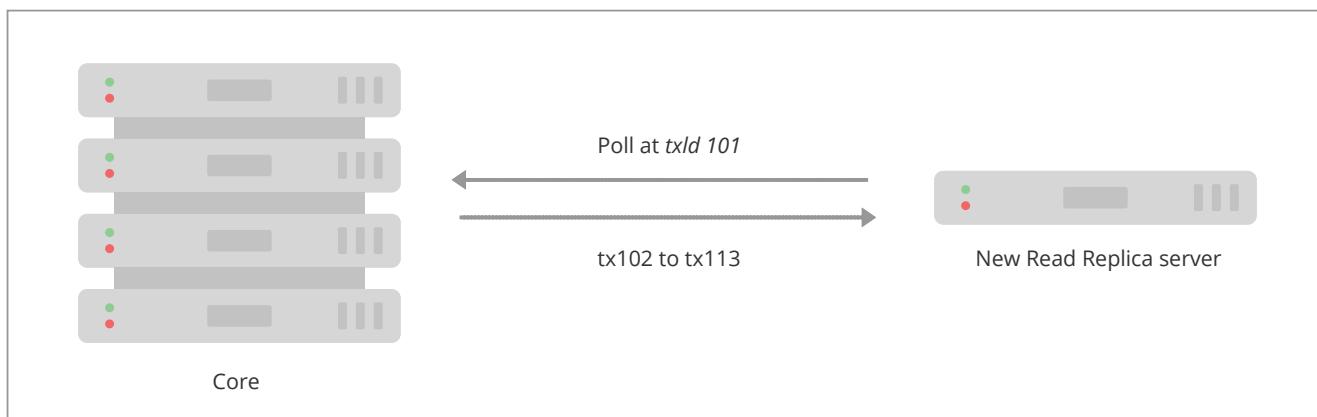


Figure 20. Transactions shipped from Core to Read Replica.

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.

i 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.



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.

C.1.7. 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.

C.1.8. 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).



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

This section introduces the multi-data center functionality in Neo4j.

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](#)

C.2.1. 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.2.2. Multi-data center design

This section describes common patterns for multi-data center deployments that can act as building blocks for your own multi-data center production environments.

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 131. Homogeneous three data center deployment

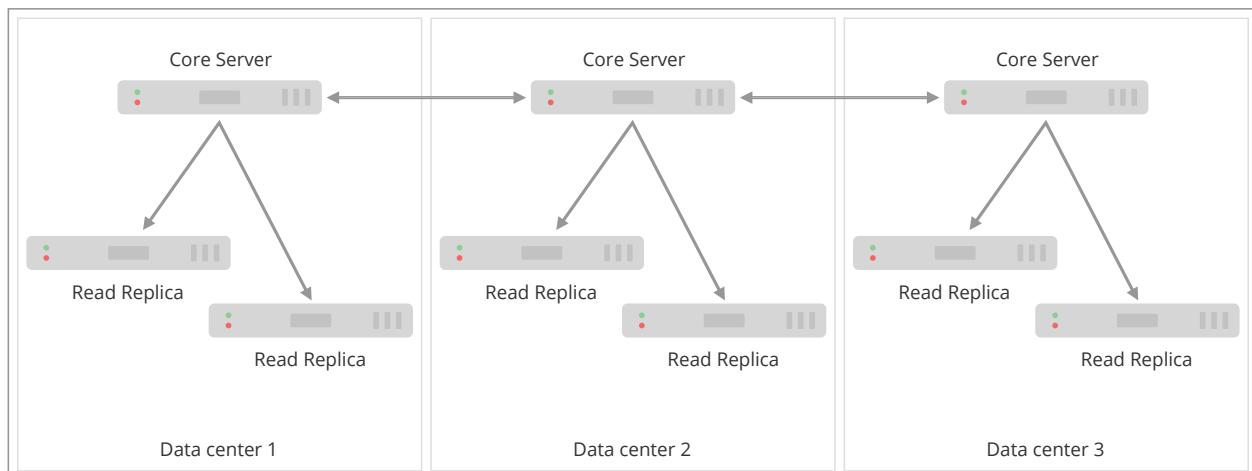


Figure 21. Homogeneous deployment across three data centers with one Core instance in each

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.

