



The Neo4j Operations Manual

v3.5

Table of Contents

1. Introduction	2
1.1. Editions	2
1.1.1. Community Edition	2
1.1.2. Enterprise Edition	2
1.1.3. Feature details	3
2. Installation	5
2.1. System requirements	5
2.2. Neo4j Desktop	6
2.3. Linux installation	7
2.3.1. Debian	7
Installation	7
Upgrade	9
File locations	11
Operation	11
2.3.2. RPM	11
Red Hat / CentOS / Fedora	11
SUSE	12
Non-interactive installation of Neo4j Enterprise Edition	12
Upgrade	12
2.3.3. Neo4j system service	12
Configuration	12
Starting the service automatically on system start	12
Controlling the service	13
Log	13
2.3.4. Linux tarball installation	13
Unix console application	13
Linux service	14
Setting the number of open files	14
2.4. Mac OS installation	15
2.4.1. Unix console application	15
2.4.2. Mac OS service	15
2.4.3. Mac OS file descriptor limits	15
2.5. Windows installation	16
2.5.1. Windows console application	16
2.5.2. Windows service	16
Java options	16
2.5.3. Windows PowerShell module	17
System requirements	17
Managing Neo4j on Windows	17
How do I import the module?	18
How do I get help about the module?	18
Example usage	18
Common PowerShell parameters	19
3. Docker	20
3.1. Introduction	20
3.1.1. Ports	20

3.1.2. Volumes	20
3.1.3. Running Neo4j as a non-root user	21
3.1.4. Neo4j editions	21
Neo4j Enterprise Edition license	21
3.2. Configuration	22
3.2.1. Environment variables	22
Neo4j Enterprise Edition	23
3.2.2. <i>/conf</i> volume	23
3.2.3. Building a new image	24
3.3. Clustering.....	24
3.3.1. Setting up a Causal Cluster	24
3.4. Docker specific operations	25
3.4.1. Using <i>/plugins</i>	26
3.4.2. Using Cypher shell	26
3.4.3. Upgrading Neo4j on Docker	26
3.5. Security.....	26
3.5.1. Encryption	26
4. Configuration	27
4.1. The neo4j.conf file	27
4.1.1. Introduction	27
4.1.2. Syntax	27
4.1.3. JVM specific configuration settings.....	28
4.1.4. List currently active settings	28
4.2. File locations	28
4.2.1. Where to find important files	28
4.2.2. Log files.....	29
4.2.3. File permissions.....	30
4.3. Ports	30
4.4. Set an initial password.....	31
4.5. Poll Neo4j.....	32
4.6. Usage Data Collector	32
4.6.1. Technical Information	33
4.6.2. Disable the Usage Data Collector	33
4.7. Configure Neo4j connectors	33
4.7.1. Additional options for Neo4j connectors	34
4.7.2. Additional options for Bolt connectors	35
4.7.3. Defaults for addresses	35
4.8. Dynamic settings	36
4.8.1. Introduction	36
4.8.2. Discover dynamic settings	37
4.8.3. Update dynamic settings	37
4.8.4. Dynamic settings reference.....	38
4.9. Transaction logs	39
5. Clustering	41
5.1. Introduction.....	42
5.1.1. Operational view	42
Core Servers.....	43
Read Replicas.....	44
5.1.2. Causal consistency	44

5.1.3. Summary	45
5.2. Lifecycle	46
5.2.1. Discovery protocol	46
5.2.2. Core membership	47
5.2.3. Read Replica membership	47
5.2.4. Transacting via the Raft protocol	48
5.2.5. Catchup protocol	49
5.2.6. Read Replica shutdown	50
5.2.7. Core shutdown	50
5.3. Create a new cluster	50
5.3.1. Configure a Core-only cluster	51
5.3.2. Add a Core Server to an existing cluster	52
5.3.3. Add a Read Replica to an existing cluster	53
5.3.4. Remove a Core from a cluster	53
5.3.5. Bias cluster leadership with follower-only instances	53
5.3.6. Initial discovery of cluster members	54
5.3.7. Initial discovery of cluster members with DNS	54
5.3.8. Initial discovery of cluster members with Kubernetes	55
5.3.9. Store copy	55
5.4. Seed a cluster	56
5.4.1. Seed from an online backup	56
5.4.2. Seed from an offline backup	57
5.4.3. Seed using the import tool	58
5.5. Intra-cluster encryption	59
5.5.1. Introduction	59
5.5.2. Example deployment	59
Generate and install cryptographic objects	59
Create an SSL policy	60
Configure Causal Clustering with the SSL policy	61
Validate the secure operation of the cluster	61
5.6. Multi-data center	62
5.6.1. Multi-data center design	62
Core Server deployment scenarios	63
Allowing Read Replicas to catch up from other Read Replicas	65
5.6.2. Multi-data center operations	69
Enable multi-data center operations	69
Server groups	69
Strategy plugins	70
5.6.3. Multi-data center load balancing	74
Introduction	74
Prerequisite configuration	74
The load balancing framework	75
Load balancing examples	77
5.6.4. Data center disaster recovery	78
Data center loss scenario	78
Procedure for recovering from data center loss	80
5.7. Multi-clustering	80
5.7.1. Introduction	80
Known limitations	82

5.7.2. Configure a multi-cluster	82
5.8. Settings reference	85
5.8.1. Common settings	85
5.8.2. Multi-data center settings	86
6. Upgrade	88
6.1. Upgrade planning	88
6.1.1. Supported upgrade paths	88
6.1.2. Prepare to upgrade	88
6.1.3. Additional resources	90
6.2. Upgrade a single instance	90
6.2.1. Upgrade from 2.x	90
6.2.2. Upgrade from 3.x	91
6.3. Upgrade a Causal Cluster	92
6.3.1. Important pre-upgrade information	92
6.3.2. Rolling upgrade	93
Rolling upgrade for fixed servers	94
Rolling upgrade for replaceable resources	94
6.3.3. Offline upgrade	95
7. Backup	97
7.1. Backup planning	97
7.1.1. Introduction	98
7.1.2. Online and offline backups	98
7.1.3. Storage considerations	98
7.1.4. Network protocols for backups	99
7.1.5. Memory configuration	99
7.2. Standalone databases	99
7.2.1. Configuration parameters	99
7.2.2. Running backups from a separate server	100
7.2.3. Running backups from the local server	100
7.3. Causal Clusters	101
7.3.1. Introduction	101
7.3.2. Configuration parameters	101
7.3.3. Encrypted backups	101
7.3.4. Running backups from a Read Replica	101
7.3.5. Running backups from a Core Server	102
7.3.6. Backup scenarios and examples	102
7.4. Perform a backup	103
7.4.1. Backup commands	103
7.4.2. Full backups	105
7.4.3. Incremental backups	105
7.4.4. Exit codes	106
7.5. Restore a backup	106
7.5.1. Restore commands	106
7.5.2. Restore a single database	107
7.5.3. Restore a Causal Cluster	107
8. Authentication and authorization	108
8.1. Introduction	108
8.2. Terminology	109
8.3. Configuration	110

8.4. Native user and role management	110
8.4.1. Native roles	110
8.4.2. Custom roles	112
8.4.3. Propagate users and roles	112
8.4.4. Procedures for native user and role management	113
Activate a suspended user.....	114
Assign a role to the user	114
Change the current user's password	115
Change the given user's password.....	116
Create a new role	117
Create a new user	117
Delete the specified role	118
Delete the specified user	118
List all available roles	119
List all roles assigned to the specified user.....	120
List all local users	121
List all users currently assigned the specified role	122
Unassign a role from the user.....	122
Suspend the specified user	123
Show the current user	124
List roles per procedure.....	124
8.5. Integration with LDAP	125
8.5.1. Introduction	126
8.5.2. Configure the LDAP auth provider	126
Configuration for Active Directory	126
Configuration for openLDAP	127
8.5.3. Use 'ldapsearch' to verify the configuration	128
8.5.4. The auth cache	129
8.5.5. Available methods of encryption	130
Use LDAP with encryption via StartTLS	130
Use LDAP with encrypted LDAPS	130
8.5.6. Use a self-signed certificate in a test environment	130
8.6. Subgraph access control	131
8.6.1. Introduction	131
8.6.2. Configuration steps	131
Create a custom role	131
Manage procedure permissions	132
8.7. Property-level access control	133
8.7.1. Introduction	134
8.7.2. Implement a property blacklist.....	134
9. Security	136
9.1. Securing extensions	136
9.1.1. Sandboxing	136
9.1.2. White listing	137
9.2. SSL framework	137
9.2.1. Introduction	138
9.2.2. Prerequisites	138
9.2.3. Define SSL policies	138
9.2.4. Apply SSL policies	140

9.2.5. Choosing an SSL provider	141
9.2.6. The legacy SSL system	141
9.2.7. Terminology	142
9.3. Browser credentials handling	143
9.4. Security checklist	144
10. Monitoring	146
10.1. Metrics	146
10.1.1. Expose metrics	147
Enable metrics logging	147
Graphite	147
Prometheus	148
CSV files	148
10.1.2. Metrics reference	148
General-purpose metrics	149
Metrics specific to Causal Clustering	151
Java Virtual Machine Metrics	152
10.2. Logging	152
10.2.1. Query logging	153
Log configuration	153
Attach metadata to a query	154
10.2.2. Security events logging	155
Log configuration	155
10.3. Query management	156
10.3.1. List all running queries	156
10.3.2. List all active locks for a query	158
10.3.3. Terminate multiple queries	159
10.3.4. Terminate a single query	160
10.4. Transaction management	161
10.4.1. Configure transaction timeout	161
10.4.2. Configure lock acquisition timeout	162
10.4.3. List all running transactions	162
10.5. Connection management	163
10.5.1. List all network connections	164
10.5.2. Terminate multiple network connections	165
10.5.3. Terminate a single network connection	166
10.6. Monitoring a Causal Cluster	167
10.6.1. Procedures for monitoring a Causal Cluster	168
Find out the role of a cluster member	168
Gain an overview over the instances in the cluster	168
Get routing recommendations	169
10.6.2. Endpoints for status information	170
Adjusting security settings for Causal Clustering endpoints	170
Unified endpoints	170
Endpoints for Core Servers	173
Endpoints for Read Replicas	173
10.6.3. Monitoring a multi-cluster	174
Gain an overview over all the instances in a multi-cluster	174
Get routing information for all databases in a multi-cluster	175
Get routing information for a given database in a multi-cluster	175

11. Performance.....	177
11.1. Memory configuration.....	177
11.1.1. Overview	177
11.1.2. Considerations	178
11.1.3. Capacity planning	179
11.2. Index configuration	180
11.2.1. Introduction.....	181
11.2.2. Schema indexes	181
Introduction	182
Index providers	182
Limitations of native indexes	183
Limitations of Lucene-backed indexes.....	187
Upgrade considerations	188
11.2.3. Fulltext schema indexes	188
Introduction	188
Configuration.....	188
Deprecation of explicit indexes.....	189
11.3. Tuning of the garbage collector	190
11.4. Bolt thread pool configuration.....	191
11.4.1. How thread pooling works	191
11.4.2. Configuration options	192
11.4.3. How to size your Bolt thread pool	192
11.5. Compressed storage	192
11.6. Linux file system tuning	194
11.7. Disks, RAM and other tips.....	195
11.7.1. Storage	195
11.7.2. Page Cache	195
Active Page Cache Warmup.....	196
Checkpoint IOPS Limit	196
11.8. Statistics and execution plans	196
11.8.1. Statistics	196
11.8.2. Execution plans.....	197
12. Tools	199
12.1. Neo4j Admin	199
12.1.1. Introduction.....	199
12.1.2. Syntax and commands	199
12.1.3. Environment variables.....	200
12.1.4. Exit codes.....	201
12.2. Consistency checker	201
12.2.1. Check consistency of a database or a backup	201
12.3. Report tool	202
12.4. Display store information	204
12.5. Memory recommendations	205
12.6. Import	205
12.6.1. Syntax	206
12.6.2. CSV file header format.....	207
Header files	207
Properties.....	207
Node files	209

Relationship files	210
Using ID spaces	211
Skipping columns	212
Import compressed files	213
12.6.3. Options	213
Output	215
12.7. Dump and load databases	215
12.8. Unbind a Core Server	216
12.9. Cypher Shell	217
12.9.1. Invoking Cypher Shell	217
12.9.2. Query parameters	219
12.9.3. Transactions	219
12.9.4. Procedures	220
Appendix A: Reference	222
A.1. Configuration settings	222
A.2. Built-in procedures	273
A.2.1. Procedures, editions and modes	273
A.2.2. Procedure reference	274
A.2.3. Enterprise Edition procedures	274
A.2.4. Community Edition procedures	283
A.3. User management for Community Edition	288
A.3.1. Change the current user's password	289
A.3.2. Add a user	289
A.3.3. Delete a user	290
A.3.4. List all users	291
A.3.5. Show details for the current user	291
Appendix B: Tutorials	293
B.1. Set up a local Causal Cluster	293
B.1.1. Download and configure	293
B.1.2. Configure the Core instances	294
Minimum configuration	294
Additional configuration	294
B.1.3. Start the Neo4j servers	295
B.1.4. Check the status of the cluster	295
B.1.5. Test the cluster	296
B.1.6. Configure the Read Replicas	296
Minimum configuration	296
Additional configuration	297
B.1.7. Test the cluster with Read Replicas	297
B.2. Use the Import tool	298
B.2.1. Basic example	298
B.2.2. Customizing configuration options	299
B.2.3. Using separate header files	300
B.2.4. Multiple input files	301
Using regular expressions for specifying multiple input files	302
B.2.5. Types and labels	303
Using the same label for every node	303
Using the same relationship type for every relationship	304
B.2.6. Property types	304

B.2.7. ID handling	305
Working with sequential or auto incrementing identifiers	305
B.2.8. Bad input data.....	306
Relationships referring to missing nodes	306
Multiple nodes with same id within same id space	307
B.3. Manage users and roles	308
B.3.1. Creating a user and managing roles	308
B.3.2. Suspending and reactivating a user.....	310
Appendix C: HA cluster	312
C.1. Architecture.....	312
C.1.1. Arbiter instance.....	313
C.1.2. Transaction propagation	313
C.1.3. Failover.....	313
C.1.4. Branching.....	314
C.1.5. Summary	314
C.2. Setup and configuration	315
C.2.1. Basic installation.....	315
C.2.2. Important configuration settings	315
C.2.3. Arbiter instances	317
C.2.4. HAProxy for load balancing.....	317
Configuring HAProxy for the Bolt Protocol	318
Configuring HAProxy for the HTTP API	319
Optimizing for reads and writes	320
Cache-based sharding with HAProxy	320
C.3. Highly Available mode on Docker	321
C.4. Status information	322
C.4.1. Introduction.....	322
C.4.2. The endpoints	322
C.4.3. Examples	323
C.5. Upgrade	324
C.6. Backup and restore	325
C.7. Metrics	326
C.8. Tutorials	326
C.8.1. Set up an HA cluster	326
Download and configure	327
Start the Neo4j Servers	328
C.8.2. Set up a local HA cluster	329
Download and configure	329

This is the operations manual for Neo4j version 3.5, 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.
- [Docker](#) — Instructions on how to use Neo4j on Docker.
- [Configuration](#) — Instructions on how to configure certain parts of the product.
- [Clustering](#) — Comprehensive descriptions of Neo4j Causal Clustering.
- [Upgrade](#) — Instructions on upgrading Neo4j.
- [Backup](#) — Instructions on setting up Neo4j backups.
- [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.
- [HA cluster](#) — Information about Neo4j HA clusters.

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

This chapter introduces Neo4j.

Neo4j is the world's leading graph database. It is built *from the ground up* to be a graph database, meaning that its architecture is designed for optimizing fast management, storage, and the traversal of nodes and relationships. Therefore, relationships are described as *first class citizens* in Neo4j.

In the world of relational databases the performance of a *join* operation will degrade exponentially with the number of relationships. However, in Neo4j the corresponding action is performed as navigation from one node to another; an operation whose performance is linear.

This different approach to storing and querying connections between entities provides traversal performance of up to four million hops per second and core. Since most graph searches are local to the larger neighborhood of a node, the total amount of data stored in a database will not affect operations runtime. Dedicated memory management, and highly scalable and memory efficient operations, contribute to the benefits.

The property graph approach is also *whiteboard friendly*. By this we mean that the schema-optimal model of Neo4j provides for a consistent use of the same model throughout conception, design, implementation, storage, and visualization. A major benefit of this is that it allows all business stakeholders to participate throughout the development cycle. Additionally, the domain model can be evolved continuously as requirements change, without the penalty of expensive schema changes and migrations.

Cypher, the declarative graph query language, is designed to visually represent graph patterns of nodes and relationships. This highly capable, yet easily readable, query language is centered around the patterns that express concepts and questions from a specific domain. Cypher can also be extended for narrow optimizations for specific use cases.

Neo4j can store trillions of entities for the largest datasets imaginable while being sensitive to compact storage. For production environments it can be deployed as a scalable, fault-tolerant cluster of machines. Due to its high scalability, Neo4j clusters require only tens of machines, not hundreds or thousands, saving on cost and operational complexity. Other features for production applications include hot backups and extensive monitoring.

1.1. Editions

There are two editions of Neo4j to choose from: [Community Edition](#) and [Enterprise Edition](#):

1.1.1. 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 compliance, Cypher, and programming APIs. It is ideal for learning Neo4j, for do-it-yourself projects, and for applications in small workgroups.

1.1.2. Enterprise Edition

The Enterprise Edition extends the functionality of Community Edition to include key features for performance and scalability, such as a clustering architecture for high availability 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.

1.1.3. Feature details

Table 1. Features

Edition	Community	Enterprise
Labeled property graph model	□	□
Native graph processing & storage	□	□
ACID transactions	□	□
Cypher graph query language	□	□
Neo4j Browser with syntax highlighting	□	□
Bolt binary protocol	□	□
Language drivers for C#, Java, JavaScript & Python	□	□
High-performance native API	□	□
High-performance caching	□	□
Cost-based query optimizer	□	□
Graph algorithms library to support AI initiatives	□	□
Fast writes via native label indexes	□	□
Composite indexes	□	□
Fulltext node & relationship indexes	-	□
<i>Slotted</i> and <i>Compiled</i> Cypher runtimes	-	□
Property-existence constraints	-	□
Node Key schema constraints	-	□
Listing and terminating running queries	-	□
Auto-reuse of space	-	□
Role-based access control	-	□
Subgraph access control	-	□
Property-level security	-	□
LDAP and Active Directory integration	-	□
Kerberos security option	-	□

Table 2. Performance & Scalability

Edition	Community	Enterprise
Causal Clustering for global scale applications	-	□
Multi-clustering	-	□
Enterprise lock manager accesses all cores on server	-	□
Intra-cluster encryption	-	□
Offline backups	□	□
Online backups	-	□
Encrypted backups	-	□

Edition	Community	Enterprise
Rolling upgrades	-	□
Automatic cache warming	-	□
Routing and load balancing with Neo4j Drivers	-	□
Advanced monitoring	-	□
Graph size limitations	34B nodes, 34B relationships, 68B 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, Mac OS, Windows, Debian, Docker, or with CAPI Flash.

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.
- [Mac OS](#) — Installation instructions for Mac OS.
- [Windows](#) — Installation instructions for Windows.

As an alternative to installation, you can also run Neo4j 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.

CPU

Performance is generally memory or I/O bound for large graphs, and compute bound for graphs that fit in memory.

Minimum

Intel Core i3

Recommended

Intel Core i7

IBM POWER8

Memory

More memory allows for larger graphs, but it needs to be configured properly to avoid disruptive garbage collection operations. See [Memory configuration](#) for suggestions.

Minimum

2GB

Recommended

16–32GB or more

Disk

Aside from capacity, the performance characteristics of the disk are the most important when selecting storage. Neo4j workloads tend significantly toward random reads. Select media with low average seek time: SSD over spinning disks. Consult [Disks, RAM and other tips](#) for more details.

Minimum

10GB SATA

Recommended

SSD w/ SATA Express, or NVMe

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.

Minimum

EXT4 (or similar)

Recommended

EXT4, ZFS

Software

[Neo4j Desktop](#) includes a Java Virtual Machine, *JVM*, for convenience. All other versions of Neo4j require Java to be pre-installed. The minimum requirement is the Java Runtime Environment, *JRE*.

The Java Development Kit, *JDK*, is required in order to run some commands of the [Report tool](#). This is a tool that lets you send logs and performance data to Neo4j Customer Success in a simple and convenient way, in case of a production issue. For this reason, JDK is recommended for all production environments that allow it.

Java

[OpenJDK 8](#) [1: [Zulu OpenJDK](#) (<https://www.azul.com/downloads/zulu/>) or a [Debian distribution](#) (<http://openjdk.java.net/>.)]

[Oracle Java 8](#) (<http://www.oracle.com/technetwork/java/javase/downloads/index.html>)

[IBM Java 8](#) (<http://www.ibm.com/developerworks/java/jdk/>)



Java 11 is supported as a runtime platform, but any custom provided code should remain Java 8 compatible.

Operating Systems

Ubuntu 14.04, 16.04; Debian 8, 9; CentOS 6, 7; Fedora, Red Hat, Amazon Linux

Additionally, SUSE is reported to work well with Neo4j, but it is not officially certified for production use.