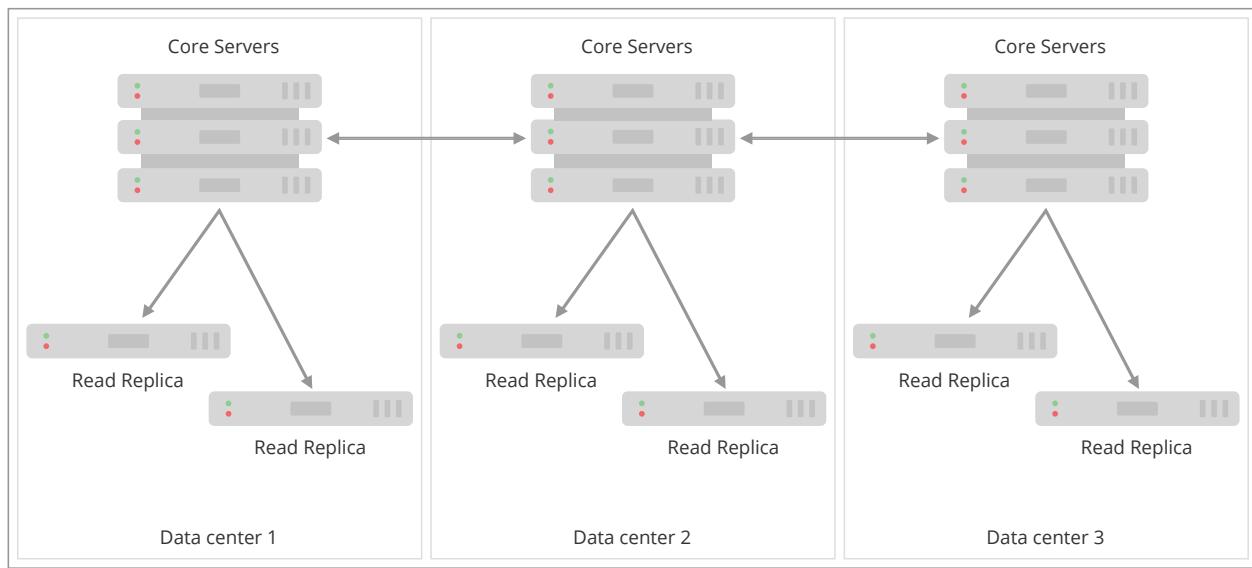


Figure 22. Homogeneous deployment across three data centers with three Core instances in each

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 132. Homogeneous two data center deployment

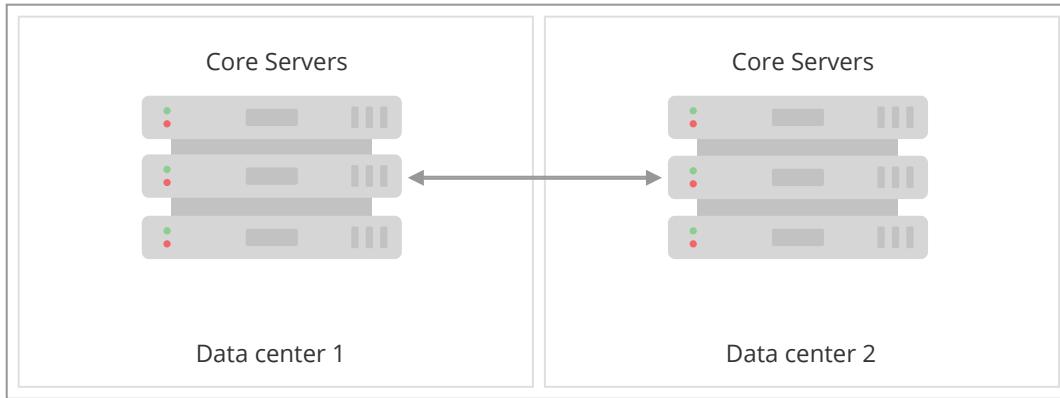


Figure 23. Homogeneous deployment across two data centers

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 133. 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.

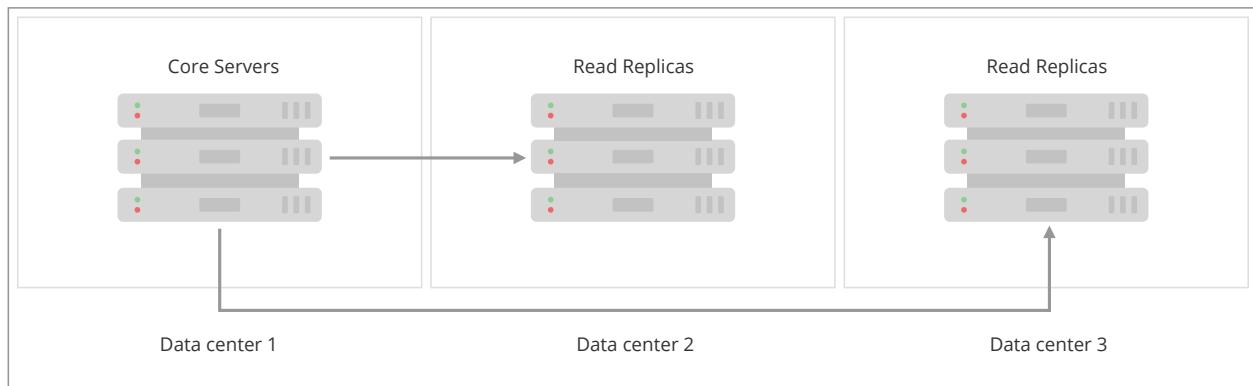


Figure 24. Heterogeneous deployment separating Read Replicas from the Core cluster

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.