Windows Server 2012, 2016

Architectures

x86

OpenPOWER (POWER8)

2.2. Neo4j Desktop

This section introduces Neo4j Desktop.

Neo4j Desktop is an easy and convenient way for developers to work with local Neo4j databases.

To install Neo4j Desktop, go to [Neo4j Download Center](https://neo4j.com/download-center/) (<https://neo4j.com/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](#)
- [Deploy Neo4j using the Neo4j RPM package](#)
- [Install Neo4j as a system service](#)
- [Install Neo4j on Linux from a tarball](#)

2.3.1. Debian

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

This section describes the following:

- [Installation](#)
 - [Prerequisites \(Ubuntu 14.04 and Debian 8 only\)](#)
 - [Add the repository](#)
 - [Install Neo4j](#)
- [Upgrade](#)
 - [Upgrade from Neo4j 3.x](#)
 - [Upgrade from Neo4j 2.3](#)
- [File locations](#)
- [Operation](#)

Installation

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

- A Java 8 runtime is installed.
- The repository containing the Neo4j Debian package is known to the package manager.

Prerequisites (Debian 8 and Ubuntu 14.04 only)

Neo4j 3.5 requires the Java 8 runtime. Java 8 is not included in Ubuntu 14.04 LTS or Debian 8 (jessie) and will have to be installed manually prior to installing or upgrading to Neo4j 3.5, as described below. Debian users can find OpenJDK 8 in [backports](https://packages.debian.org/jessie-backports/openjdk-8-jdk) (<https://packages.debian.org/jessie-backports/openjdk-8-jdk>).

Java 8 on Debian 8

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

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

- For Java 8 to install with Neo4j, first install the `ca-certificates-java` package:

```
sudo apt-get -t jessie-backports install ca-certificates-java
```

You are now ready to install Neo4j 3.5.0, which will install Java 8 automatically if it is not already installed. See [Dealing with multiple installed Java versions](#) to make sure you can start Neo4j after install.

Java 8 on Ubuntu 14.04

- Users on Ubuntu 14.04 can add Oracle Java 8 via WebUpd8. Note that when installing from WebUpd8 or any other PPA, you must install Java 8 manually before installing Neo4j. Otherwise there is a risk that Java 9 will be installed in, which is not compatible with Neo4j.

```
sudo add-apt-repository ppa:webupd8team/java  
sudo apt-get update  
sudo apt-get install oracle-java8-installer
```

- Once 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 8, or Neo4j 3.5.0 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-1.7.0-openjdk-amd64 1071 /usr/lib/jvm/java-1.7.0-openjdk-amd64  
java-1.8.0-openjdk-amd64 1069 /usr/lib/jvm/java-1.8.0-openjdk-amd64
```

- Identify your Java 8 version, in this case it is `java-1.8.0-openjdk-amd64`. Then set it as the default with (replacing `<java8name>` with the appropriate name from above)

```
sudo update-java-alternatives --jre --set <java8name>
```

Add the repository

- The Debian package is available from <https://debian.neo4j.org>. To use the repository, follow these steps:

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

Install Neo4j

To install Neo4j Community Edition:

```
sudo apt-get install neo4j=1:3.5.0
```

To install Neo4j Enterprise Edition:

```
sudo apt-get install neo4j-enterprise=1:3.5.0
```

Note that the version includes an epoch version component (`1:`), in accordance with the [Debian policy on versioning](https://www.debian.org/doc/debian-policy/#s-f-version) (<https://www.debian.org/doc/debian-policy/#s-f-version>).

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
```

Upgrade

Upgrade from Neo4j 3.x

For upgrade of any 3.x version of Neo4j to 3.5.0, follow instructions in [Upgrade](#).

Upgrade from Neo4j 2.3

Follow the steps below when upgrading a Neo4j Debian/Ubuntu installation:

1. Upgrade to Neo4j 3.2:
 - a. Migrate the configuration files.

The configuration files changed between Neo4j versions 2.3 and 3.2. If you have not edited the configuration files, the Debian package will simply remove the files that are no longer necessary, and replace the old default files.

If you have changed configuration values in your 2.3 installation, you can use the provided config migration tool. Two arguments are provided to tell the config migrator where to find the `conf`/ directory for the source and the destination. Both must be provided, due to the filesystem layout of the Debian packages.

Because the Neo4j files and directories are owned by the `neo4j` user and `adm` group on Debian, it is necessary to use `sudo` to make sure the permissions remain intact:

```
sudo -u neo4j -g adm java -jar /usr/share/neo4j/bin/tools/2.x-config-migrator.jar /var/lib/neo4j /var/lib/neo4j
```

b. Import the Neo4j 2.3 database to Neo4j 3.2.

The location of the database changed between Neo4j versions 2.3 and 3.2. The Neo4j 2.3 databases will need to be imported to Neo4j 3.2. To do this, use the `neo4j-admin import` command.

For example, to import a database called `graph.db` (the default database name in Neo4j 2.3), use the following command:

```
sudo -u neo4j neo4j-admin import --mode=database --database=graph.db --from=/var/lib/neo4j/data/graph.db
```

This command will import the database located in `/var/lib/neo4j/data/graph.db` into Neo4j 3.2, and call it `graph.db`.

Once a database has been imported, and the upgrade has completed successfully, the old database can be removed safely.

c. Migrate the Neo4j 2.3 database to Neo4j 3.2.

The previous import step moved the database from its old on-disk location to the new on-disk location, but it did not upgrade the store format. To do this, you must start the database service with the option to migrate the database format to the latest version.

In `neo4j.conf` uncomment the option `dbms.allow_upgrade=true`. You can use the following command to change the line in-place if you have it commented out already, as it is in the default configuration:

```
sudo sed -i 's/#dbms.allow_upgrade=true/dbms.allow_upgrade=true/' /etc/neo4j/neo4j.conf
```

Start the database service with the format migration option enabled, and the format migration will take place immediately:

```
sudo service neo4j start
```

2. Upgrade to Neo4j 3.5.0.

Upgrade to Neo4j 3.5.0 by following the instructions in [Upgrade](#).

File locations

File locations for all Neo4j packages are documented [here](#).

Operation

Most Neo4j configuration goes into [neo4j.conf](#). 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. See this page for details.

2.3.2. RPM

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

Red Hat / CentOS / Fedora

For distros like Red Hat, CentOS, Fedora, and Amazon Linux the steps are as follows:

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

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

2. Install Neo4j.

- To install Neo4j Community Edition as `root`:

```
yum install neo4j-3.5.0
```

- To install Neo4j Enterprise Edition as `root`:

```
yum install neo4j-enterprise-3.5.0
```

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

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

SUSE



SUSE is not certified for production use. These instructions are provided for convenience for those wishing to use Neo4j in non-production environments.

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.org/stable neo4j-repository
```

2. Install Neo4j.

- To install Neo4j Community Edition as `root`:

```
zypper install neo4j-3.5.0
```

- To install Neo4j Enterprise Edition as `root`:

```
zypper install neo4j-enterprise-3.5.0
```

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-3.5.0
```

Upgrade

For upgrade of any 3.x version of Neo4j to Neo4j 3.5.0, follow instructions in [Upgrade](#).

2.3.3. 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.

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.

2.3.4. 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](https://neo4j.com/download-center/) (<https://neo4j.com/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 are interested in running Neo4j as a system service, please install one of our [Debian](#) or [RPM](#) packages instead.

Setting the number of open files

Linux platforms impose an upper limit on the number of concurrent files a user may have open. This number is reported for the current user and session with the `ulimit -n` command:

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

The usual default of 1024 is often not enough. This is especially true when many indexes are used or a server installation sees too many connections. Network sockets count against the limit as well. Users are therefore encouraged to increase the limit to a healthy value of 40 000 or more, depending on usage patterns. It is possible to set the limit with the `ulimit` command, but only for the root user, and it only affects the current session. To set the value system wide, follow the instructions for your platform.

What follows is the procedure to set the open file descriptor limit to 40 000 for user *neo4j* under Ubuntu 10.04 and later.



If you opted to run the Neo4j service as a different user, change the first field in step 2 accordingly.

1. Become root, since all operations that follow require editing protected system files.

```
user@localhost:~$ sudo su -
Password:
root@localhost:~$
```

2. Edit `/etc/security/limits.conf` and add these two lines:

```
neo4j soft    nofile 40000
neo4j hard   nofile 40000
```

3. Edit `/etc/pam.d/su` and uncomment or add the following line:

```
session required pam_limits.so
```

4. A restart is required for the settings to take effect.

After the above procedure, the Neo4j user will have a limit of 40 000 simultaneous open files. If you continue experiencing exceptions on `Too many open files` or `Could not stat() directory`, you may have to raise the limit further.

2.4. Mac OS installation

This section describes how to install Neo4j on Mac OS.

2.4.1. Unix console application

1. Download the latest release from [Neo4j Download Center](https://neo4j.com/download-center/) (<https://neo4j.com/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. Mac OS service

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

2.4.3. Mac OS 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 Mac OS system can be inspected with the `launchctl limit maxfiles` command. The method for changing the limit may differ depending on the version of Mac OS. 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 Mac OS. 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](https://neo4j.com/download-center/) (<https://neo4j.com/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](#) (<https://neo4j.com/download/other-releases/#releases>) 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](#) (<https://technet.microsoft.com/en-us/library/hh847748.aspx>).

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

How do I import the module?

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

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

This will add the module to the current session.

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

```
Invoke-Neo4j console
```

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

How do I get help about the module?

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

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

The output should be similar to the following:

CommandType	Name	Version	Source
Function	Invoke-Neo4j	3.5.0	Neo4j-Management
Function	Invoke-Neo4jAdmin	3.5.0	Neo4j-Management
Function	Invoke-Neo4jBackup	3.5.0	Neo4j-Management
Function	Invoke-Neo4jImport	3.5.0	Neo4j-Management
Function	Invoke-Neo4jShell	3.5.0	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. 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 does not run natively on OS X or Windows. For running Docker on OS X and Windows, please consult the [documentation provided by Docker](https://docs.docker.com/engine/installation) (<https://docs.docker.com/engine/installation>).

3.1. Introduction

An introduction to how Neo4j runs in a Docker container.

The Neo4j Docker image, and instructions on how to start using it, can be found here:
https://hub.docker.com/_/neo4j/.

3.1.1. Ports

By default, the Docker image exposes three ports for remote access:

- [7474](#) for HTTP.
- [7473](#) for HTTPS.
- [7687](#) for Bolt.

Note that when Docker is used, Neo4j is configured automatically to allow remote access to the HTTP, HTTPS, and Bolt services. This is different than the default Neo4j configuration, where the HTTP, HTTPS, and Bolt services do not allow remote connections. For more information on configuring connections, see [Configure Neo4j connectors](#).

3.1.2. Volumes

The Docker image also exposes the following volumes. Directories on the host can be mounted using the `--volume` option. See [File locations](#) for details about the directories used by default in different Neo4j distributions.

- `/conf`
- `/data`
- `/import`
- `/logs`
- `/metrics`
- `/plugins`

- /ssl/

It is often desirable to keep database and logs outside of the container. The following command will start a container with ports for Bolt and HTTP published, and with the `/data` and `/logs` volumes mapped to directories on the host.

```
docker run \
--publish=7474:7474 --publish=7687:7687 \
--volume=$HOME/neo4j/data:/data \
--volume=$HOME/neo4j/logs:/logs \
neo4j:3.4
```

Point your browser at `http://localhost:7474` on Linux or `http://$(docker-machine ip default):7474` on OS X.

All the volumes in this documentation are stored under `$HOME` in order to work on OS X where `$HOME` is automatically mounted into the machine VM. On Linux the volumes can be stored anywhere.



By default Neo4j requires authentication and requires you to login with `neo4j/neo4j` at the first connection and set a new password. You can set the password for the Docker container directly by specifying `--env NE04J_AUTH=neo4j/<password>` in your run directive. Alternatively, you can disable authentication by specifying `--env NE04J_AUTH=none` instead.

3.1.3. Running Neo4j as a non-root user

For security reasons Neo4j will run 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 would run 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:3.4
```

3.1.4. Neo4j editions

Tags are available for both Community Edition and Enterprise Edition. Version-specific Enterprise Edition tags have an `-enterprise` suffix, for example `neo4j:3.4.0-enterprise`. Community Edition tags have no suffix, for example `neo4j:3.4.0`. The latest Neo4j Enterprise Edition release is available as `neo4j:enterprise`.

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
(<mailto: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
```

3.2. 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 [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.

3.2.1. 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:3.4
```

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=NE04J_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=NE04J_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"
```

Neo4j Enterprise Edition

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 [Clustering](#) for examples of how to configure Causal Clustering on Docker.

3.2.2. /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:3.4
```

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.connectors.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:3.4 dump-config
```

3.2.3. Building a new image

For more complex customization of the image you can create a new image based on this one.

```
FROM neo4j:3.4
```

If you need to make your own configuration changes, we provide a hook so you can do that in a script:

```
COPY extra_conf.sh /extra_conf.sh
```

Then you can pass in the `EXTENSION_SCRIPT` environment variable at runtime to source the script:

```
docker run -e "EXTENSION_SCRIPT=/extra_conf.sh" cafe12345678
```

When the extension script is sourced, the current working directory will be the root of the Neo4j installation.

3.3. Clustering

This chapter describes how to set up Causal Clustering when running Neo4j in a Docker container.

3.3.1. Setting up a Causal Cluster

In order to run Neo4j in CC mode under Docker you need to wire up the containers in the cluster so that they can talk to each other. Each container must have a network route to each of the others and the `NEO4J_causal_clustering_expected_core_cluster_size` and `NEO4J_causal_clustering_initial_discovery_members` environment variables must be set for cores. Read Replicas only need to define `NEO4J_causal_clustering_initial_discovery_members`.

Within a single Docker host, this can be achieved as follows. Note that the default ports for HTTP, HTTPS and Bolt are used. For each container, these ports are mapped to a different set of ports on the Docker host.

```

docker network create --driver=bridge cluster

docker run --name=core1 --detach --network=cluster \
--publish=7474:7474 --publish=7473:7473 --publish=7687:7687 \
--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 \
neo4j:3.4-enterprise

docker run --name=core2 --detach --network=cluster \
--publish=8474:7474 --publish=8473:7473 --publish=8687:7687 \
--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 \
neo4j:3.4-enterprise

docker run --name=core3 --detach --network=cluster \
--publish=9474:7474 --publish=9473:7473 --publish=9687:7687 \
--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 \
neo4j:3.4-enterprise

```

Additional instances can be added to the cluster in an ad-hoc fashion. A Read Replica can for example be added with:

```

docker run --name=read_replica1 --detach --network=cluster \
--publish=10474:7474 --publish=10473:7473 --publish=10687:7687 \
--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 \
neo4j:3.4-enterprise

```

When each container is running on its own physical machine and Docker network is not used, it is necessary to define the advertised addresses to enable communication between the physical machines. Each container should also bind to the host machine's network.

Each instance would then be invoked similar to:

```

docker run --name=neo4j-core --detach \
--network=host \
--publish=7474:7474 --publish=7687:7687 \
--publish=5000:5000 --publish=6000:6000 --publish=7000:7000 \
--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 \
neo4j:3.4-enterprise

```

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

See [Create a new cluster](#) for more details of Neo4j Casual Clustering.

3.4. Docker specific operations

This chapter describes various operations that are specific to running Neo4j in a Docker container.

3.4.1. Using `/plugins`

To install user-defined procedures, provide a `/plugins` volume containing the jars.

```
docker run --publish=7474:7474 --publish=7687:7687 --volume=$HOME/neo4j/plugins:/plugins neo4j:3.4
```

See [Java Reference □ Procedures](#) for more details on procedures.

3.4.2. Using Cypher shell

The Neo4j shell can be run locally within a container using a command like this:

```
docker exec --interactive --tty <container> bin/cypher-shell
```

3.4.3. Upgrading Neo4j on Docker

To enable upgrades, set `NEO4J_dbms_allow__upgrade` to `true`. For more details on upgrading, see:

- [Single-instance upgrade](#)
- [Upgrade a Neo4j Causal Cluster](#)

3.5. Security

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

3.5.1. Encryption

The Docker image can expose Neo4j's native TLS support. To use your own key and certificate, provide an `/ssl` volume with the key and certificate inside. The files must be called `neo4j.key` and `neo4j.cert`. You must also publish port `7473` to access the HTTPS endpoint.

```
docker run --publish=7473:7473 --publish=7687:7687 --volume=$HOME/neo4j/ssl:/ssl neo4j:3.4
```

Chapter 4. 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.
- [Poll Neo4j](#) — How to poll for Neo4j started status.
- [Usage Data Collector](#) — Information about the Usage Data Collector.
- [Configure Neo4j connectors](#) — How to configure Neo4j connectors.
- [Configure dynamic settings](#) — How to configure certain Neo4j parameters while Neo4j is running.
- [Transaction logs](#) — How to configure transaction logs.

4.1. The `neo4j.conf` file

This section introduces the `neo4j.conf` file, and its syntax.

This section contains the following:

- [Introduction](#)
- [Syntax](#)
- [JVM specific configuration settings](#)
- [List currently active settings](#)

4.1.1. 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 [The locations of important files](#).

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.

4.1.2. 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.
- 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.

4.1.3. JVM specific configuration settings

- `dbms.memory.heap.initial_size`
- `dbms.memory.heap.max_size`
- `dbms.jvm.additional`

4.1.4. List currently active settings

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

Example 2. List currently active configuration settings

```
CALL dbms.listConfig()
YIELD name, value
WHERE name STARTS WITH 'dbms.'
RETURN name, value
ORDER BY name
LIMIT 4;
```

```
+-----+
| name          | value   |
+-----+
| "dbms.active_database" | "graph.db" |
| "dbms.allow_format_migration" | "false" |
| "dbms.allow_upgrade" | "false" |
| "dbms.checkpoint" | "periodic" |
+-----+
4 rows
```

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

4.2. File locations

This section provides an overview of where files are stored in the different Neo4j distributions, and the necessary file permissions for running Neo4j.

This section describes the following:

- [Where to find important files](#)
- [Log files](#)
- [File permissions](#)

4.2.1. Where to find important files

The table below lists the location of important files, and whether the location is customizable:

Table 3. The locations of important files

Package	Configuration [1]	Data [2]	Logs	Metrics	Import	Bin	Lib	Plugins
Linux or OS X tarball	<neo4j-home>/conf/neo4j.conf	<neo4j-home>/data	<neo4j-home>/logs	<neo4j-home>/metrics	<neo4j-home>/import	<neo4j-home>/bin	<neo4j-home>/lib	<neo4j-home>/plugins
Windows zip	<neo4j-home>/conf\neo4j.conf	<neo4j-home>\data	<neo4j-home>\logs	<neo4j-home>\metrics	<neo4j-home>\import	<neo4j-home>\bin	<neo4j-home>\lib	<neo4j-home>\plugins
Debian	/etc/neo4j/neo4j.conf	/var/lib/neo4j/data	/var/log/neo4j/Data [3]	/var/lib/neo4j/metrics	/var/lib/neo4j/import	/usr/bin	/usr/share/neo4j/lib	/var/lib/neo4j/plugins
RPM	/etc/neo4j/neo4j.conf	/var/lib/neo4j/data	/var/log/neo4j/Data [3]	/var/lib/neo4j/metrics	/var/lib/neo4j/import	/usr/bin	/usr/share/neo4j/lib	/var/lib/neo4j/plugins
Windows desktop	%APPDATA%\Neo4j Community Edition\neo4j.conf	%APPDATA%\Neo4j Community Edition	%APPDATA%\Neo4j Community Edition\logs	%APPDATA%\Neo4j Community Edition\metrics	%APPDATA%\Neo4j Community Edition\import	%ProgramFiles%\Neo4j CE 3.5\b\bin	(in package)	%ProgramFiles%\Neo4j CE 3.5\plugins
OS X desktop	\${HOME}/Documents/Neo4j/neo4j.conf	\${HOME}/Documents/Neo4j	\${HOME}/Documents/Neo4j/logs	\${HOME}/Documents/Neo4j/metrics	\${HOME}/Documents/Neo4j/import	(in package)	(in package)	(in package)
Customizable by option	See Configuration of <neo4j-home> and conf	dbms.directories.data	dbms.directories.logs	dbms.directories.metrics	dbms.directories.import	Not applicable	dbms.directories.lib	dbms.directories.plugins

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

[2] Please note that 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 `journalctl --unit=neo4j`.

The locations of `<neo4j-home>` and `conf` can be configured using environment variables, as described below:

Table 4. Configuration of `<neo4j-home>` and `conf`

Location	Default	Environment variable	Notes
<code><neo4j-home></code>	parent of <code>bin</code>	<code>NEO4J_HOME</code>	Must be set explicitly if <code>bin</code> is not a subdirectory.
<code>conf</code>	<code><neo4j-home>/conf</code>	<code>NEO4J_CONF</code>	Must be set explicitly if it is not a subdirectory of <code><neo4j-home></code> .

4.2.2. Log files

Filename	Description
<code>neo4j.log</code>	The standard log, where general information about Neo4j is written. Not written for Debian and RPM packages. See relevant sections.
<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. (Enterprise Edition only.)

Filename	Description
<code>security.log</code>	Log of security events. (Enterprise Edition only.)
<code>service-error.log</code>	Log of errors encountered when installing or running the Windows service. (Windows only.)

4.2.3. File permissions

The user that Neo4j runs as must have the following permissions:

Read only

- `conf`
- `import`
- `bin`
- `lib`
- `plugins`

Read and write

- `data`
- `logs`
- `metrics`

Execute

- all files in `bin`

4.3. Ports

This section lists ports relevant to a Neo4j installation.

This section provides an overview for determining which Neo4j-specific ports should be opened up in your firewalls. 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.

Name	Default port number	Related settings	Comments
Backups	6362–6372	<code>dbms.backup.enabled</code> <code>dbms.backup.address</code>	Backups are enabled by default. In production environments, external access to the backup port(s) should be blocked by a firewall. See also Backup .
HTTP	7474	See Configure Neo4j connectors .	It is recommended to not open up this port for external access in production environments, since traffic is unencrypted. Used by Neo4j Browser. Also used by REST API.
HTTPS	7473	See Configure Neo4j connectors .	Also used by REST API.
Bolt	7687	See Configure Neo4j connectors .	Used by Cypher Shell and by Neo4j Browser.

Name	Default port number	Related settings	Comments
Causal Cluster	5000, 6000, 7000	causal_clustering.discovery_listen_address causal_clustering.transaction_listen_address causal_clustering.raft_listen_address	The listed ports are the default ports in <code>neo4j.conf</code> . The ports are likely be different in a production installation; therefore the potential opening of ports must be modified accordingly. See also Settings reference .
HA Cluster	5001, 6001	ha.host.coordination ha.host.data	The listed ports are the default ports in <code>neo4j.conf</code> . The ports will most likely be different in a production installation; therefore the potential opening of ports must be modified accordingly. See also HA cluster .
Graphite monitoring	2003	metrics.graphite.server	This is an outbound connection in order for the Neo4j database to communicate with the Graphite server. See also Metrics .
Prometheus monitoring	2004	metrics.prometheus.enabled and metrics.prometheus.endpoint	See also Metrics .
JMX monitoring	3637	dbms.jvm.additional=-Dcom.sun.management.jmxremote.port=3637	This setting is for exposing the JMX. This is not the recommended way of inspecting a Neo4j database. It is not enabled by default.

4.4. 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>
```

Example 3. 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
```

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.

4.5. Poll Neo4j

This section gives an example of how to poll Neo4j, in order to know when it is ready for requests after starting up.

After starting Neo4j it may take some time before the database is ready to serve requests. Systems that depend on the database should be able to retry if it is unavailable in order to cope with network glitches and other brief outages. To specifically wait for Neo4j to be available after starting, poll the Bolt or HTTP endpoint until it gives a successful response.

The details of how to poll depend upon:

- Whether the client uses HTTP or Bolt.
- Whether encryption or authentication are enabled.

It is important to include a timeout in case Neo4j fails to start. Normally, ten seconds should be sufficient, but database recovery or upgrade may take much longer depending on the size of the store. If the instance is part of a cluster then the endpoint will not be available until other instances have started up, and the cluster has formed.

Here is an example of polling (in Bash), using the HTTP endpoint with encryption and authentication disabled:

```
end=$((SECONDS+60))
while true; do
    [[ "200" = "$(curl --silent --write-out %{http_code} --output /dev/null http://localhost:7474)" ]] &&
    break
    [[ "${SECONDS}" -ge "${end}" ]] && exit 1
    sleep 1
done
```

Using Bolt instead of HTTP, the example could look like this.

```
end=$((SECONDS+60))
while true; do
    nc -w 2 localhost 7687 && break
    [[ "${SECONDS}" -ge "${end}" ]] && exit 1
    sleep 1
done
```

4.6. Usage Data Collector

This section describes the Neo4j Usage Data Collector.

The Neo4j Usage Data Collector, *UDC*, is a sub-system that gathers usage data, reporting it to the UDC-server at udc.neo4j.org. It is easy to disable, and does not collect any data that is confidential. For more information about what is being sent, see below.

The Neo4j team uses this information as a form of automatic, effortless feedback from the Neo4j community. We want to verify that we are doing the right thing by matching download statistics with usage statistics. After each release, we can see if there is a larger retention span of the server software.

The data collected is described in the section below. If any future versions of this system collect additional data, we will clearly announce those changes.

The Neo4j team is very concerned about your privacy. We do not disclose any personally identifiable information.

4.6.1. Technical Information

To gather good statistics about Neo4j usage, UDC collects this information:

- Kernel version: The build number, and if there are any modifications to the kernel.
- Store ID: A randomized globally unique ID created at the same time a database is created.
- Ping count: UDC holds an internal counter which is incremented for every ping, and reset for every restart of the kernel.
- Source: This is either "neo4j" or "maven". If you downloaded Neo4j from the Neo4j website, it's "neo4j", if you are using Maven to get Neo4j, it will be "maven".
- Java version: The referrer string shows which version of Java is being used.
- Registration ID: For registered server instances.
- Tags about the execution context (e.g. test, language, web-container, app-container, spring, ejb).
- Neo4j Edition (Community, Enterprise).
- A hash of the current cluster name (if any).
- Distribution information for Linux (rpm, dpkg, unknown).
- User-Agent header for tracking usage of REST client drivers
- MAC address to uniquely identify instances behind firewalls.
- The number of processors on the server.
- The amount of memory on the server.
- The JVM heap size.
- The number of nodes, relationships, labels and properties in the database.

After startup, UDC waits for ten minutes before sending the first ping. It does this for two reasons; first, we don't want the startup to be slower because of UDC, and secondly, we want to keep pings from automatic tests to a minimum. The ping to the UDC servers is done with a HTTP GET.

4.6.2. Disable the Usage Data Collector

UDC is disabled by setting `dbms.udc.enabled=false` in `neo4j.conf`. For more information on where to find the `neo4j.conf` file, see [File locations](#).

4.7. Configure Neo4j connectors

This section describes how to configure connectors for Neo4j.

Neo4j supports clients using either the Bolt binary protocol or HTTP/HTTPS. There are three different Neo4j connectors that are configured by default:

- a `bolt` connector
- a `http` connector
- a `https` connector

Table 5. Default connectors and their ports

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 connector, see also [SSL framework](#) for details on how to work with SSL certificates.

4.7.1. Additional options for Neo4j connectors

Some additional options are available for the connectors. They are summarized in the table below and subsequently explained in more detail.

Table 6. Configuration options for connectors

Option name	Default	Description
enabled	true	Allows the client connector to be enabled or disabled.
listen_address	127.0.0.1:<connector-default-port>	The address for incoming connections.
advertised_address	localhost:<connector-default-port>	The address that clients should use for this connector.
tls_level	OPTIONAL	Allows the connector to accept encrypted and/or unencrypted connections.

enabled

The `enabled` setting allows the client connector to be enabled or disabled. When disabled, Neo4j does not listen for incoming connections on the relevant port. For example, set the following to disable the HTTPS connector:

```
dbms.connector.https.enabled=false
```

It is not possible to disable the HTTP connector.



To prevent clients from connecting to HTTP, you should block the HTTP port with the firewall, or configure `listen_address` for the http connector to only listen on the loopback interface (127.0.0.1), thereby preventing connections from remote clients.

listen_address

The `listen_address` setting specifies how Neo4j listens for incoming connections. It consists of two parts; an IP address (e.g. 127.0.0.1 or 0.0.0.0) and a port number (e.g. 7687), and is expressed in the format `<ip-address>:<port-number>`.

Example 4. 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
```

advertised_address

The `advertised_address` 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>.

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
```

tls_level

The `tls_level` setting is only available for the `bolt` connector. It defines whether this Bolt connector will accept encrypted and/or unencrypted client connections. The values that `tls_level` setting accept are described in the table below:

Table 7. Available values to `tls_level`

Name	Description
REQUIRED	Only encrypted client connections will be accepted by this connector. All unencrypted connections will be rejected.
OPTIONAL	Either encrypted or unencrypted client connections are accepted by this connector.
DISABLED	Only unencrypted client connections are accepted by this connector. All encrypted connections will be rejected.

4.7.2. Additional options for Bolt connectors

See [Bolt thread pool configuration](#) to learn more about Bolt thread pooling and how to configure it on the connector level.

4.7.3. Defaults for addresses

The two configuration settings, `dbms.connectors.default_listen_address` and `dbms.connectors.default_advertised_address`, can be used to specify the IP address and address parts of `listen_address` and `advertised_address`, respectively. Setting a default value will apply to all the connectors, unless specifically configured for a certain connector.

Table 8. Defaults for addresses

Option name	Default	Description
<code>dbms.connectors.default_listen_address</code>	127.0.0.1	The default IP address for <code>listen_address</code> for all connectors.
<code>dbms.connectors.default_advertised_address</code>	localhost	The default address for <code>advertised_address</code> for all connectors.

default_listen_address

The listen address 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). If the IP address part of the `listen_address` is not specified, the interface is inherited from the shared setting `default_listen_address`.

Example 5. 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 `default_listen_address` setting, and then specifying the port number for the Bolt connector.

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

default_advertised_address

The advertised address consists of two parts; an address (fully qualified domain name, hostname, or IP address) and a port number (e.g. 7687). If the address part of the `advertised_address` is not specified, the interface is inherited from the shared setting `default_advertised_address`.

Example 6. 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 `default_advertised_address` setting, and then specifying the port number for the Bolt connector.

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

4.8. Dynamic settings

This section describes how to change your Neo4j configuration while Neo4j is running, and which settings can be changed.

This section contains the following:

- [Introduction](#)
- [Discover dynamic settings](#)
- [Update dynamic settings](#)
- [Dynamic settings reference](#)

4.8.1. Introduction

Neo4j Enterprise Edition supports changing some configuration settings at runtime, without restarting the service.



Changes to the configuration at runtime are not persisted. To avoid losing changes when restarting Neo4j make sure to update [neo4j.conf](#) as well.

4.8.2. Discover dynamic settings

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

Example 7. 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
```

4.8.3. Update dynamic settings

An [administrator](#) is able to change some configuration settings at runtime, without restarting the service.

Syntax:

```
CALL dbms.setConfigValue(setting, value)
```

Returns:

Nothing on success.

Exceptions:

Unknown or invalid setting name.

The setting is not dynamic and can not be changed at runtime.

Invalid setting value.

The following example shows how to dynamically enable query logging.

Example 8. Set a config value

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

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

Example 9. 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': must be 'true' or 'false'
```

To reset a config value to its default, pass an empty string as the *value* argument.

Example 10. Reset a config value to default

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

4.8.4. Dynamic settings reference

Dynamic settings reference

- [dbms.checkpoint.iops.limit](#): Limit the number of IOs the background checkpoint process will consume per second.
- [dbms.logs.query.allocation_logging_enabled](#): Log allocated bytes for the executed queries being logged.
- [dbms.logs.query.enabled](#): Log executed queries that take longer than the configured threshold, dbms.logs.query.threshold.
- [dbms.logs.query.page_logging_enabled](#): Log page hits and page faults for the executed queries being logged.
- [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 - provided query logging is enabled.
- [dbms.logs.query.time_logging_enabled](#): Log detailed time information for the executed queries being logged.
- [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.timeout](#): The maximum time interval of a transaction within which it should be completed.
- [dbms.tx_log.rotation.retention_policy](#): Make Neo4j keep the logical transaction logs for being able to backup the database.
- [dbms.tx_log.rotation.size](#): Specifies at which file size the logical log will auto-rotate.

4.9. Transaction logs

This section explains the retention and rotation policies for the Neo4j transaction logs, and how to configure them.

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. 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.

Log location

By default, transaction logs are kept in the same folder as the store they belong to. The directory where transaction logs are stored is configured by `dbms.directories.tx_log`. 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 two different ways:

- `dbms.tx_log.rotation.retention_policy=<true/false>`

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.

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>`

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 `dbms.checkpoint.interval.time` and `dbms.checkpoint.interval.tx`. 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.