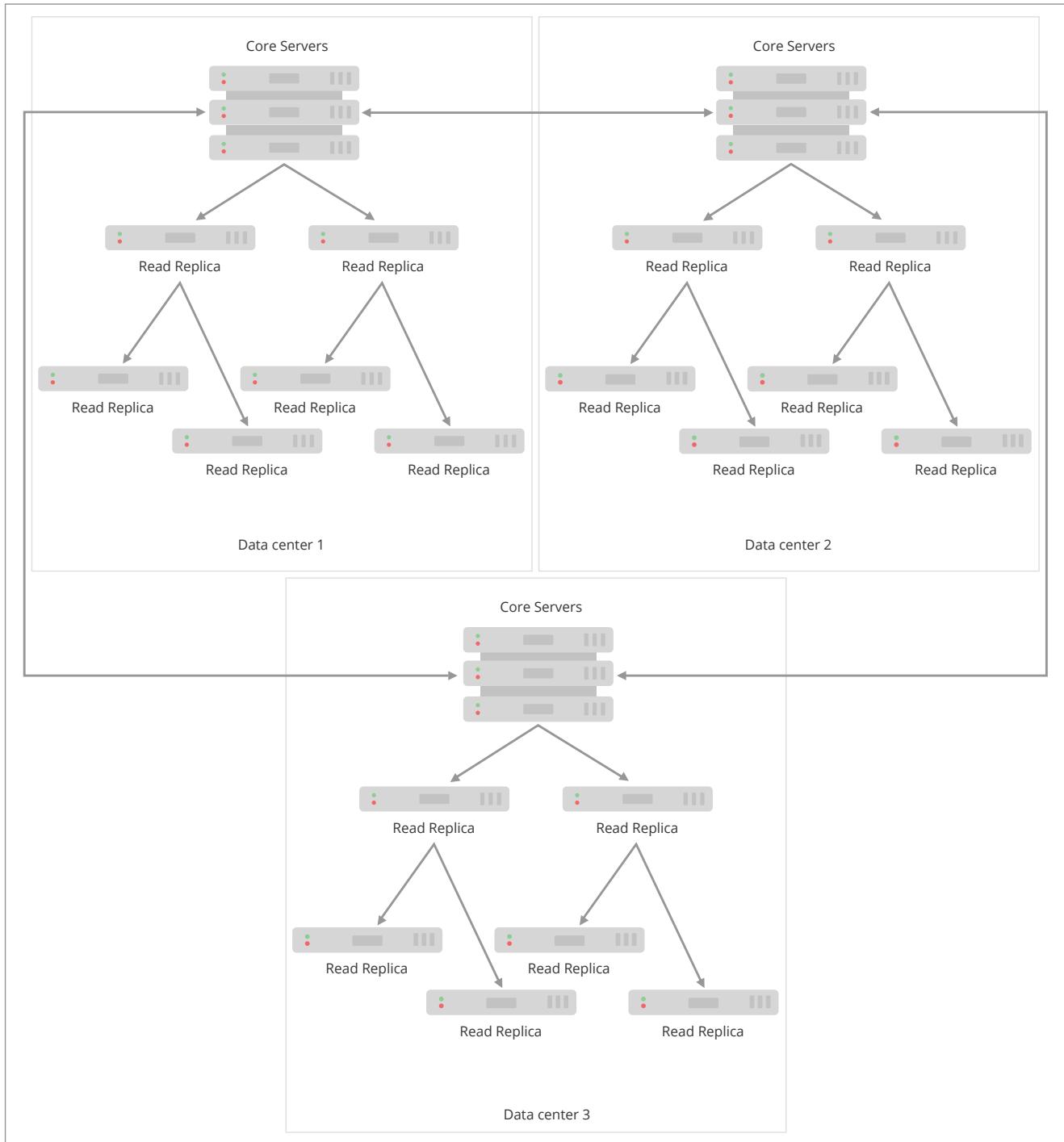


Figure 25. Fan out from Core Servers for scale at log cost

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.

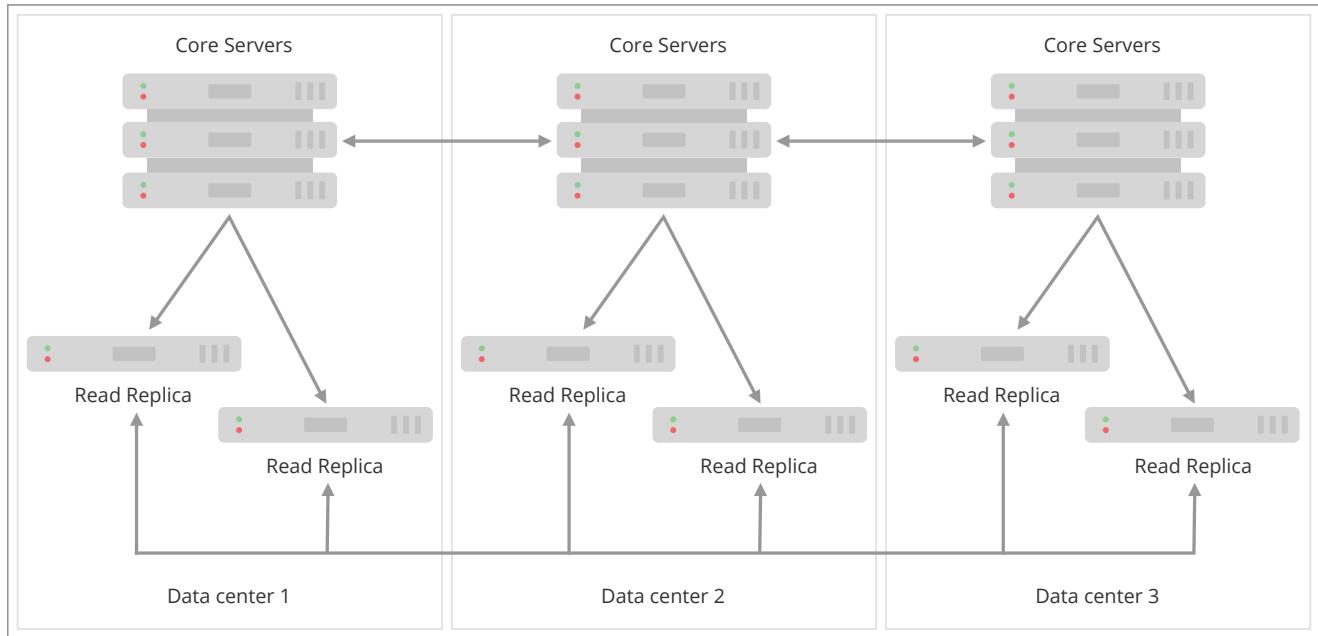


Figure 26. Peer-to-peer Read Replica catchup

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.



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](#) 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.