Table 9. Types that can be used to control log retention

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 11. 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
```

Chapter 5. Clustering

This chapter describes the Neo4j Causal Clustering feature.

Installation and configuration

This part starts with the theoretical background and a discussion about architecture. It then proceeds to explicit configuration details and gives instructions on how to configure and operate the Causal Cluster.

- [Introduction](#) — An overview of the Causal Clustering architecture.
- [Lifecycle](#) — A walk-through of the life cycle of a cluster.
- [Create a new cluster](#) — How to create a new Causal Cluster.
- [Seed a cluster](#) — How to create a Causal Cluster from an existing database.
- [Intra-cluster encryption](#) — How to secure cluster communication.

For instructions on how you can upgrade your Neo4j Causal Cluster, see [Upgrading a Causal Cluster](#).

For instructions on setting up Causal Clustering when running Neo4j in a Docker container, see [Causal Clustering on Docker](#).

For a summary of the facilities that are available for monitoring a Neo4j Causal Cluster, see [Monitoring](#) (and specifically, [Monitoring a Causal Cluster](#)).

Multi-data center

This part is dedicated to advanced deployments and configuration options for multi-data center operations.

- [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.

Multi-clustering design and configuration

This part is dedicated to creating and configuring a multi-clustering deployment. It shows how to set up multiple distinct clusters containing independent graph stores, which are connected and exposed by a single discovery service.

- [Multi-clustering](#) — Overview of the multi-clustering section.
- [Introduction](#) — Introduction to multi-clustering functionality.
- [Configure a multi-cluster](#) — Configuration instructions for a multi-cluster.

For a summary of the facilities that are available for monitoring a Neo4j multi-cluster, see [Monitoring](#)

a multi-cluster.

Settings reference

- [Settings reference](#) — A summary of the most important Causal Cluster settings.

For a tutorial on setting up a test cluster locally on a single machine, see [Set up a local Causal Cluster](#).

5.1. Introduction

This section gives an introduction to Neo4j Causal Clustering.

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.

5.1.1. Operational view

From an operational point of view, it is useful to view the cluster as being composed from its two different roles: Core and Read Replica.

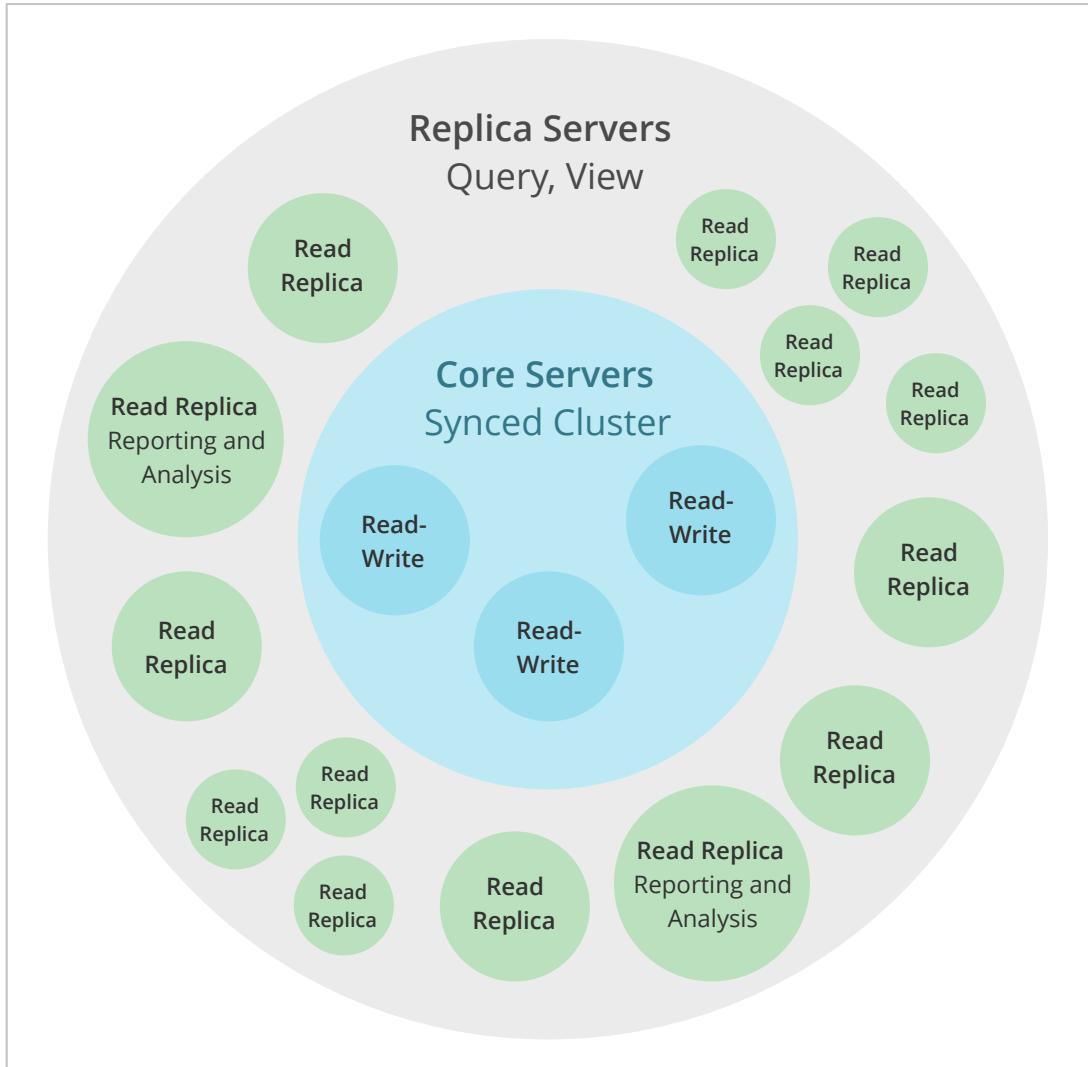


Figure 1. Causal Cluster Architecture

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

Core Servers' main responsibility is to safeguard data. The 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 simply 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 servers. However, that cluster will not be fault-tolerant; if one of the two servers fails, the remaining server will become read-only.

Note that should the Core Server cluster suffer enough failures that it can no longer process writes, it will become read-only to preserve safety.

Read Replicas

Read Replicas' main responsibility is to scale out graph workloads (Cypher queries, procedures, and so on). Read Replicas act like caches for the data that the Core Servers safeguard, but they are not simple key-value caches. In fact Read Replicas are fully-fledged Neo4j databases capable of fulfilling arbitrary (read-only) graph queries and procedures.

Read Replicas are asynchronously replicated from Core Servers via transaction log shipping. Periodically (usually in the ms range) a Read Replica will poll a Core Server for any new transactions that it has processed since the last poll, and the Core Server will ship those transactions to the Read Replica. 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.

Unlike Core Servers however, Read Replicas do not participate in decision making about cluster topology. Read Replicas should be typically 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.

5.1.2. 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. Client applications never see stale data and (logically) interact with the database as if it was a single server. Consequently client applications enjoy read-your-own-writes semantics making interaction with even large clusters simple and predictable.

Causal consistency makes it easy 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.

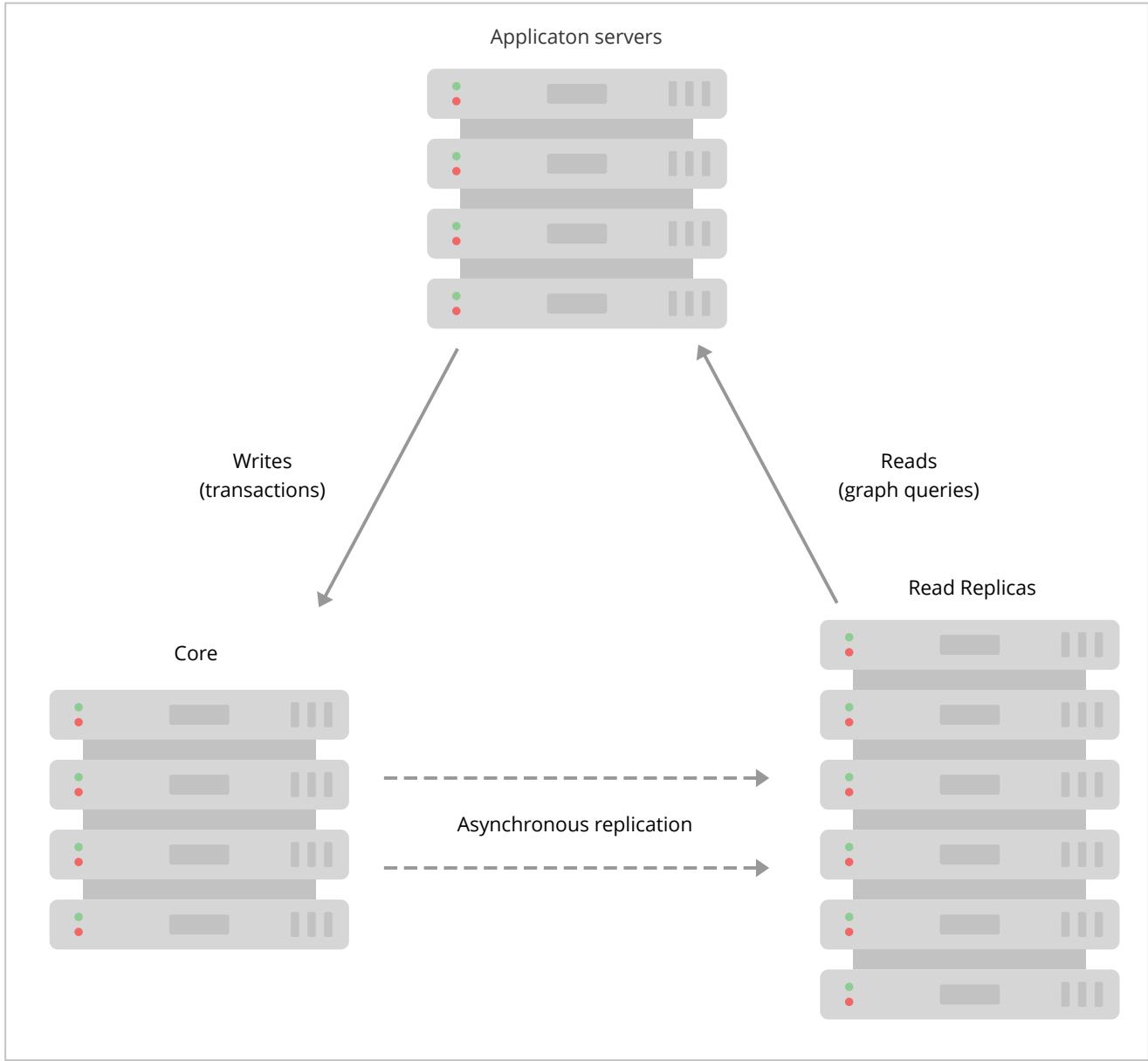


Figure 2. Causal Cluster setup with causal consistency via Neo4j drivers

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.

5.1.3. 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 in the cluster 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.

5.2. Lifecycle

This section describes the lifecycle of a Neo4j Causal Cluster.

[Introduction](#) provided an overview of a Causal Cluster. 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.

5.2.1. 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.

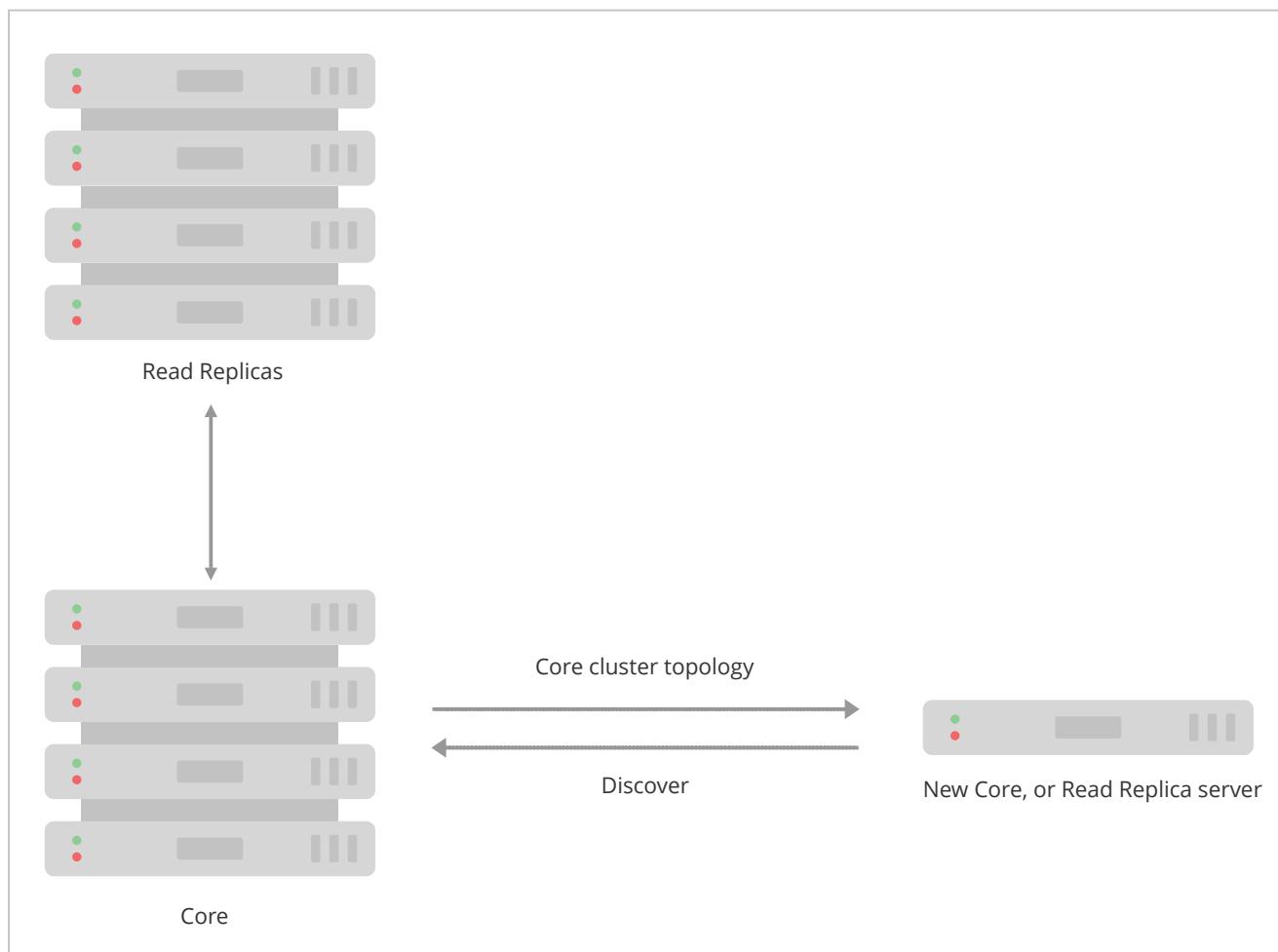


Figure 3. Causal Cluster discovery protocol: Core-to-Core or Read Replica-to-Core only.

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](#).

5.2.2. 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.



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](#) (<https://ramcloud.stanford.edu/~ongaro/thesis.pdf>) 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 log 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.

When an instance establishes a connection to any other instance, it determines the current state of the cluster and ensures that it is eligible to join. To be eligible the Neo4j instance must host the same database store as other members of the cluster (although it is allowed to be in an older, outdated, state), or be a new deployment without a database store.

5.2.3. 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.

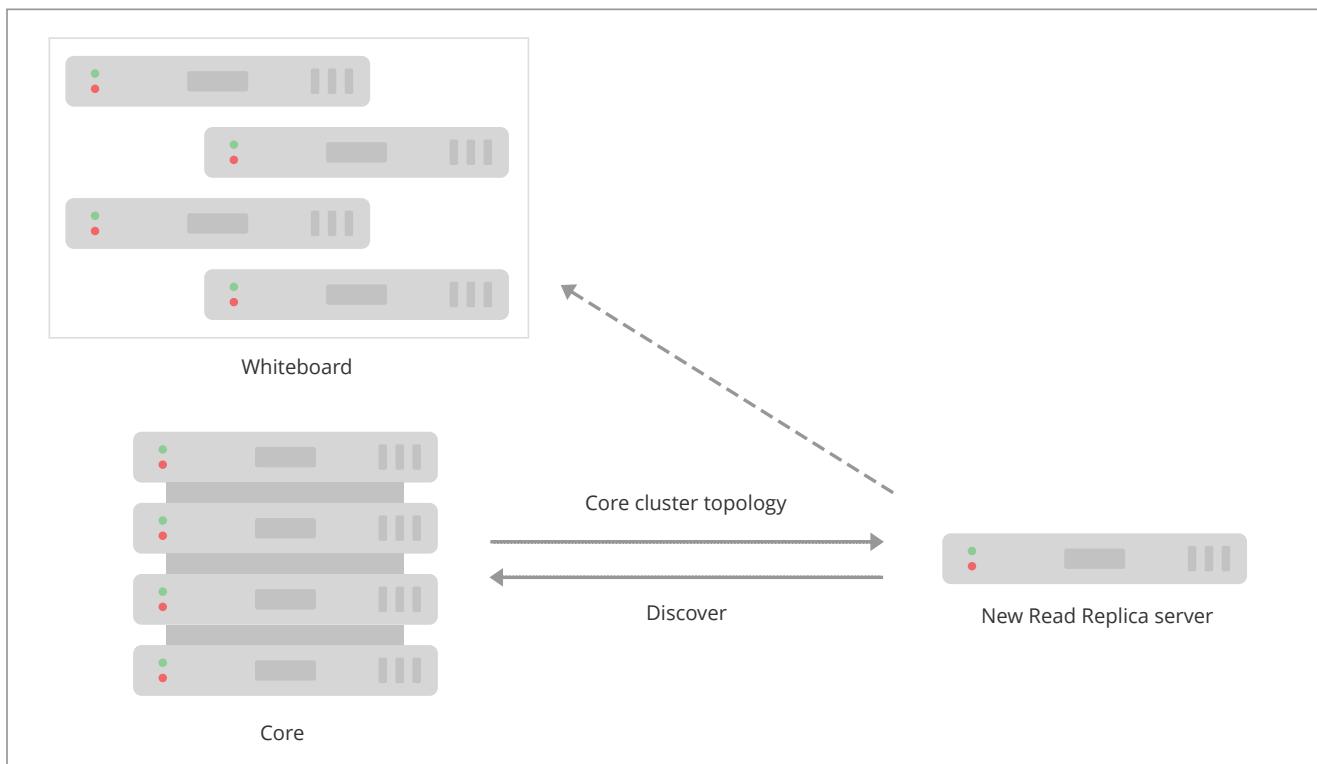


Figure 4. All Read Replicas registered with shared whiteboard.

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.

5.2.4. 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 easy 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.



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.

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, then they can instigate a leadership election by entering the *Candidate* state. In Neo4j Core Servers this happens at ms timescale, around 500ms by default.

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.

5.2.5. 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.

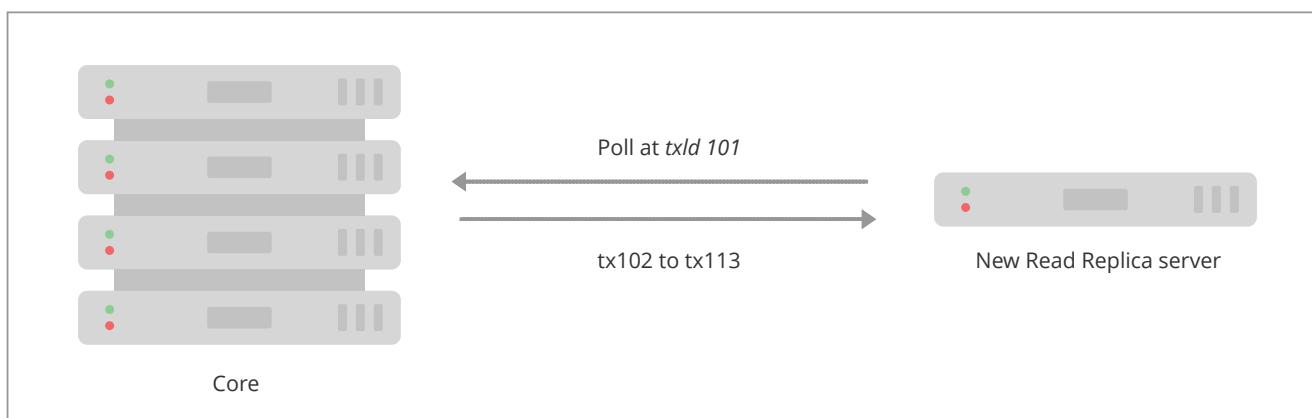


Figure 5. Transactions shipped from Core to Read Replica.

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.



Neo4j transaction IDs are strictly monotonic integer values (they always increase). This makes it simple 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 a 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.

5.2.6. 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.

5.2.7. Core shutdown

A clean Core Server shutdown, like Core Server booting, is handled via the Raft protocol. When a Core Server is shut down, it appends a membership entry to the Raft log which is then replicated around the Core Servers. Once a majority of Core Servers have committed that membership entry the leaver has logically left the cluster and can safely shut down. All remaining instances accept that the cluster has grown smaller, and is therefore less fault tolerant. If the leaver happened to be playing the Leader role at the point of leaving, it will be transitioned to another Core Server after a brief election.

An unclean shutdown does not directly inform the cluster that a Core Server has left. Instead the Core cluster size remains the same for purposes of computing majorities for commits. Thus an unclean shutdown in a cluster of 5 Core Servers now requires 3/4 members to agree to commit which is a tighter margin than 3/5 before the unclean shutdown.



Of course when Core Servers fail, operators or monitoring scripts can be alerted so that they can intervene in the cluster if necessary.

If the leaver was playing the Leader role, there will be a brief election to produce a new Leader. Once the new Leader is established, the Core cluster continues albeit with less redundancy. However even with this failure, a Core cluster of 5 servers reduced to 4 can still tolerate one more fault before becoming read-only.

5.3. Create a new cluster

This section describes how to deploy a new Neo4j Causal Cluster.

In this section we will learn how to set up a brand new Causal Cluster consisting of three Core instances; the minimum number of servers needed for a fault-tolerant Core cluster (for a discussion on the number of servers required for a Causal Cluster, see [Core Servers](#)). We then learn how to add more Core Servers as well as Read Replicas. From this basic pattern, we can extend what we have learned here to create any sized cluster.

The following subjects are described:

- Configure a Core-only cluster
- Add a Core Server to an existing cluster
- Add a Read Replica to an existing cluster
- Remove a Core from a cluster
- Bias cluster leadership with follower-only instances
- Initial discovery of cluster members
- Initial discovery of cluster members with Kubernetes
- Store copy

For a description of the clustering architecture and cluster concepts encountered here, please refer to [Introduction to Causal Clustering](#). We will not describe how to import data from an existing Neo4j instance; for help on using an existing Neo4j database to seed a new Causal Cluster, please see [Seed a Causal Cluster](#).

If you want to try to set up a Causal Cluster on your local machine, refer to [Set up a local Causal Cluster](#).

5.3.1. Configure a Core-only cluster

The instructions in this section assume that you have already downloaded the appropriate version of Neo4j Enterprise Edition from [the Neo4j download site](#) (<https://neo4j.com/download/other-releases/#releases>), and have installed it on the servers that will make up the cluster.

When deploying a new Causal Cluster, the following configuration settings are important to consider for basic cluster operation.

The settings below are located in [*neo4j.conf*](#) under the header "Network connector configuration".

`dbms.connectors.default_listen_address`

The address or network interface this machine uses to listen for incoming messages.
Uncommenting this line sets this value to `0.0.0.0` which allows Neo4j to bind to any and all network interfaces. Uncomment the line `dbms.connectors.default_listen_address=0.0.0.0`

`dbms.connectors.default_advertised_address`

The address that other machines are told to connect to. In the typical case, this should be set to the public IP address of this server. For example, if the IP address is 33.44.55.66, this setting should be:
`dbms.connectors.default_advertised_address=33.44.55.66`

The settings below are located in [*neo4j.conf*](#) under the header "Causal Clustering Configuration".

`dbms.mode`

The operating mode of a single database instance. For Causal Clustering, there are two possible modes: `CORE` or `READ_REPLICA`. On three of our instances, we will use the `CORE` mode. Uncomment the line: `#dbms.mode=CORE`.

`causal_clustering.minimum_core_cluster_size_atFormation`

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. For example, `causal_clustering.minimum_core_cluster_size_atFormation=3` will specify that the cluster requires three Core members in order to start.

`causal_clustering.minimum_core_cluster_size_atRuntime`

The minimum number of Core instances which will exist in the consensus group. For example, `causal_clustering.minimum_core_cluster_size_atRuntime=3` will specify that the cluster will not

dynamically vote out instances when the consensus group reaches the size of three, even if they are lost.

`causal_clustering.initial_discovery_members`

The network addresses of an initial set of Core cluster members that are available to bootstrap this Core or Read Replica instance. How this value is interpreted is determined by `causal_clustering.discovery_type`. When discovery type is set to `LIST`, which is its default value, the initial discovery members are given as a comma-separated list of address/port pairs, for example `core1:5000,core2:5000,core3:5000`. The default port for the discovery service is `:5000`. You should include the address of the local machine in this setting.

`causal_clustering.discovery_type`

The mechanism to use along with the value provided for `config_causal_clustering.initial_discovery_members` to determine the addresses of other members of the cluster on startup. In the simplest case, `config_causal_clustering.discovery_type` is set to `LIST` and each of the specified network addresses resolves to a working Neo4j Core Server instance. This value (`LIST`) requires you to know either the hostnames or the IP addresses of the Core instances in the cluster. If you do not know the addresses of the other Core instances, alternative discovery mechanisms are described in [Initial discovery of cluster members with DNS](#).

Apply these settings to the configuration file on each instance. The values can be the same for each.

Start the Neo4j servers as usual. Note that the startup order does not matter.

Example 12. Start the Core-only cluster

```
server-1$ ./bin/neo4j start
```

```
server-2$ ./bin/neo4j start
```

```
server-3$ ./bin/neo4j start
```

Startup Time



If you want to follow along with the startup of a server you can follow the messages in `neo4j.log`. While an instance is joining the cluster, the server may appear unavailable.

Now you can access the three servers and check their status. Open the locations below in a web browser and issue the following query: `CALL dbms.cluster.overview()`. This will show you the status of the cluster and information about each member of the cluster.

- <http://server-1:7474/>
- <http://server-2:7474/>
- <http://server-3:7474/>

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

5.3.2. Add a Core Server to an existing cluster

Adding instances to the Core cluster is simply a matter of starting a new database server with the appropriate configuration as described in [Configure a Core-only cluster](#). Following those instructions, we need to change `neo4j.conf` to reflect the new Core Server's desired configuration like so:

- Set `dbms.mode=CORE`.
- Set `causal_clustering.initial_discovery_members` to be identical to the corresponding parameter on the existing Core Servers.

Once we've done that, start the server and the new server will integrate itself with the existing cluster. 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. When the server has copied over the graph data from its peers it will become available.

5.3.3. 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 simply need to know the addresses of some of the Core Servers which they can bind to in order to run the discovery protocol (see: [Discovery protocol](#) for details). Once it has completed the initial discovery the Read Replica becomes aware of the available Core Servers and can choose an appropriate one from which to catch up (see: [Catchup protocol](#) for how that happens).

In the `neo4j.conf` file in the section "Causal Clustering Configuration", the following settings need to be changed:

- Set operating mode to Read Replica: `dbms.mode=READ_REPLICA`.
- Set the parameter `causal_clustering.initial_discovery_members` to be identical to the corresponding parameter on the Core Servers.

5.3.4. Remove a Core from a cluster

A Core Server can be downgraded to a standalone instance, using the `neo4j-admin unbind` command.

Once a server has been unbound from a cluster, the store files are equivalent to a Neo4j standalone instance. From this point those files could be used to run a standalone instance by restarting it in `SINGLE` mode.

The on-disk state of Core Server instances is different to that of standalone server instances. It is important to understand that once an instance unbinds from a cluster, it cannot be re-integrated with that cluster. This is because both the cluster and the single instance are now separate databases with different and irreconcilable writes having been applied to them. Technically the cluster will have written entries to its Raft log, whilst the standalone instance will have written directly to the transaction log (since there is no Raft log in a standalone instance).

 If we try to reinsert the standalone instance 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.

5.3.5. Bias cluster leadership with follower-only instances

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* that replicate the leader's state. In Neo4j terms this means writes to the database are ordered by the Core instance currently playing the leader role.

In some situations, operators might want to actively prevent some instances from taking on the leader role. For example in a multi-data center scenario, we might want to ensure that the leader remains in a favored geographic location for performance or governance reasons. In Neo4j Causal Clustering we

can configure instances to *refuse to become leader*, which is equivalent to always remaining a follower. This is done by configuring the setting `causal_clustering.refuse_to_be_leader`. It is not generally advisable to use this option, since the first priority in the cluster is to maximize availability. The highest availability stems from having any healthy instance take leadership of the cluster on pathological failure.



Despite superficial similarities, a non-leader capable Core instance is not the same as a Read Replica. Read Replicas do not participate in transaction processing, nor are they permitted to be involved in cluster topology management.

Conversely, a follower-only Core instance will still process transactions and cluster membership requests as per Raft to ensure consistency and safety.

5.3.6. Initial discovery of cluster members

When a Neo4j Core Server starts up it needs to contact other Core members, to form a new cluster or to join an existing cluster. To contact the other members, the Core Server needs to know their addresses. The address consist of the hostname or IP address, and the port on which the discovery service is listening.

When a cluster member discovers another, they share what they know of the rest of the cluster. It is therefore not necessary for each instance to have the address of every other instance in the cluster. However, if the initial discovery members form two or more disjoint sets, then a cluster will not be able to form.

When the addresses of the other cluster members are known they can be listed explicitly. In this case we can use the default `causal_clustering.discovery_type=LIST` and hard code the addresses in the configuration of each machine, for example

```
causal_clustering.initial_discovery_members=10.0.0.1:5000,10.0.0.2:5000,10.0.0.3:5000.
```

Explicitly listing the addresses of Core members for discovery is convenient, but has limitations.

- First, 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.
- Second, under some circumstances the addresses are not known when configuring the cluster. This can be the case for example when using container orchestration to deploy a Causal Cluster.

For cases where it is not practical or possible to explicitly list the addresses of cluster members to discover, additional discovery mechanisms are provided using DNS.

5.3.7. Initial discovery of cluster members with DNS

There are two DNS based mechanisms that can be used to get the addresses of Core cluster members for discovery.

- With `config_causal_clustering.discovery_type=DNS`, the initial discovery members will be resolved from DNS A records to find the IP addresses to contact.
- With `config_causal_clustering.discovery_type=SRV`, the initial discovery members will be resolved from DNS SRV records to find the IP addresses/hostnames and discovery service ports to contact.

When using `discovery_type=DNS` the `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=cluster1.neo4j.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 member. The configured Core Server will use all the IP addresses from the A records to join or form a cluster.

If `discovery_type=DNS` is used, the discovery port on all Cores must be the same. If this is not possible, consider using the `discovery_type=SRV` configuration.

When using `discovery_type=SRV`, the `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=cluster1.neo4j.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.

For both `DNS` and `SRV` configurations, the 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.

5.3.8. Initial discovery of cluster members with Kubernetes

If a Causal Cluster is running in [Kubernetes](https://kubernetes.io/) (<https://kubernetes.io/>) and each Core Server is running as a Kubernetes service, then the addresses of Core cluster members can be obtained using the [List Service](https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.11/#list-service-v1-core) (<https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.11/#list-service-v1-core>) API.

This mechanism can be used by setting `config_causal_clustering.discovery_type=K8S`. Unlike `LIST` or the `DNS` mechanisms, `causal_clustering.initial_discovery_members` is not used or required.

Instead, `causal_clustering.kubernetes.label_selector` must be set to a [label selector](https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/#label-selectors) (<https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/#label-selectors>) for the Causal Cluster services, and `causal_clustering.kubernetes.service_port_name` must be set to the name of the [service port](https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.11/#serviceport-v1-core) (<https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.11/#serviceport-v1-core>) used in the Kubernetes service definition for the Core's discovery port.

Please note that:



- 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](https://kubernetes.io/docs/reference/access-authn-authz/rbac/) (<https://kubernetes.io/docs/reference/access-authn-authz/rbac/>) or [ABAC authorization](https://kubernetes.io/docs/reference/access-authn-authz/abac/) (<https://kubernetes.io/docs/reference/access-authn-authz/abac/>).
- 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.`

Like DNS based methods, the Kubernetes record lookup is only performed at start up.

5.3.9. Store copy

Causal Clustering uses a robust and configurable store copy protocol. When a store copy is started it will first send a prepare request to the specified instance. If the prepare request is successful the client will send file and index requests, one request per file or index, to provided upstream members in the cluster. The retry logic per request can be modified through `causal_clustering.store_copy_max_retry_time_per_request`. If a request fails and that maximum retry time is met 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 catchup request. Bear in mind that this setting is for all requests from the catchup client, including pulling of transactions.

There are three scenarios that will start a store copy. The upstream selection strategy is different for each scenario.

Backup

Upstream strategy is set to a fixed member by the `neo4j-admin backup` command. All requests will go to the specified member.

Seeding a new member with empty store

Will use configured upstream strategy for that instance.

Instance falling too far behind

Will use configured upstream strategy for that instance.

The upstream strategy differs for Cores and Read Replicas. Cores will always send the prepare request to the leader to get the most up-to-date information of the store. The strategy for the file and index requests for Cores is to vary every other request to random Read Replica and every other to random Core member. Read Replicas' strategy is the same as for pulling transactions. The default is to pull from a random Core member.

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

5.4. Seed a cluster

This section describes how to seed a new Neo4j Causal Cluster with existing data.

In [Create a new cluster](#) we learned how to create a cluster with an empty store. However, regardless if 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.

In this section we will describe the following methods for seeding a cluster:

- [Seed from an online backup](#)
- [Seed from an offline backup](#)
- [Seed using the import tool](#)

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 which format our data is in:

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



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

5.4.1. Seed from an online backup

For this example, it is assumed that we already have a healthy backup of an existing Neo4j database as a result of an online backup from a running Neo4j instance (for details on online backups, please refer to [Backup](#)). This could be a standalone Neo4j instance, a Neo4j Highly Available cluster, or a running instance of another Neo4j Causal Cluster. 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 operability 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. If you start the cluster in order to test that it works, you have to subsequently stop it and perform `neo4j-admin unbind` on each of the instances.

2. Seed the cluster.

Use the `restore` command of `neo4j-admin` to restore the seeding store from the backed-up database on all the Core instances in the cluster. Since all instances are seeded with the store, the cluster will be fully available right away once the instances are started.

Example 13. Restore the backup to seed all Core members.

This example assumes that the database name is the default `graph.db` and that we have a valid backup residing under the `seed-dir` directory. If you have a different setup, change the command line arguments accordingly.

```
neo4j-01$ ./bin/neo4j-admin restore --from=seed-dir --database=graph.db
```

```
neo4j-02$ ./bin/neo4j-admin restore --from=seed-dir --database=graph.db
```

```
neo4j-03$ ./bin/neo4j-admin restore --from=seed-dir --database=graph.db
```

3. Start the cluster.

At this point, all of the instances in the Core cluster have the store that contains our graph data. Between them the Core Servers have everything necessary to form a cluster. We are ready to start all instances. The cluster will form and data will be replicated between the instances.

Example 14. Start each of the Core instances.

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

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

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

5.4.2. Seed from an offline backup

There are cases where we may want to seed a database in an offline fashion, for example if we are upgrading from Neo4j Community to Enterprise, or if we choose to transplant a database from one Enterprise site to another. To handle offline backups, we use the `dump` and `load` commands of `neo4j-admin`. For more detailed instructions on these, please refer to [Dump and load databases](#).

The overall process for seeding from an offline backup is the same as for an online backup, with the difference that the backup must be restored onto all the Core members.

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. If you start the cluster in order to test that it works, you have to subsequently stop it and perform `neo4j-admin unbind` on each of the instances.

2. Seed the cluster.

Seed the cluster by loading the dump file into each of the newly created Core member using the `load` command of `neo4j-admin`.

Example 15. Load the database into each Cluster member

In this example we assume that we have an offline backup of your Neo4j database as a result of using the `dump` command of `neo4j-admin`. The database name is the default `graph.db` and we have a dump file called `graph.dump` in the `seed-dir` directory. If you have a different setup, change the command line arguments accordingly.

```
neo4j-01$ ./bin/neo4j-admin load --from=seed-dir/graph.dump --database=graph.db
```

```
neo4j-02$ ./bin/neo4j-admin load --from=seed-dir/graph.dump --database=graph.db
```

```
neo4j-03$ ./bin/neo4j-admin load --from=seed-dir/graph.dump --database=graph.db
```

3. Start the cluster.

At this point all the instances of the Core cluster have the store that contains our graph data. We are ready to start all instances the same way that is illustrated in [Start each of the Core instances..](#)

5.4.3. Seed using the import tool

In the case that we wish to create a cluster based on imported data, we follow a procedure that is very similar to that of seeding from an offline backup.

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. If you start the cluster in order to test that it works, you have to subsequently stop it and perform `neo4j-admin unbind` on each of the instances.

2. Seed the cluster.

Seed the cluster by loading imported data into each of the newly created Core members using the [import tool](#).

3. Start the cluster.

At this point all the instances of the Core cluster have the store that contains our graph data. We are ready to start all instances the same way that is illustrated in [Start each of the Core instances..](#)



Using copy-and-paste to move the internal `data` directory, in order to transfer and seed databases is not supported. If you have an existing Neo4j database whose data you wish to use for a new cluster, follow one of the processes described in this section.

5.5. Intra-cluster encryption

This chapter describes how to secure the cluster communication between server instances.

5.5.1. 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.

5.5.2. Example deployment

The following steps will create an example deployment, and each step is expanded in further detail below.

- Generate and install [cryptographic objects](#)
- Create an SSL policy
- 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.

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.

Neo4j is able to generate self-signed certificates but a deployment using these should generally be regarded as a special case. It can make sense in some circumstances and even a mutual authenticated setup can be created by sharing the self-generated certificates among instances.

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 16. Generate and install cryptographic objects

In this example we assume a plan to name the SSL policy `cluster`, and 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
```

Create an SSL policy

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 10. Example settings

Setting suffix	Value	Comment
<code>client_auth</code>	<code>REQUIRE</code>	Setting this to <code>REQUIRE</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 (AES) 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 17. Create an 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 wish to create a cluster SSL policy called `cluster`. We do this by defining the following parameter.

```
dbms.ssl.policy.cluster.base_directory=certificates/cluster
```

Since our policy is named `cluster` the corresponding settings will be available to configure under the `dbms.ssl.policy.cluster.*` group. We add the following content to our `neo4j.conf` file:

```
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
```

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.

Configure Causal Clustering with the SSL policy

There is no default SSL policy for Causal Clusters. This means that by default, cluster communication is unencrypted. To configure a Causal Cluster to encrypt its intra-cluster communication, set `causal_clustering.ssl_policy` to the name of a valid SSL policy.

Example 18. Configure Causal Clustering with the SSL policy

In this example we assume that the tasks in the previous two examples have been performed. We now configure our cluster to use the SSL policy that we have named `cluster`.

```
causal_clustering.ssl_policy=cluster
```

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.

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 19. 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.

5.6. 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.

The following topics are detailed:

- [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.

Licensing for multi-data center operations

Most of Neo4j is open source software and can be used legally within the constraints of the open source licenses under which it is released. However, multi-data center operation is intended for very demanding users of Neo4j who typically operate under a commercial database license. On a commercial basis, 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.

5.6.1. 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 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 20. Homogeneous three data center deployment

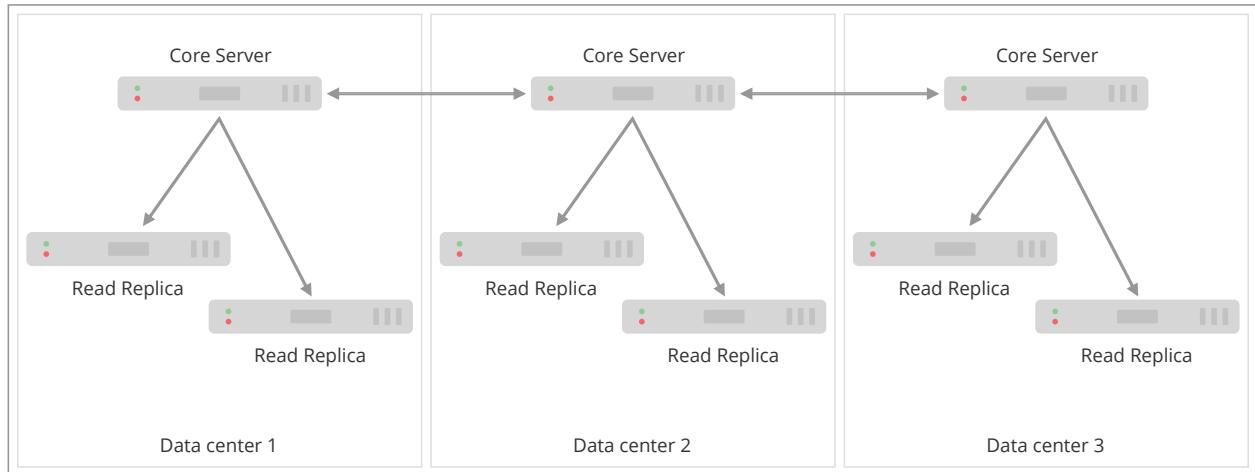


Figure 6. Homogeneous deployment across three data centers with one Core instance in each

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.

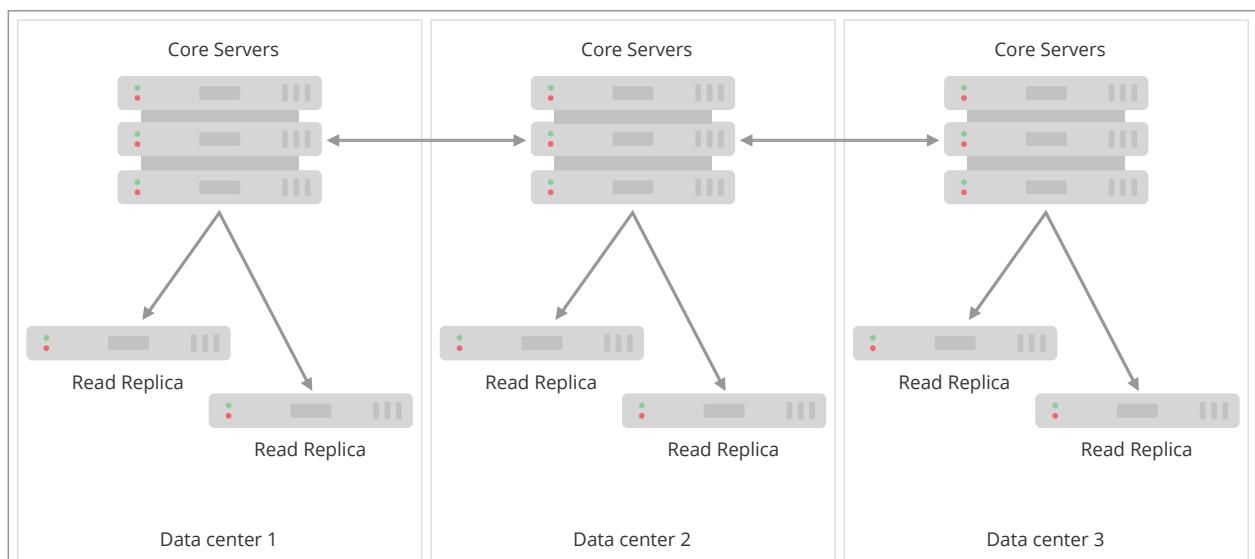


Figure 7. Homogeneous deployment across three data centers with three Core instances in each

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 21. Homogeneous two data center deployment

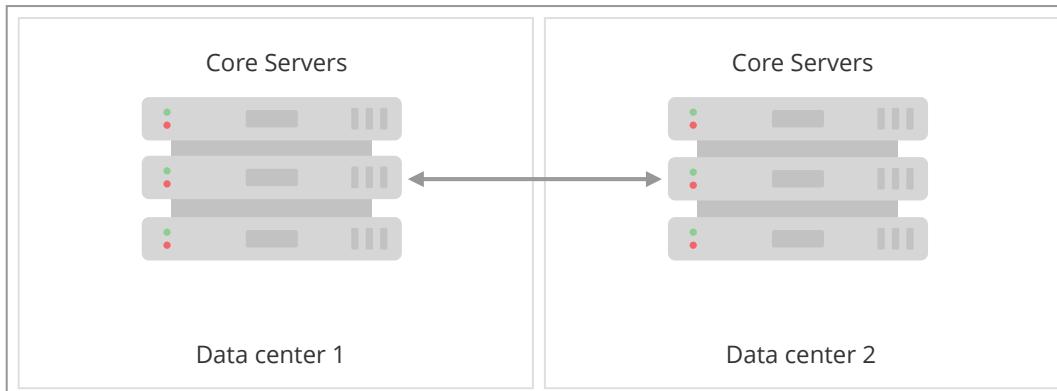


Figure 8. Homogeneous deployment across two data centers

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 22. 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.

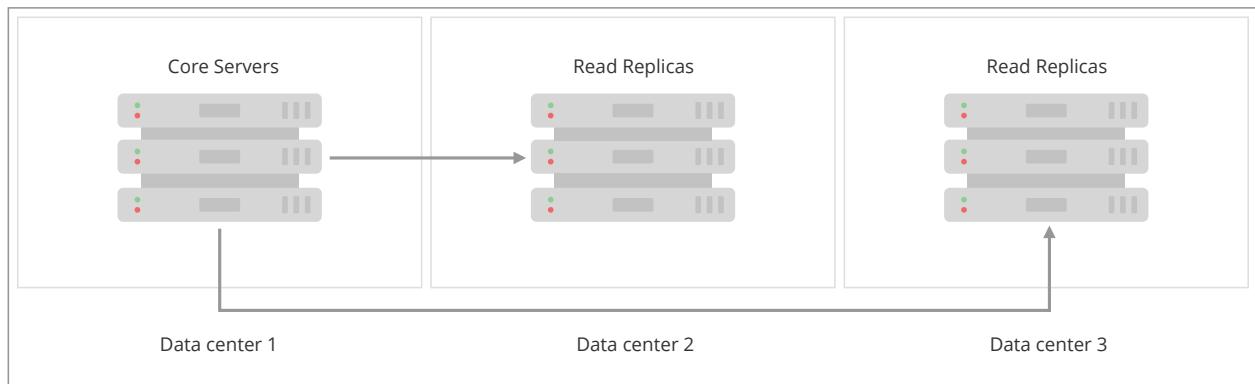


Figure 9. Heterogeneous deployment separating Read Replicas from the Core cluster

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 as [follower-only](#). 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.

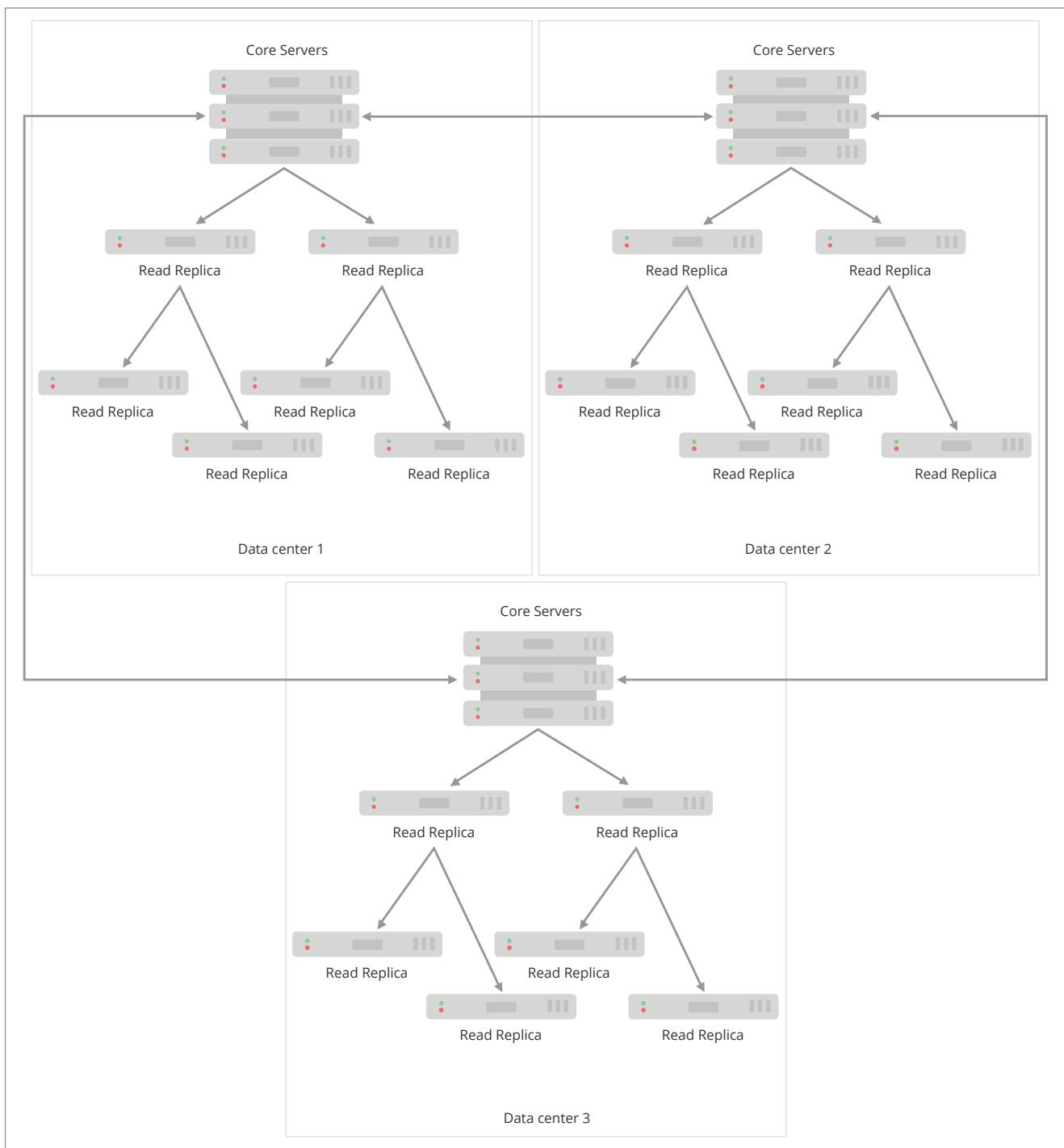


Figure 10. Fan out from Core Servers for scale at log cost

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.