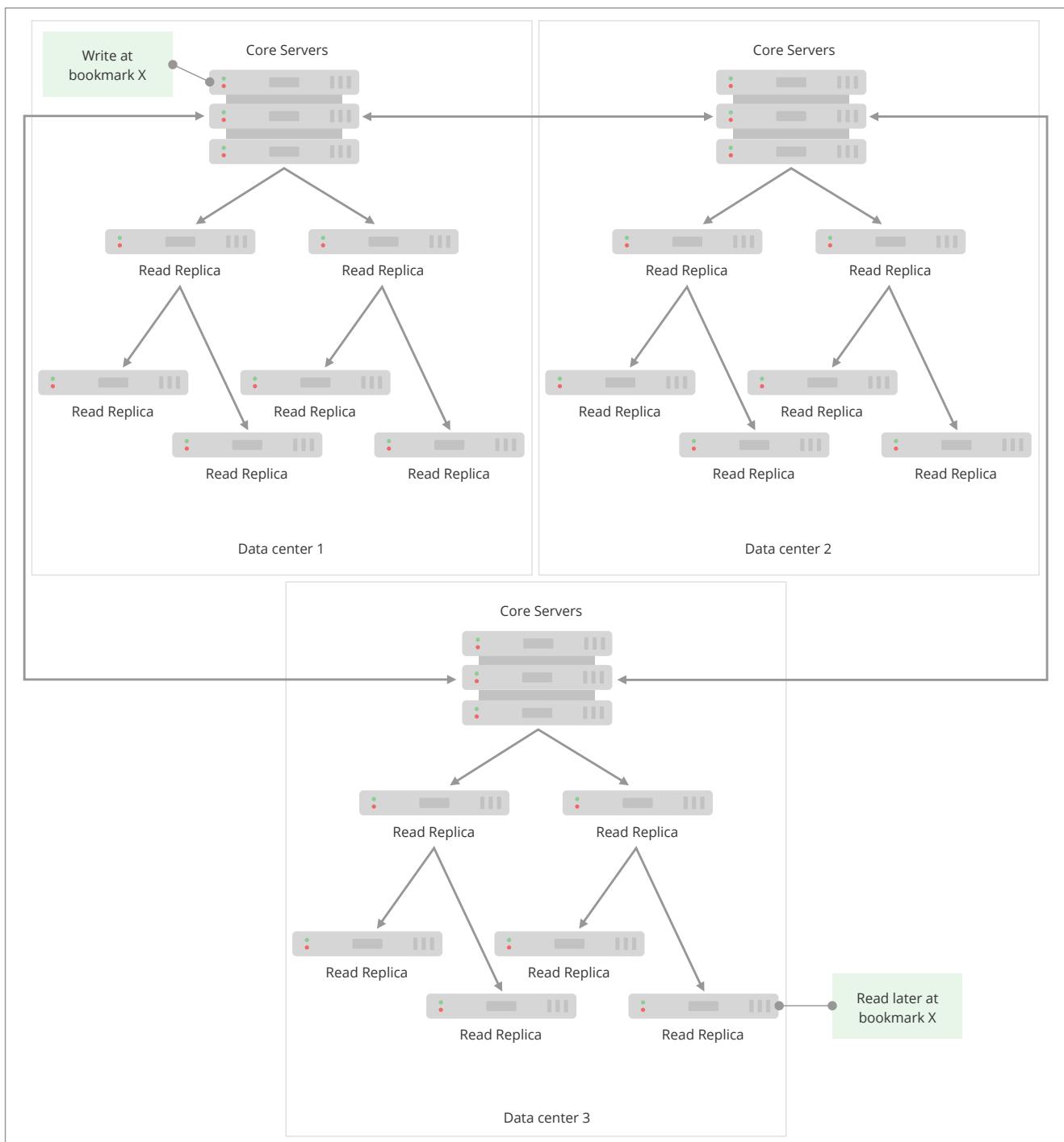


Figure 27. Each tier in the Read Replicas is further behind the source of truth, but offers greater scale-out

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.2.3. Multi-data center operations

This section shows how to configure Neo4j servers so that they are topology/data center-aware. It describes the precise configuration needed to achieve a scalable multi-data center deployment.

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](#).

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 134. 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 Design 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
<code>connect-to-random-core-server</code>	Connect to any Core Server selecting at random from those currently available.
<code>typically-connect-to-random-read-replica</code>	Connect to any available Read Replica , but around 10% of the time connect to any random Core Server.
<code>connect-randomly-to-server-group</code>	Connect at random to any available Read Replica in any of the server groups specified in the comma-separated list <code>causal_clustering.connect-randomly-to-server-group</code> .
<code>leader-only</code>	Connect only to the current Raft leader of the Core Servers.
<code>connect-randomly-within-server-group</code>	Connect at random to any available Read Replica in any of the server groups to which this server belongs. Deprecated, please use <code>connect-randomly-to-server-group</code> .

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 135. 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 `config_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.