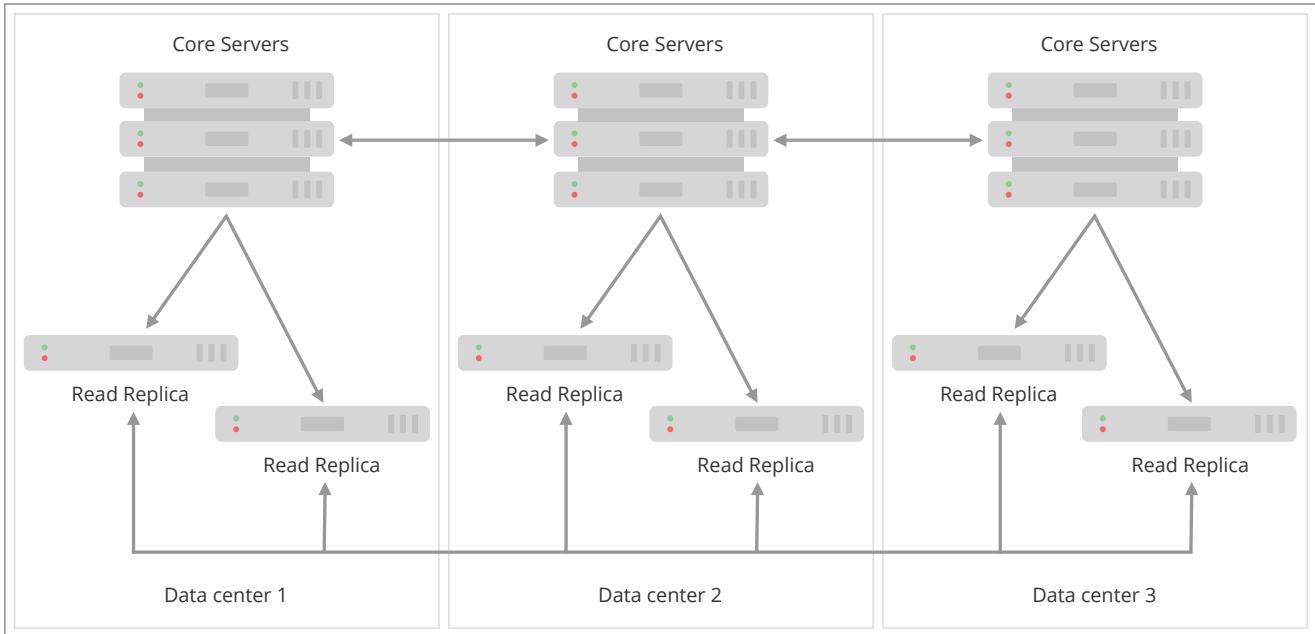


Figure 11. Peer-to-peer Read Replica catchup

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.

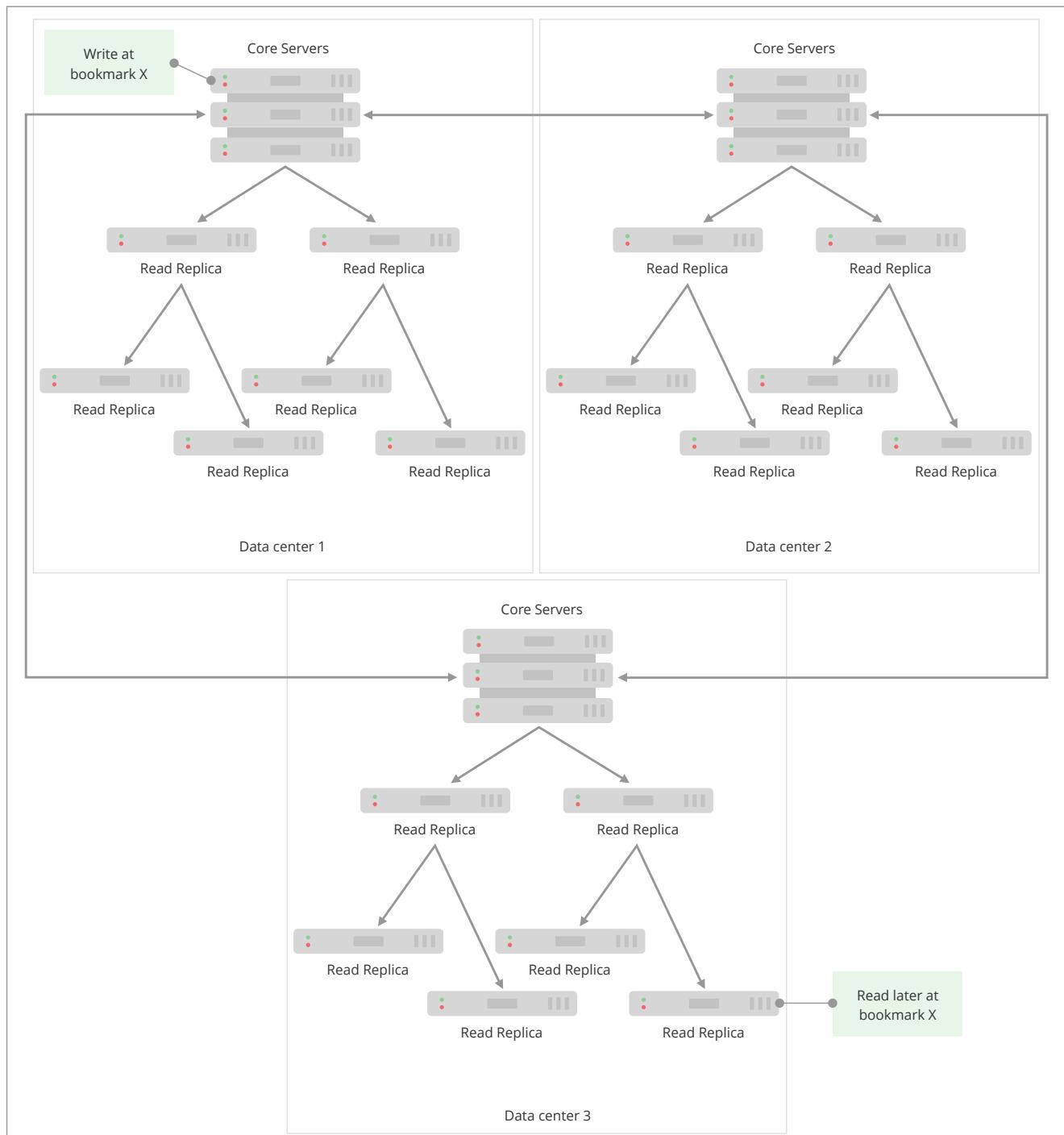


Figure 12. Each tier in the Read Replicas is further behind the source of truth, but offers greater scale-out

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.

5.6.2. 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.



Enabling multi-data center operation

The multi-data center functionality is separately licensed and must be specifically enabled. See [Licensing for multi-data center operations](#) for details.

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](#).

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 23. 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 24. 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.

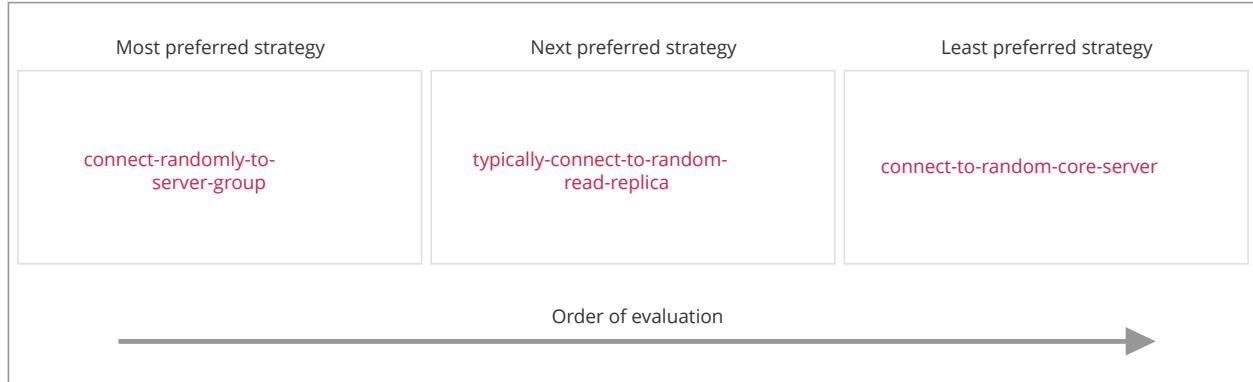


Figure 13. The first satisfactory response from a strategy will be used.

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 25. 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 14. 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 26. 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 simply copy it into the `plugins` directory and restart the instance.

Favoring data centers

In [Bias cluster leadership with follower-only instances](#) we saw how we can bias the leadership credentials of instances in a cluster. Generally speaking this is a rare occurrence in a homogenous cluster inside a single data center.

In a multi-DC scenario, while it remains a rare occurrence, it does allow expert operators to bias which data centers are used to host Raft leaders (and thus where writes are directed). We apply `causal_clustering.refuse_to_be_leader=true` in those data centers where we do not want leaders to

materialize. In doing so we implicitly prefer the instances where we have **not** applied that setting.

This may be useful when planning for highly distributed multi-data center deployments. However this must be very carefully considered because in failure scenarios it limits the availability of the cluster. It is advisable to engage Neo4j professional services to help design a suitably resilient topology.

5.6.3. 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.

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 15. 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 27. 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 28. 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 29. 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.

5.6.4. 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.

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.

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](#).

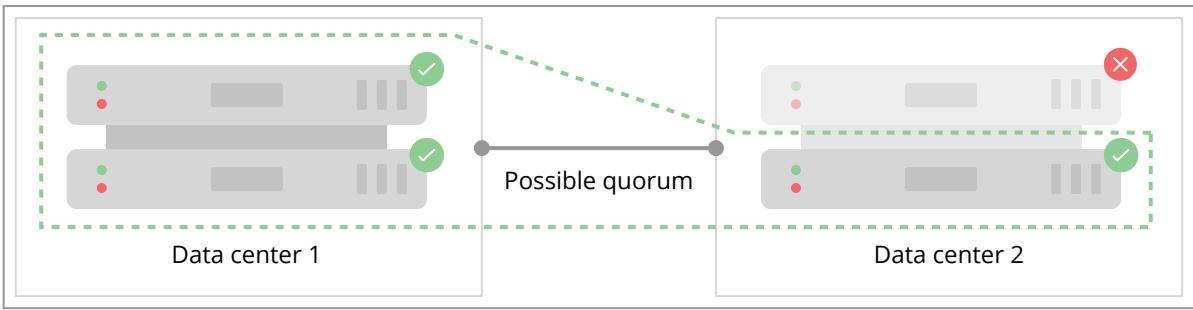


Figure 16. Two Data Center Deployment with Four Core Instances

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.

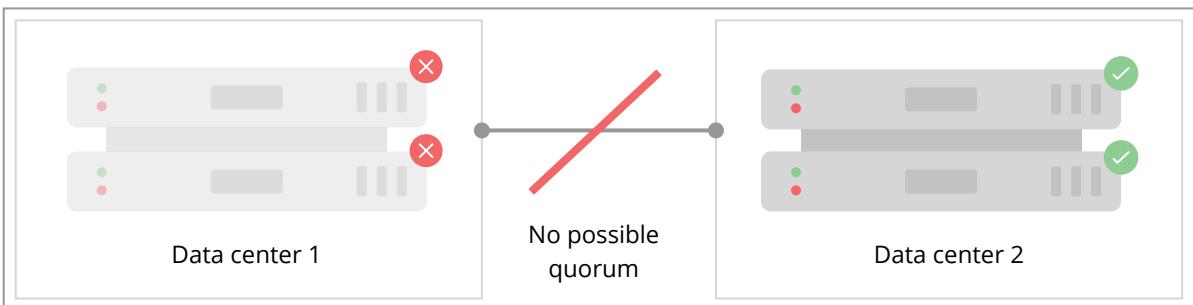


Figure 17. Data Center Loss Requires Guided Recovery

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 recover 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. 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 is as follows:

1. Ensure that a catastrophe has occurred and that we have access to the surviving members of the cluster in the surviving data center. Then for each instance:
 2. Stop the instance with `bin/neo4j stop` or shut down the service.
 3. 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.
 4. Ensure that the settings `causal_clustering.minimum_core_cluster_size_atFormation=2` and `causal_clustering.minimum_core_cluster_size_atRuntime=2` (assuming two surviving instances) appear in the `neo4j.conf` settings file.
 5. Unbind the instance using `neo4j-admin unbind`.
 6. Start the instance with `bin/neo4j start` or start the `neo4j` service.

Once this 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 [Create a new cluster](#) for details of how to configure instances to join the cluster from scratch.

5.7. Multi-clustering

This section introduces the multi-clustering functionality in Neo4j.

Multi-clustering is a limited form of multitenancy and database discovery available in Neo4j. The functionality builds upon and extends [Causal Clustering](#).

This section includes:

- [Introduction](#) — Introduction to multi-clustering functionality.
- [Configure a multi-cluster](#) — Configuration instructions for a multi-cluster.

5.7.1. Introduction

This section provides an introduction to Neo4j multi-clustering—running multiple co-ordinated Causal Clusters with a shared discovery service.

[Multitenancy](#) (<https://en.wikipedia.org/wiki/Multitenancy>), in the context of databases, refers to the concept of having a single database management system manage multiple separate databases. These databases may be looked up and queried by multiple separate clients.

With Neo4j, this is achieved by organizing multiple Neo4j Causal Clusters, each managing its own database, into a multi-cluster with a shared discovery service. Each member of the multi-cluster is configured with a database name. Members configured to manage the same database are grouped by the multi-cluster's single discovery service and form multiple smaller [Raft consensus groups](#) — that is, individual clusters.

This is different from the default behaviour of Neo4j Causal Clustering, where the discovery service forms a Raft consensus group using all Core instances in a cluster. In fact, the default clustering behaviour is a special case of the multi-clustering behaviour, where every instance has the same (default) database name.

Once formed, each constituent cluster is almost entirely distinct. They may be queried separately, and will contain different store contents as a result. Furthermore, the fault tolerance of one cluster does not depend on the fault tolerance of another.

As an example, consider a Neo4j Causal Cluster containing fifteen instances, nine of which are [Core](#) instances, and six are [Read Replicas](#). Sets of three cores and two Read Replicas are configured with the database names `foo`, `bar` and `baz` respectively. The figure below reflects this deployment.

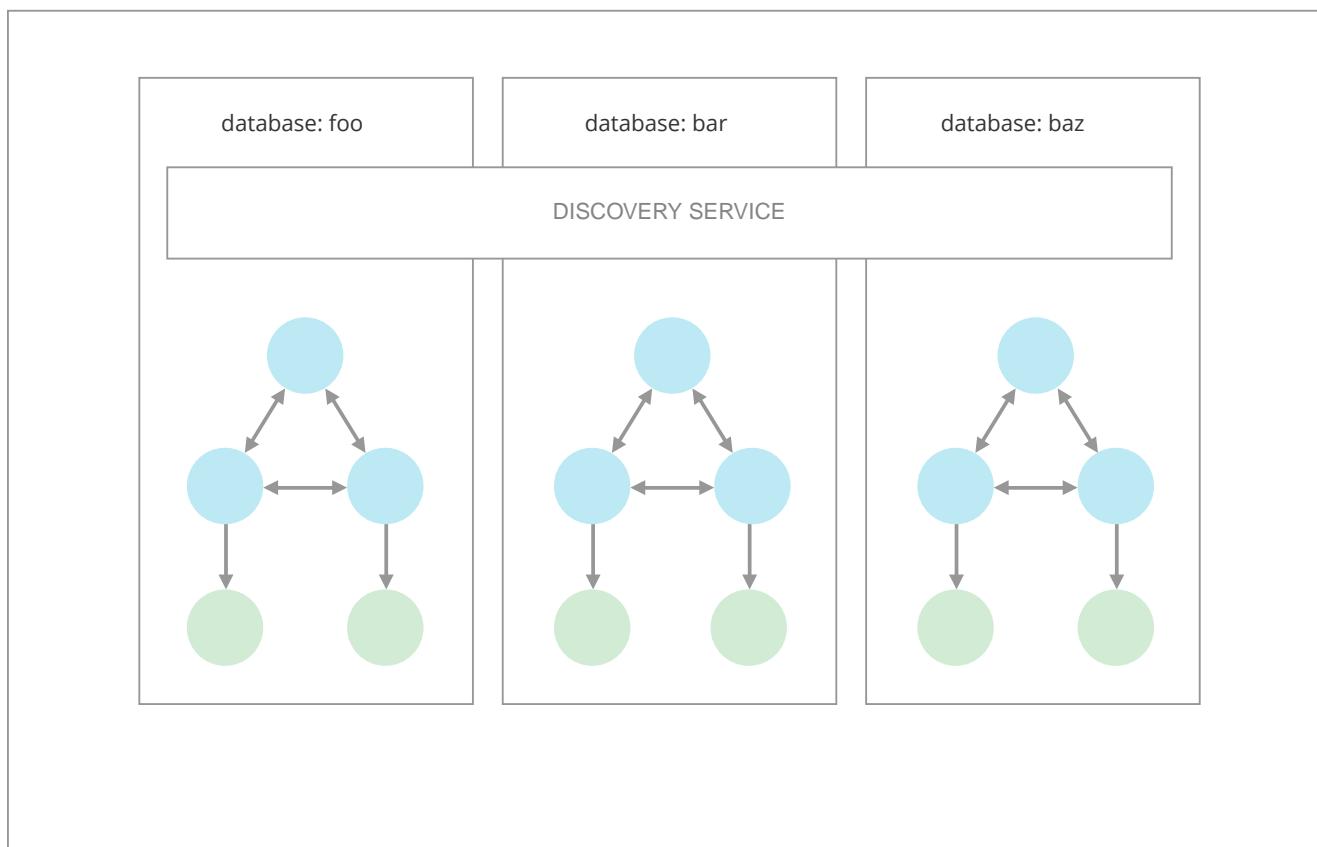


Figure 18. A multi-cluster with three distinct clusters.

In this scenario, all `foo` instances may go offline without affecting the ability of `bar` instances to answer queries. Similarly, elections in one cluster do not trigger elections in another.

The difference from running three independent clusters is that all members of a multi-cluster share a discovery service. Members of any single cluster are aware of the members of all other clusters, and with the names of the databases they manage. This makes it possible to ask any routing member in the multi-cluster for routing information for any database by name. Client connector libraries, like the Neo4j drivers, can therefore provide a unified directory service where databases may be retrieved and written to by name.

Known limitations

- Each cluster in a multi-cluster may only hold one distinct database. For example, if we wish our multi-cluster to host two distinct databases, we must provision at least 4 Core instances (i.e. `causal_clustering.minimum_core_cluster_size_atFormation` must be set to a minimum of 2 for each cluster).
- A single transaction may not read from, or update, multiple constituent clusters.
- No data is automatically shared or replicated between constituent clusters.
- If you stop and unbind the instances in a cluster, while the rest of the multi-cluster is still running, then you may need to restart all members from all clusters in order to have your recently unbound cluster rejoin the multi-cluster correctly.

5.7.2. Configure a multi-cluster

This section describes how to configure several Causal Clusters in order to construct a single multi-cluster.

In this section we will configure the multi-cluster illustrated in the picture below. Each cluster has three Core Servers and two Read Replicas:

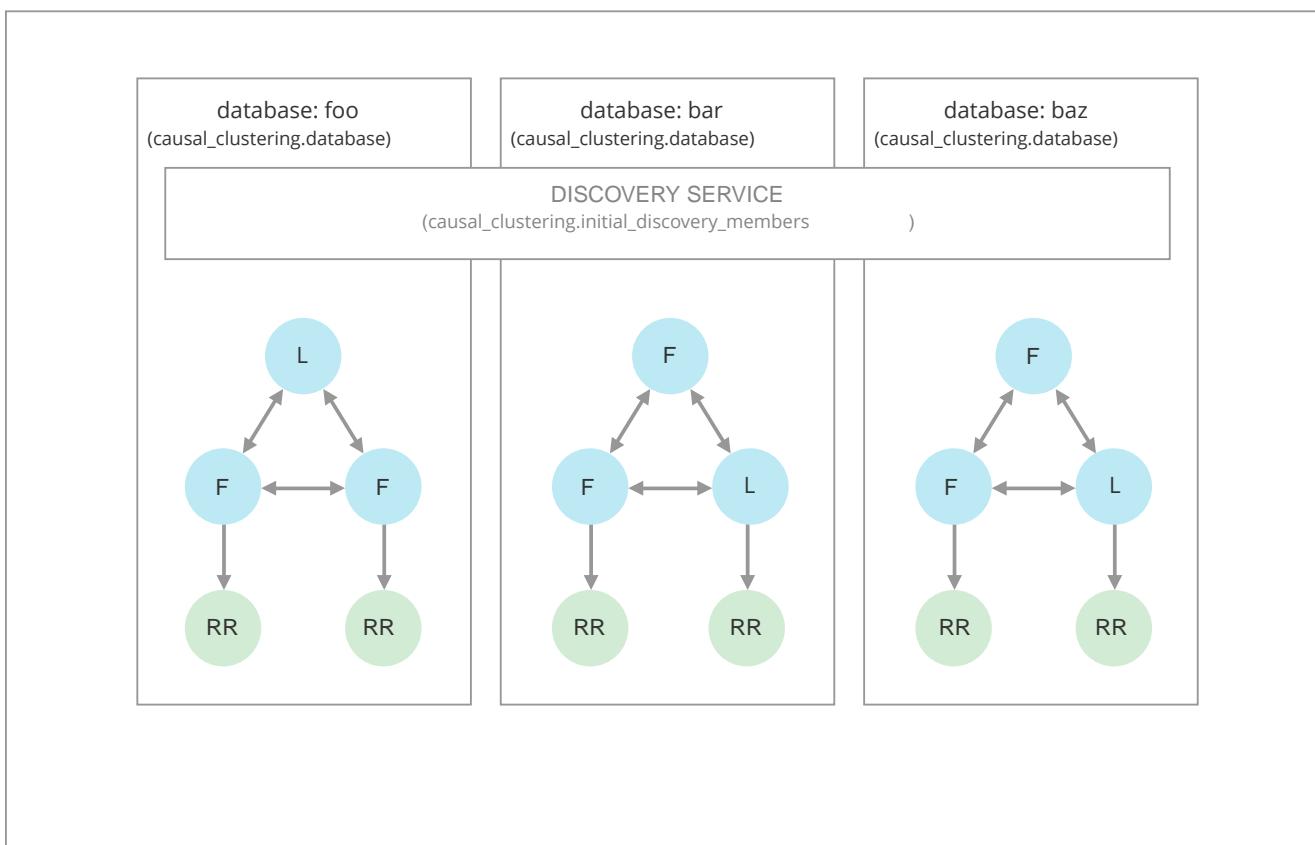


Figure 19. Configure a multi-cluster with three clusters.