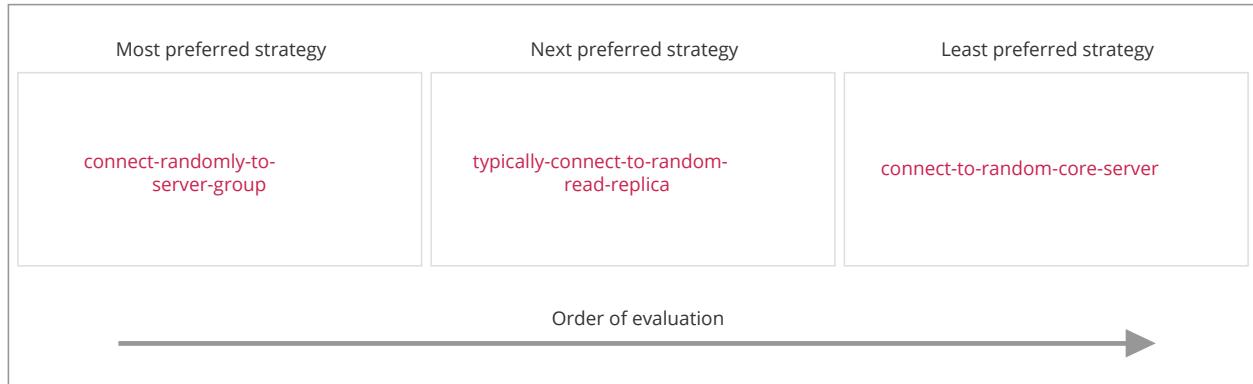


Figure 28. The first satisfactory response from a strategy will be used.

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 136. 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 29. 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 137. 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
<code>org.neo4j.causalclustering.discovery.TopologyService</code>	This is a directory service which provides access to the addresses of all servers and server groups in the cluster.
<code>org.neo4j.kernel.configuration.Config</code>	This provides the configuration from <code>neo4j.conf</code> for the local instance. Configuration for our own plugin can reside here.
<code>org.neo4j.causalclustering.identity.MemberId</code>	This provides the unique cluster <code>MemberId</code> 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_cluster.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.2.4. Multi-data center load balancing

This section describes the topology-aware load balancing options available for client applications in a multi-data center Neo4j deployment. It describes how to configure the load balancing for the cluster so that client applications can direct its workload at the most appropriate cluster members, such as those nearby.

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](#)

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.

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 [Driver Manual](#) 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.



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 30. 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 138. Specific data center only

```
causal_clustering.load_balancing.config.server_policies.north1_only=\n    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 139. Specific data center preferably