1. Configure each cluster as you would configure a regular Causal Cluster.

Refer to [Create a new cluster](#) for details.

For example, if you configure a multi-cluster of three clusters, where each has three Core members, then in each of the clusters the value of `causal_clustering.minimum_core_cluster_size_atFormation` should be set to 3 (as opposed to 9, which would be the sum of all Core members).

2. Start up each of the clusters and test that it functions as expected.
3. Shut down all the members of all the clusters.
4. Assign a name to each of the databases/clusters:

Configure `causal_clustering.database` on each of the members in each of the clusters.

Example 30. Assign a name to a cluster

We wish to assign the name `foo` to this database. We do so by adding the following entry to `neo4j.conf`:

```
causal_clustering.database=foo
```



If an intended member of a cluster is missing the `causal_clustering.database` setting, it will form a separate cluster with the implicit name of `default`.

Limitations: Once an instance is first launched with a non-default name, that name is persisted. On subsequent startups of that instance, if a name change is detected it will cause the startup to fail. The database name configured for an instance may only be changed after unbinding the instance using `neo4j-admin unbind`.

5. Configure discovery members.

When setting up a multi-cluster, we must configure the discovery service to locate all members from all constituent clusters. This enables them to join the multi-cluster-wide discovery service depicted in the image above.

This is achieved by combining each cluster's `causal_clustering_initial_discovery_members` config value. See [Discovery protocol](#) for details about discovery.

Example 31. Initial discovery configuration for a multi-cluster

Assume that we have three clusters with initial discovery members configured as follows:

```
# `foo` members
causal_clustering.initial_discovery_members=neo20:5000,neo21:5000,neo22:5000
```

```
# `bar` members
causal_clustering.initial_discovery_members=neo23:5000,neo24:5000,neo25:5000
```

```
# `baz` members
causal_clustering.initial_discovery_members=neo26:5000,neo27:5000,neo28:5000
```

To form a multi-cluster with these three clusters, update the config value on all members of all clusters:

```
# `foo`, `bar` and `baz` members
causal_clustering.initial_discovery_members=neo20:5000,neo21:5000,neo22:5000,neo23:5000,neo24:5000,neo25:5000,neo26:5000,neo27:5000,neo28:5000
```

6. Start up and test the multi-cluster:

- Start up all the members of the multi-cluster.
- Use the `dbms.cluster.overview()` procedure to test that the multi-cluster works as expected.

Example 32. Test the multi-cluster

To test that the multi-cluster is working correctly, and to get information about the participating members and the databases they manage, use the `dbms.cluster.overview()` procedure:

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

The procedure will show output similar to the following:

id	addresses	role	groups	database
08eb9305-53b9-4394-9237-0f0d63bb05d5	[bolt://neo20:7687, http://neo20:7474, https://neo20:7473]	LEADER	[]	foo
cb0c729d-233c-452f-8f06-f2553e08f149	[bolt://neo21:7687, http://neo21:7474, https://neo21:7473]	FOLLOWER	[]	foo
ded9eed2-dd3a-4574-bc08-6a569f91ec5c	[bolt://neo22:7687, http://neo22:7474, https://neo22:7473]	FOLLOWER	[]	foo
3df6ec18-6333-4fb8-8b34-db7ecdf967f	[bolt://neo23:7687, http://neo23:7474, https://neo23:7473]	LEADER	[]	bar
395f708f-ae0c-4e24-a48d-9e90d4d63590	[bolt://neo24:7687, http://neo24:7474, https://neo24:7473]	FOLLOWER	[]	bar
fef25f21-6a99-4694-ae1c-10d91465acd2	[bolt://neo25:7687, http://neo25:7474, https://neo25:7473]	FOLLOWER	[]	bar
dcc57b70-d4a4-486b-ae57-e635bdcb1563	[bolt://neo26:7687, http://neo26:7474, https://neo26:7473]	LEADER	[]	baz
a664ede1-1f87-4828-aae8-1ffd2bcce09c	[bolt://neo27:7687, http://neo27:7474, https://neo27:7473]	FOLLOWER	[]	baz
17add42-a9f9-427e-9b32-e59d650e35af	[bolt://neo28:7687, http://neo28:7474, https://neo28:7473]	FOLLOWER	[]	baz
a049dea4-d4de-421c-898c-e638fa98440b	[bolt://neo30:7687, http://neo30:7474, https://neo30:7473]	READ_REPLICA	[]	foo
894ced90-85cd-47c2-9a75-f86b22ef74bf	[bolt://neo31:7687, http://neo31:7474, https://neo31:7473]	READ_REPLICA	[]	foo
151e46c9-14b8-49bf-bf25-cf09cf3ad582	[bolt://neo32:7687, http://neo32:7474, https://neo32:7473]	READ_REPLICA	[]	bar
e1d153aa-c3e6-40fd-acf7-d73c6c79baf4	[bolt://neo33:7687, http://neo33:7474, https://neo33:7473]	READ_REPLICA	[]	bar
6e0b1801-9dba-4925-bc45-ceee91c10338	[bolt://neo34:7687, http://neo34:7474, https://neo34:7473]	READ_REPLICA	[]	baz

id	addresses	role	groups	database
87577513-db46-4ca3-a75f-3d8919f9b454	[bolt://neo35:7687, http://neo35:7474, https://neo35:7473]	READ_REPLICA	[]	baz

The `dbms.cluster.overview()` procedure provides an overview of the whole multi-cluster, regardless of which cluster member is used to run the query, and which constituent cluster that member belongs to. The name of the database managed by each cluster member is provided by the *database* column.

5.8. Settings reference

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

5.8.1. Common settings

Parameter	Explanation
<code>dbms.mode</code>	This setting configures the operating mode of the database. For Causal Clustering, there are two possible modes: <code>CORE</code> or <code>READ_REPLICA</code> . Example: <code>dbms.mode=READ_REPLICA</code> will define this server as a Read Replica.
<code>causal_clustering.minimum_core_cluster_size_atFormation</code>	The expected cluster size at initial cluster formation. The minimum number of instances to form a safe cluster is two, but to allow for minimal operational fault tolerance it defaults to three such that the loss of one instance will leave the cluster operable. Example: <code>causal_clustering.minimum_core_cluster_size_atFormation=3</code> will specify that the cluster is intended to start with three Core members (and will be safe to start when a cluster of two out of three has formed).
<code>causal_clustering.minimum_core_cluster_size_atRuntime</code>	The minimum Core cluster size at runtime. The minimum number of instances to maintain a fault-tolerant cluster is two, but for sane operational characteristics three is the default. Example: <code>causal_clustering.minimum_core_cluster_size_atRuntime=3</code> will specify that the cluster should not drop below three Core members in the voting set (allowing a majority of just two instances to continue to run the cluster).
<code>causal_clustering.discovery_type</code>	Causal Clustering instances communicate over the network to ensure consistency of the database. Each member of the cluster must be given bootstrap information about the other members that are expected to be in the cluster. This setting specifies the strategy that the instance will use to determine the addresses for other instances in the cluster to contact for bootstrapping. The value of this setting determines how <code>causal_clustering.initial_discovery_members</code> is interpreted. Options are <code>LIST</code> , <code>DNS</code> , <code>SRV</code> , and <code>K8S</code> . <code>[.compact] LIST</code> :: Treat <code>causal_clustering.initial_discovery_members</code> as a list of addresses of Core Servers to contact for discovery. <code>DNS</code> :: 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> . <code>SRV</code> :: 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. <code>K8S</code> :: Ignores <code>causal_clustering.initial_discovery_members</code> . 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. Detailed information about discovery and discovery configuration options is given in the Initial discovery of cluster members section. Example: <code>causal_clustering.discovery_type=DNS</code> combined with <code>causal_clustering.initial_discovery_members=neo20.example.com:5000</code> will fetch all DNS A records for <code>neo20.example.com</code> and attempt to reach Neo4j instances listening on port 5000 for each A record's IP address.

Parameter	Explanation
<code>causal_clustering.initial_discovery_members</code>	Causal Clustering instances communicate over the network to ensure consistency of the database. An instance will contact these initial discovery members to both register itself as a member of the cluster and to ask them for contact information (advertised addresses) for other members of the cluster. Specifying an instance's own address is permitted. It is not necessary to specify an address for every other cluster member. However, if the initial discovery members configured for the cluster form two or more disjoint sets, then the cluster will not form correctly. Do not use any whitespace in this configuration option. Detailed information about discovery and discovery configuration options is given in the Initial discovery of cluster members section. Example: <code>causal_clustering.discovery_type=LIST</code> combined with <code>causal_clustering.initial_discovery_members=neo22:5001,neo21:5001,neo20:5001</code> will attempt to reach Neo4j instances listening on neo22 on port 5001 and neo21 on port 5001 and neo20 also on port 5001.
<code>causal_clustering.raft_advertised_address</code>	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. Example: <code>causal_clustering.raft_advertised_address=192.168.33.20:7000</code> will listen for cluster communication in the network interface bound to 192.168.33.20 on port 7000.
<code>causal_clustering.transaction_advertised_address</code>	The address/port setting that specifies where the instance advertises where it will listen for requests for transactions in the transaction-shipping catchup protocol. 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 192.168.33.20 on port 6001.
<code>causal_clustering.discovery_listen_address</code>	The address/port setting for use by the discovery protocol. This is the setting which will be included in the setting <code>causal_clustering.initial_discovery_members</code> which are set in the configuration of the other members of the cluster. 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 5001.
<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 7000.
<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 7000.
<code>causal_clustering.refuse_to_be_leader</code>	Prevents the current instance from volunteering to become Raft leader if set to <code>true</code> . Defaults to <code>false</code> , and should only be used in exceptional circumstances by expert users, or when advised by Neo4j Professional Services. Example: <code>causal_clustering.refuse_to_be_leader=false</code>
<code>causal_clustering.cluster_allow_reads_on_followers</code>	Defaults to <code>true</code> so that followers are available for read-only queries in a typical heterogeneous setup. Note: if there are no Read Replicas in the cluster, followers are made available for read, regardless the value of this setting. Example: <code>causal_clustering.cluster_allow_reads_on_followers=true</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>

5.8.2. 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> .

Parameter	Explanation
<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> .
<code>causal_clustering.user_defined_upstream_strategy</code>	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> . 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> .
<code>causal_clustering.load_balancing.plugin</code>	The load balancing plugin to use. One pre-defined plugin named <code>server_policies</code> is available by default. Example: <code>causal_clustering.load_balancing.plugin=server_policies</code> will enable custom policy definitions.
<code>causal_clustering.load_balancing.config.server_policies.<policy-name></code>	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. 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.

Chapter 6. Upgrade

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

It is recommended that your installation of Neo4j is kept up to date. Upgrading your installation to Neo4j 3.5.0 will ensure that you are provided with improvements in performance and security, as well as any latest bug fixes.

This chapter describes the following:

- [Upgrade planning](#)
 - [Supported upgrade paths](#)
 - [Prepare to upgrade](#)
- [Single-instance upgrade](#)
 - [Upgrade from 2.x](#)
 - [Upgrade from 3.x](#)
- [Upgrade a Causal Cluster](#)

6.1. Upgrade planning

This section describes how to plan for an upgrade of Neo4j.

Plan your upgrade by following the steps in this chapter.

6.1.1. Supported upgrade paths

The following Neo4j upgrade paths are supported:

- 2.3.latest ↳ 3.5.0
- 3.0.latest ↳ 3.5.0
- 3.1.latest ↳ 3.5.0
- 3.2.latest ↳ 3.5.0
- 3.3.latest ↳ 3.5.0
- 3.4.any ↳ 3.5.0

Limitations

A Neo4j upgrade must be performed as an isolated operation. If you have other changes planned, such as switching from a single-instance installation to a Causal Cluster, you must action these separately.

6.1.2. Prepare to upgrade

1. Review supported upgrade paths.

Before upgrading to a new major or minor release, the database must first be upgraded to the latest version within the relevant release. The latest version is available at this page:
<http://neo4j.com/download/other-releases>.

2. Review Release Notes.

To view the details of the changes that are included in each version, see the [Release Notes](https://neo4j.com/release-notes) (<https://neo4j.com/release-notes>).

3. Apply configuration changes.

- a. With each version, new configuration settings may be introduced and existing configurations may change. On rare occasions, settings can change name between versions. Any such changes are detailed in the *Upgrade guide*, as mentioned above. Please ensure that you have taken such changes into account.
- b. It is also useful to inspect the current configuration file and take note of any non-default settings. When upgrading, it is particularly important to note any custom values of the settings `dbms.directories.data` and `dbms.active_database`. In cluster installations, pay attention to cluster-specific configuration settings, which may be different on different cluster members.

4. Plan index upgrade.

Review [Index configuration](#) and determine whether you have indexes that ought to be rebuilt in order to take advantage of performance improvements. Include the rebuilding of these indexes in your test upgrade.

5. Upgrade application code.

As part of upgrade planning, it is vital to test and potentially update all applications that use Neo4j. How much development time is required to update the code will depend on the particular application.

In particular, the Cypher language may evolve between Neo4j versions. For backward compatibility, Neo4j provides directives which enable you to explicitly select a previous Cypher language version. This is possible to do globally or for individual statements, as described in the [Neo4j Cypher Manual](#).

6. Upgrade custom plugins.

Check the `plugins` directory (see [File locations](#)) to verify whether custom plugins are used in your deployment. Ensure that any plugins are compatible with Neo4j 3.5.0.

7. Plan disk space requirements.

An upgrade requires substantial free disk space, as it makes an entire copy of the database. For the upgrade, make sure to make available an additional ($50\% * \text{size_of}(\text{database directory})$), where the default `database directory` is located in: `data/databases/graph.db`). In addition to this, do not forget to reserve the disk space needed for the pre-upgrade backup.

The upgraded database may require slightly larger data files overall.

8. Perform a test upgrade.

Based on the findings in this chapter, allocate a 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, as well as potential additional action points, such as upgrade of plugins and application code.

9. Review the logs.

The `neo4j.log` file contains valuable information on how many steps the upgrade will involve and how far it has progressed. For large upgrades, it is a good idea to monitor this log continuously. Below is a sample of what the log may look like:

```

2018-09-18 13:24:23.243+0000 INFO Starting...
2018-09-18 13:24:24.262+0000 INFO Initiating metrics...
2018-09-18 13:24:24.488+0000 INFO Starting upgrade of database
2018-09-18 13:24:24.538+0000 INFO Migrating Indexes (1/5):
2018-09-18 13:24:24.542+0000 INFO    10% completed
2018-09-18 13:24:24.543+0000 INFO    20% completed
2018-09-18 13:24:24.543+0000 INFO    30% completed
...
...
2018-09-18 13:24:24.574+0000 INFO Migrating Counts store (5/5):
2018-09-18 13:24:24.574+0000 INFO    10% completed
2018-09-18 13:24:24.574+0000 INFO    20% completed
2018-09-18 13:24:24.575+0000 INFO    30% completed
...
...
2018-09-18 13:24:24.576+0000 INFO    100% completed
2018-09-18 13:24:24.584+0000 INFO Successfully finished upgrade of database

```

6.1.3. Additional resources

In addition to the steps outlined in this chapter, it is valuable to review the [Upgrade guide](#) at [neo4j.com](https://neo4j.com/guides/upgrade/) (<https://neo4j.com/guides/upgrade/>). The upgrade guide is maintained by Neo4j Customer Support, and contains valuable information about upgrade actions particular to this release.

6.2. Upgrade a single instance

This section describes how to upgrade a single Neo4j instance.

For instructions on upgrading a Neo4j Causal Cluster, see [Upgrade a Causal Cluster](#). For instructions on upgrading a Neo4j HA cluster, see [Upgrade](#).

6.2.1. Upgrade from 2.x

If you use Debian, refer to [Upgrade](#).

Pre-upgrade steps

Read [Upgrade planning](#) thoroughly and perform all the steps listed there.

Shutdown and backup

1. If the database is running, shut it down cleanly.
2. Make a backup copy of the database directory (in a default configuration: `data/databases/graph.db`), and of `neo4j.conf`.

If using the [online backup tool](#), ensure that backups have completed successfully.

3. Back up `neo4j.conf`.

Upgrade

1. Install Neo4j 3.5.0.
2. Review the settings in the configuration files of the previous installation and transfer any custom settings to the 3.5.0 installation:
 - Since many settings have been changed between Neo4j 2.x and 3.5.0, it is advisable to use the `2.x-config-migrator` (available in the `tools` directory) to migrate the configuration files for you.

For example, the `2.x-config-migrator` can be invoked with a command like:

```
java -jar 2.x-config-migrator.jar path/to/neo4j2.3 path/to/neo4j3.5.0.
```

- Take note of any warnings printed, and manually review the edited configuration files produced.

3. Use the following to import your data from the old installation:

```
neo4j-admin import --mode=database --database=<database-name> --from=<source-directory>
```

4. If the database is not called `graph.db`, set `dbms.active_database` in `neo4j.conf` to the name of the database.
5. Set `dbms.allow_upgrade=true` in `neo4j.conf` of the 3.5.0 installation. Neo4j will fail to start without this configuration.
6. Start up Neo4j 3.5.0. The database upgrade will take place during startup.

The `neo4j.log` file contains valuable information on how many steps the upgrade will involve and how far it has progressed. For large upgrades, it is a good idea to monitor this log continuously.

Post-upgrade steps

1. When the upgrade has finished, `dbms.allow_upgrade` should be set to `false` or be removed.
2. Restart the database.
3. It is good practice to make a full backup immediately after the upgrade.

6.2.2. Upgrade from 3.x

Pre-upgrade steps

Read [Upgrade planning](#) thoroughly and perform all the steps listed there.

Shutdown and backup

1. If the database is running, shut it down cleanly.
2. Make a backup copy of the database directory (in a default configuration: `data/databases/graph.db`), and of `neo4j.conf`.

If using [offline backups](#), take a backup of Neo4j.

If using the [online backup tool](#), ensure that backups have completed successfully.

3. Back up `neo4j.conf`.
4. If using [native user and role management](#), back up the `dbms` directory (in a default configuration: `data/dbms`).

Upgrade

1. Install Neo4j 3.5.0 using one of the following methods, specific to your technology:
 - a. If using a tarball or zipfile for installation:
 - i. Untar or unzip Neo4j 3.5.0.
 - ii. Transfer any custom settings to the 3.5.0 installation, as noted under the *Apply configuration changes* step in [Upgrade planning](#).
 - iii. Set `dbms.allow_upgrade=true` in `neo4j.conf` of the 3.5.0 installation. Neo4j will fail to start without this configuration.

- iv. Copy the `data` directory from the old installation to the new one. This step is not applicable if you have `dbms.directories.data` pointing to a directory outside of `NEO4J_HOME`.
- b. If using a Debian or RPM distribution:
 - i. Set `dbms.allow_upgrade=true` in `neo4j.conf`.
 - ii. Install Neo4j 3.5.0.
 - iii. When prompted, review the differences between the `neo4j.conf` files of the previous version and Neo4j 3.5.0. Transfer any custom settings to the 3.5.0 installation, as noted under the *Apply configuration changes* step in [Upgrade planning](#). Make sure to preserve `dbms.allow_upgrade=true` as set in the instruction above. Neo4j will fail to start without this configuration.
2. Start up Neo4j 3.5.0. The database upgrade will take place during startup.

The `neo4j.log` file contains valuable information on how many steps the upgrade will involve and how far it has progressed. For large upgrades, it is a good idea to monitor this log continuously.

Post-upgrade steps

1. When the upgrade has finished, `dbms.allow_upgrade` should be set to `false` or be removed.
2. Restart the database.
3. It is good practice to make a full backup immediately after the upgrade.

6.3. Upgrade a Causal Cluster

This section describes how to upgrade a Neo4j Causal Cluster.

This section describes the following:

- [Important pre-upgrade information](#)
- [Rolling upgrade](#)
 - [Rolling upgrade for fixed servers](#)
 - [Rolling upgrade for replaceable resources](#)
- [Offline upgrade](#)

6.3.1. Important pre-upgrade information

All upgrades should be approached with careful planning and testing.

Pre-upgrade steps

1. Read [Upgrade planning](#) thoroughly and perform all the steps listed there.
2. Perform and verify backups:
 - Back up `neo4j.conf`.
 - If using [native user and role management](#), back up the `dbms` directory (in a default configuration: `data/dbms`).
 - Verify that you have a [full backup](#) that is stored in a safe location.
3. Prepare a new `neo4j.conf` file for each of the servers in the cluster, following the instructions

under the *Apply configuration changes* step in [Upgrade planning](#).

If following instructions for [Rolling upgrade for replaceable resources](#), all the servers that form the cluster will be replaced. Therefore, make sure that the parameter `causal_clustering.initial_discovery_members` is updated to reflect the new Core members.

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.
- In order to minimize risk, it is recommended that while upgrading, you do not change configuration, perform architectural restructuring, or similar tasks.

Monitoring the status of cluster members

Use the procedure `dbms.cluster.overview()` to monitor the status of the cluster members during the upgrade.

Upgrade methods

The following are methods of upgrading a Neo4j Causal Cluster:

- [Rolling upgrade](#); this method is available under certain conditions.
 - [Rolling upgrade for fixed servers](#)
 - [Rolling upgrade for replaceable resources](#)
- [Offline upgrade](#); this method is always available.