```
causal_clustering.load_balancing.config.server_policies.north1=\n    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 140. Gracefully falling back to neighbors

```
causal_clustering.load_balancing.config.server_policies.north_app1=\n    groups(north1,north2)->min(2);\n    groups(north);\n    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.2.5. Data center disaster recovery

This section describes how to recover your Neo4j Causal Cluster following a data center failure. Specifically it covers safely turning a small number of surviving instances from a read-only state back into a fully operational cluster of read/write instances.

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](#).

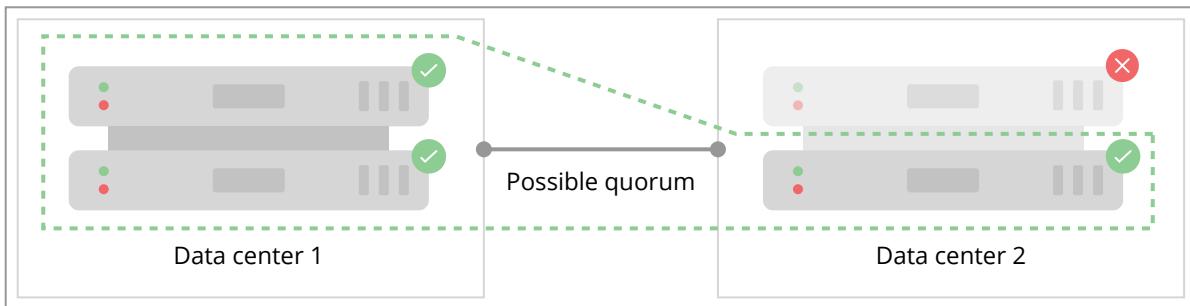


Figure 31. Two Data Center Deployment with Four Core Instances

However if an entire data center becomes offline because of some disaster, then a *majority quorum* cannot be formed in this case.



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.

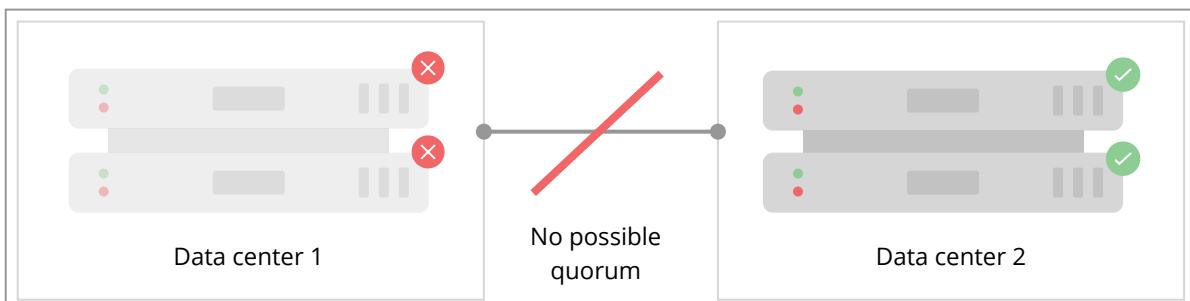


Figure 32. Data Center Loss Requires Guided Recovery

But, from a bird'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.



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.

C.3. Embedded usage

This section describes how to embed a Neo4j Causal Cluster in your application.

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.

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](#) □ [Administration](#) □ [Security](#)

See also a worked example in [Fine-grained access control](#).

D.1. Enterprise Edition

This section describes deprecated procedures for native user and role management for Neo4j 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
<code>dbms.security.activateUser</code>	Activate a suspended user	[admin]	
<code>dbms.security.addRoleToUser</code>	Assign a role to the user	[admin]	
<code>dbms.security.changePassword</code>	Change the current user's password	[reader,editor,publisher,architect,admin]	□
<code>dbms.security.changeUserPassword</code>	Change the given user's password	[admin]	
<code>dbms.security.createRole</code>	Create a new role	[admin]	
<code>dbms.security.createUser</code>	Create a new user	[admin]	□
<code>dbms.security.deleteRole</code>	Delete the specified role. Any role assignments will be removed	[admin]	
<code>dbms.security.deleteUser</code>	Delete the specified user	[admin]	□
<code>dbms.security.listRoles</code>	List all available roles	[admin]	
<code>dbms.security.listRolesForUser</code>	List all roles assigned to the specified user	[admin]	
<code>dbms.security.listUsers</code>	List all local users	[admin]	□

Procedure name	Description	Executable by role(s)	Available in Community Edition
<code>dbms.security.listUsersForRole</code>	List all users currently assigned the specified role	[admin]	
<code>dbms.security.removeRoleFromUser</code>	Unassign a role from the user	[admin]	
<code>dbms.security.suspendUser</code>	Suspend the specified user	[admin]	

D.1.1. 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
<code>username</code>	String	This is the username of the user to be activated.
<code>requirePasswordChange</code>	Boolean	This is optional, with a default of <code>true</code> . If this is <code>true</code> , (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 141. 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')
```

D.1.2. 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 142. 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')
```

D.1.3. 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.

D.1.4. 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.

Name	Type	Description
requirePasswordChange	Boolean	This is optional, with a default of <code>true</code> . If this is <code>true</code> , (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 143. 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')
```

D.1.5. 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 144. Create a new role

The following example creates a new custom role.

```
CALL dbms.security.createRole('operator')
```

D.1.6. 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 <code>true</code> . If this is <code>true</code> , (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 145. 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')
```

D.1.7. 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 146. Delete the specified role

The following example deletes the custom role 'operator' from the system.

```
CALL dbms.security.deleteRole('operator')
```

D.1.8. 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 147. Delete the specified user

The following example deletes a user with the username 'janebrown'.

```
CALL dbms.security.deleteUser('janebrown')
```

D.1.9. 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 148. 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
```

D.1.10. 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 149. 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
```

D.1.11. 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 150. 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
```

D.1.12. 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 151. 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
```

D.1.13. 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 152. 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')
```

D.1.14. 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 153. Suspend the specified user

The following example suspends a user with the username '**billjones**'.

```
CALL dbms.security.suspendUser('billjones')
```

D.2. Community Edition

This section describes deprecated procedures for user and password management for Neo4j 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

D.2.1. 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.

D.2.2. 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 <code>true</code> . If this is <code>true</code> , (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 154. 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)
```

D.2.3. 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 155. Delete a user

The following example deletes a user with the username 'janebrown'.

```
CALL dbms.security.deleteUser('janebrown')
```

D.2.4. 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 156. 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
```

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>.