Installing the new Neo4j version

For instructions on how to install the software package itself, refer to [Installation](#).

Post-upgrade

Note that backups taken before the upgrade are no longer valid for update via the incremental online backup. Therefore, it is important to perform a *full backup*, using an empty target directory, immediately after the upgrade.

6.3.2. Rolling upgrade

Limitations

Rolling upgrades are only supported between patch releases, and only when a store format upgrade is not needed. Read the [Data Store Changes](https://neo4j.com/release-notes/data-store-changes/) (<https://neo4j.com/release-notes/data-store-changes/>) page to find out whether a certain upgrade path involves a store format change. You can also contact Customer Support to obtain this information.

If you cannot use rolling upgrades, then follow the instructions for [offline upgrades](#) instead.

Downtime

Since downgrades are not supported, a failed upgrade will require recovery from a pre-upgrade backup. Therefore, the safest path is to disable write loads during the upgrade process.

Recommendations

- When upgrading a cluster, it is important to do so in a systematic, predictable way. The critical point during the upgrades is knowing when to switch off the original members. It is recommended that you use the status endpoint when performing a rolling upgrade, which provides access to information that can aid you when deciding which member to switch off, and when it is safe to do so. For more information, see [Status endpoint](#).



It is only possible to use the status endpoint when upgrading from Neo4j v3.5, since that is when the status endpoint was introduced.

- Since replacing cores is no different from temporarily removing cores, the process of doing a rolling upgrade reduces fault tolerance temporarily. If you absolutely need to maintain a minimum threshold of fault tolerance, then you should increase the cluster size to account for this planned failure.

Rolling upgrade for fixed servers

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

Upgrade steps

1. On one of the Core Servers:
 - a. (Optional/Recommended) Use the [status endpoint](#) to evaluate whether it is safe to remove this member.
 - b. Shutdown the instance.
 - c. Install Neo4j using one of the following methods, specific to your technology:
 - If using a tarball or zipfile for installation:
 - i. Untar or unzip the version of Neo4j that you want to upgrade to.
 - ii. Place the *neo4j.conf* file, that you have prepared for this server, into the [Configuration](#) directory.
 - iii. Copy the *data* directory from the old installation to the new one. This step is not applicable if you have [dbms.directories.data](#) pointing to a directory outside of *NEO4J_HOME*.
 - If using a Debian or RPM distribution:
 - i. Install the version of Neo4j that you want to upgrade to.
 - ii. 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 in the pre-upgrade step.
 - d. Start up Neo4j, and see it join the cluster.
2. Repeat the previous steps for every server instance.

Rolling upgrade for replaceable resources

This variant is suitable for deployments utilizing replaceable cloud or container resources. With this method, it is possible to avoid a reduction in availability, since you are adding a new instance before removing an old instance.

Upgrade steps

1. Configure a new instance with the version of Neo4j that you want to upgrade to.
2. Use the new *neo4j.conf* that you have prepared in the pre-upgrade step.
3. If applicable, place the backed-up *data/dbms* directory into the *data* directory of the new instance.
4. Start up the new instance, and let it complete a store copy.
5. See the new instance join the cluster.
6. (Optional/Recommended) Use the [status endpoint](#) to evaluate whether it is safe to remove an old member.

7. Shutdown the old instance.
8. Repeat the previous steps for every server instance.

6.3.3. Offline upgrade

This variant is suitable for cases where a rolling upgrade is not possible due to breaking changes, or could be undesirable for other reasons.

Downtime

If the offline upgrade method is selected, this will involve downtime. A test upgrade on a production-like equipment provides information on the duration of the downtime.

Upgrade steps

1. Shut down all the servers in the cluster
 2. On one of the Core Servers:
 - a. Set `dbms.mode=SINGLE` in `neo4j.conf`.
 - b. Install Neo4j using one of the following methods, specific to your technology:
 - If using a tarball or zipfile for installation:
 - i. Untar or unzip the version of Neo4j that you want to upgrade to.
 - ii. Place the `neo4j.conf` file, that you have prepared for this server, into the *Configuration* directory.
 - iii. Set `dbms.allow_upgrade=true` in `neo4j.conf`. Neo4j will fail to start without this configuration.
 - iv. Copy the `data` directory from the old installation to the new one. This step is not applicable if you have `dbms.directories.data` pointing to a directory outside of `NEO4J_HOME`.
 - If using a Debian or RPM distribution:
 - i. Set `dbms.allow_upgrade=true` in `neo4j.conf`.
 - ii. Install the version of Neo4j that you want to upgrade to.
 - iii. 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 in the pre-upgrade step. Make sure to preserve `dbms.allow_upgrade=true` as set in the instruction above. Neo4j will fail to start without this configuration.
 - c. Start up Neo4j. The database upgrade will take place during startup.
- The `neo4j.log` file contains valuable information on how many steps the upgrade will involve and how far it has progressed. For large upgrades, it is a good idea to monitor this log continuously.
- d. Set `dbms.allow_upgrade=false`, or remove it.
 - e. Set `dbms.mode=CORE` in `neo4j.conf` to re-enable Causal Clustering in the configuration.
 - f. Use `neo4j-admin dump` to make a copy of the database.
 - g. Do not yet start the database.
3. On each of the other Core Servers:
 - a. Delete the database directory (in a default configuration: `data/databases/graph.db`).

- b. Install the version of Neo4j that you want to upgrade to.
 - c. Transfer any custom settings to the new installation, as prepared in the pre-upgrade step.
 - d. If applicable, place the backed-up *data/dbms* directory into the *data* directory of the new instance.
 - e. Perform `neo4j-admin unbind` on the instance.
 - f. Using `neo4j-admin load`, restore the upgraded database onto this server.
4. Startup all the Core Servers and see the cluster form.
5. On each of the Read Replica servers:
- a. Stop Neo4j.
 - b. Delete the database directory (in a default configuration: *data/databases/graph.db*).
 - c. Install the version of Neo4j that you want to upgrade to..
 - d. Transfer any custom settings to the new installation, as prepared in the pre-upgrade step.
 - e. If applicable, place the backed-up *data/dbms* directory into the *data* directory of the new instance.
 - f. Using `neo4j-admin dump/load`, restore the upgraded database onto this server. Alternatively, you can omit this step and let the Read Replica do a complete store copy.
 - g. Start the Read Replica and see it join the cluster.

Chapter 7. Backup

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

This chapter describes the following:

- [Backup planning](#)
 - [Online and offline backups](#)
 - [Storage considerations](#)
 - [Network protocols for backups](#)
 - [Memory configuration](#)
- [Standalone databases](#)
 - [Configuration parameters](#)
 - [Running backups from a separate server](#)
 - [Running backups from the local server](#)
- [Causal Clusters](#)
 - [Introduction](#)
 - [Configuration parameters](#)
 - [Encrypted backups](#)
 - [Running backups from a Read Replica](#)
 - [Running backups from a Core Server](#)
 - [Backup scenarios and examples](#)
- [Perform a backup](#)
 - [Backup commands](#)
 - [Full backups](#)
 - [Incremental backups](#)
 - [Exit codes](#)
- [Restore a backup](#)
 - [Restore commands](#)
 - [Restore a single database](#)
 - [Restore a Causal Cluster](#)

7.1. Backup planning

This section introduces how to prepare for backing up a Neo4j database.

This section includes:

- [Introduction](#)
- [Online and offline backups](#)

- Storage considerations
- Network protocols for backups
- Memory configuration

7.1.1. Introduction

Designing an appropriate backup strategy for your Neo4j database is a fundamental part of database 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?

7.1.2. 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 live Neo4j instance, while offline backups require that the database is shut down.
- 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.

7.1.3. 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 should use a different availability zone within the same cloud, or use a separate cloud for backups.

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 rotating transaction log files. The transaction log files are files that keep track of recent changes. Recovery from backups with the same transaction log files as the source server can be helpful, but it isn't always necessary. Please note that removing transactions 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 in either the default backup configuration (`$/NEO4J_HOME/conf/neo4j.conf`), or in the `$/NEO4J_CONF` config file. Alternatively you can use the `--additional-config` override.

7.1.4. Network protocols for backups

The backup client can use two different protocols:

- Backups of members of a [Causal Cluster](#), whether Core Servers or Read Replicas, use the [catchup protocol](#). This is the same protocol that is used for keeping Read Replicas up-to-date within a Causal Cluster.
- Backups of single-instance servers use the [common backup protocol](#)

Since the backup client is not aware of, ahead of time, what type of server it will run the backup against, it will at first attempt the catchup protocol. If that does not succeed, it will try the common backup protocol. If you want to control this behavior, you can use the `--protocol` option when performing a backup.

7.1.5. Memory configuration

The following options are available for controlling memory allocation 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.

7.2. Standalone databases

This section discusses configuration options for backups of standalone databases.

This section includes:

- [Configuration parameters](#)
- [Running backups from a separate server](#)
- [Running backups from the local server](#)

7.2.1. Configuration parameters

The table below lists the configuration parameters relevant to backups:

Table 11. Configuration parameters backups; standalone databases

Parameter name	Default value	Description
dbms.backup.enabled	true	Enable support for running online backups.
dbms.backup.address	127.0.0.1:6362-6372	Listening server for online backups.

7.2.2. Running backups from a separate server

Consider the image below:



In this configuration, the backup client is run on a separate server. This is the recommended configuration for a standalone database.

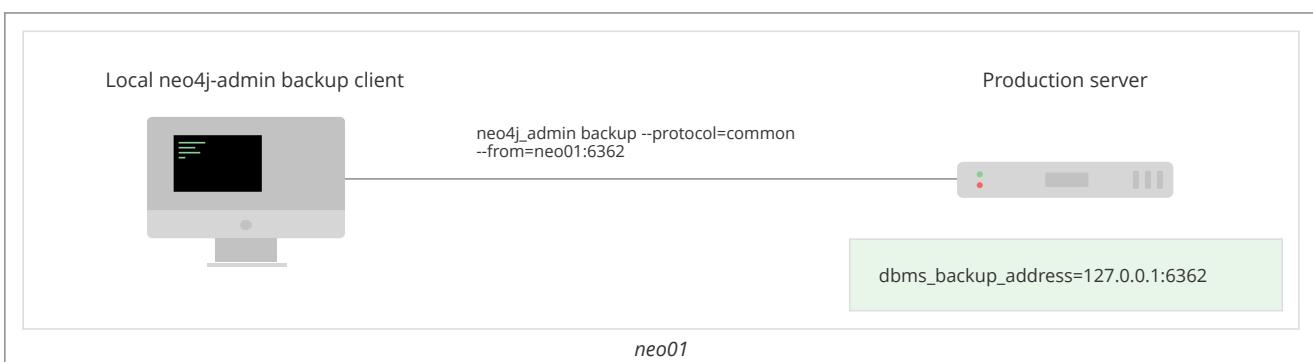
The server that runs the backup client must have Neo4j installed, but does not need to host a database.



It is strongly recommended to run backups from a different server than the production server.

7.2.3. Running backups from the local server

Consider the image below:



In this configuration, the backup program is run on the same server that hosts the Neo4j production database.

When the backup program is started, it will start up a new Java process. If there is a running Neo4j database, this will run in parallel to the Neo4j process. On a production system, Neo4j is typically configured to take maximum advantage of the system's available RAM. If you run backups on a production system, the overall performance can be negatively affected, and in extreme cases can cause failure with an out-of-memory error. It is therefore strongly recommended to run backups from a different server than the production server.

If it is not possible to use a separate backup server, you can control the impact on the production system by explicitly defining how much memory to allocate to the backup process. See [Memory configuration](#).

7.3. Causal Clusters

This section discusses considerations when designing the backup strategy for a Neo4j Causal Cluster.

This section includes:

- [Introduction](#)
- [Configuration parameters](#)
- [Encrypted backups](#)
- [Running backups from a Read Replica](#)
- [Running backups from a Core Server](#)
- [Backup scenarios and examples](#)

7.3.1. Introduction

Backups of a Neo4j Causal Cluster can be configured in a variety of ways with regards to physical configuration and SSL implementation. This section discusses some considerations that should be taken into account before determining which backup configuration to use.

7.3.2. Configuration parameters

The table below lists the configuration parameters relevant to backups:

Table 12. Configuration parameters backups; Causal Clusters

Parameter name	Default value	Description
<code>dbms.backup.enabled</code>	<code>true</code>	Enable support for running online backups.
<code>dbms.backup.address</code>	<code>127.0.0.1:6362</code>	Listening server for online backups.
<code>dbms.backup.ssl_policy</code>		The SSL policy used on the backup port.

7.3.3. Encrypted backups

Encrypted backups are available with Causal Clustering.

Both the server running the backup, and the backup target, must be configured with the same [SSL policy](#). This can be the same as that used for encrypting the regular cluster traffic (see [Intra-cluster encryption](#)), or a separate one. The policy to be used for encrypting backup traffic must be assigned on both servers.

For examples on how to configure encrypted backups, see [Backup scenarios and examples](#).

7.3.4. Running backups from a Read Replica

It is generally recommended to select Read Replicas to act as backup providers, 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.

However, since Read Replicas are asynchronously replicated from Core Servers, it is possible for them to be fall behind in applying transactions with respect to the Core Cluster. It may even be possible for a Read Replica to become orphaned from a Core Server such that its contents are quite stale.

We can use transaction IDs in order to avoid taking a backup from a Read Replica that has lagged too far behind the Core Cluster. Since transaction IDs are strictly increasing integer values, we can check the last transaction ID processed on the Read Replica and verify that it is sufficiently close to the latest transaction ID processed by the Core Server. If so, we can safely proceed to backup from our Read Replica in confidence that it is up-to-date with respect to the Core Servers.

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.

7.3.5. Running backups from a Core Server

In a Core-only cluster, we do not have access to Read Replicas for scaling out workload. Instead, we pick one of the Core Servers to run backups based on factors such as its physical proximity, bandwidth, performance, and liveness.

The cluster will function as normal even while large backups are taking place. However, the additional I/O burdens placed on the Core Server being used as a backup server, may impact its performance.

A very conservative view would be to treat the backup server as an unavailable instance, assuming its performance will be lower than the other instances in the cluster. In such cases, it is recommended that there is sufficient redundancy in the cluster such that one slower server does not reduce the capacity to mask faults.

We can factor this conservative strategy into our cluster planning. The equation $M = 2F + 1$ demonstrates the relationship between M being the number of members in the cluster required to tolerate F faults. To tolerate the possibility of one slower machine in the cluster during backup we increase F . Thus if we originally envisaged a cluster of three Core Servers to tolerate one fault, we could increase that to five to maintain a plainly safe level of redundancy.

7.3.6. Backup scenarios and examples

As described in [Network protocols for backups](#), the `catchup` protocol is used both for keeping Read Replicas up-to-date within a Causal Cluster, and for backups. It is therefore possible to run backups by defining a separate `dbms.backup.address` for backup traffic, or simply by "listening to" the same messages as Read Replicas do for keeping in sync with the Core Cluster.

To perform backups on a Causal Cluster, you will need to combine some settings and arguments. The table below illustrates the available options when using the `catchup` protocol:

Table 13. Causal Clustering backup settings

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

Before performing a backup of a Causal Cluster, you need to consider which port you will be performing backup from. For example, if you are planning to perform a backup from the transaction port, the backup policy for your backup client should match the cluster policy of the server. Otherwise, if you are planning to perform the backup from the backup port, the backup policy for the backup

client should match the server's backup policy.

The images below illustrate the settings and arguments to be used when setting up backups for your cluster using either `dbms.backup.ssl_policy` or `causal_clustering.ssl_policy`:



Figure 20. Settings and arguments for `dbms.backup.ssl_policy`



Figure 21. Settings and arguments for `causal_clustering.ssl_policy`

7.4. Perform a backup

This section describes how to perform a backup of a Neo4j database.

This section includes:

- [Backup commands](#)
- [Full backups](#)
- [Incremental backups](#)
- [Exit codes](#)

7.4.1. Backup commands

A Neo4j database can be backed up in online mode, using the `backup` command of `neo4j-admin`.

Syntax

```
neo4j-admin backup --backup-dir=<backup-path> --name=<graph.db-backup>
[--from=<address>] [--protocol=<any|catchup|common>]
[--fallback-to-full[=<true|false>]]
[--pagecache=<pagecache>]
[--timeout=<timeout>]
[--check-consistency[=<true|false>]]
[--additional-config=<config-file-path>]
[--cc-graph[=<true|false>]]
[--cc-indexes[=<true|false>]]
[--cc-label-scan-store[=<true|false>]]
[--cc-property-owners[=<true|false>]]
[--cc-report-dir=<directory>]
```

Options

Option	Default	Description
--backup-dir		Directory to place backup in.
--name		Name of backup. If a backup with this name already exists an incremental backup will be attempted.
--from	localhost:6362	Host and port of Neo4j.
--protocol	any	Protocol over which to perform backup. When you use any , catchup is tried first. If that fails, then it will fall back to common . There shouldn't be a need to set this, but if it takes time to initiate a backup, or if you would like the backup to fail fast, then you can set this to catchup for causal cluster backups, or common for HA or single instance backups. For more information, see Backup scenarios and examples .
--fallback-to-full	true	If an incremental backup fails backup will move the old backup to <name>.err.<N> and fallback to a full backup instead.
--pagecache	8M	The size of the page cache to use for the backup process.
--timeout	20m	Timeout in the form <time>[ms s m h], where the default unit is seconds. This is a debugging option that should only be used if instructed to by Neo4j Professional Services.
--check-consistency	true	If a consistency check should be made.
--additional-config		Configuration file to supply additional configuration in.
--cc-graph	true	Perform checks between nodes, relationships, properties, types and tokens.
--cc-indexes	true	Perform checks on indexes.
--cc-label-scan-store	true	Perform checks on the label scan store.
--cc-property-owners	false	Perform additional checks on property ownership. This check is very expensive in time and memory.
--cc-report-dir	.	Directory where consistency report will be written.

7.4.2. Full backups

A full backup is performed whenever there is no backup directory specified.

Example 33. Back up a database

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.

Now you can perform a full backup:

```
$neo4j-home> export HEAP_SIZE=2G
$neo4j-home> mkdir /mnt/backup
$neo4j-home> bin/neo4j-admin backup --from=192.168.1.34 --backup-dir=/mnt/backup --name=graph.db-
backup --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/backup` you will now see that you have a backup of Neo4j called `graph-db.backup`.

7.4.3. Incremental backups

An incremental backup is performed whenever an existing backup directory is specified, and the transaction logs are present since the last backup. The backup command will then copy any new transactions from Neo4j and apply them to the backup. The result will be an updated backup that is consistent with the current server state.

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

Example 34. Perform an 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/backup --name=graph.db-
backup --fallback-to-full=true --check-consistency=true --pagecache=4G
Destination is not empty, doing incremental backup...
Backup complete.
```

The incremental backup will fail if the existing directory does not contain a valid backup and `--fallback-to-full=false`. It will also fail if the required transaction logs have been removed and `--fallback-to-full=false`. Setting `--fallback-to-full=true` is a safeguard which will result in a full backup in case an incremental backup cannot be performed.

It is important to note that `--check-consistency` is `true` by default. For a quicker incremental backup we can set this to `--check-consistency=false` and `--fallback-to-full=false`.



When copying outstanding transactions, the server needs access to the transaction logs.

These logs are maintained by Neo4j and are automatically removed after a period of time, based on the parameter `dbms.tx_log.rotation.retention_policy`.

i When designing your backup strategy it is important to configure `dbms.tx_log.rotation.retention_policy` such that transaction logs are kept between incremental backups.

7.4.4. 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 14. Neo4j Admin backup exit codes

Code	Description
0	Success.
1	Backup failed.
2	Backup succeeded but consistency check failed.
3	Backup succeeded but consistency check found inconsistencies.

7.5. Restore a backup

This section describes how to restore from a backup of a Neo4j database.

This section includes:

- [Restore commands](#)
- [Restore a single database](#)
- [Restore a Causal Cluster](#)

7.5.1. Restore commands

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

Syntax

```
neo4j-admin restore --from=<backup-directory> [--database=<name>]
[--force[=<true|false>]]
```

Options

Option	Default	Description
<code>--from</code>		Path to backup to restore from.
<code>--database</code>	<code>graph.db</code>	Name of database.
<code>--force</code>	<code>false</code>	If an existing database should be replaced.

7.5.2. Restore a single database

To restore from a backup, follow these steps:

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

Example 35. Restore a single database

Restore the database `graph.db` from the backup located in `/mnt/backup/graph.db-backup`. Note that the database to be restored must be shut down.

```
neo4j-home> bin/neo4j stop  
neo4j-home> bin/neo4j-admin restore --from=/mnt/backup/graph.db-backup --database=graph.db --force  
neo4j-home> bin/neo4j start
```

7.5.3. Restore a Causal Cluster

To restore from a backup in a Causal Cluster, follow these steps:

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

Chapter 8. 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](#)
- [Terminology](#)
- [Configuration](#)
- [Native user and role management](#)
 - Native roles
 - Custom roles
 - Propagate users and roles
 - Procedures for native user and role management
- [Integration with LDAP](#)
- [Subgraph access control](#)
 - Introduction
 - Configuration steps
- [Property-level access control](#)
 - Introduction
 - Implement a property blacklist



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. [User management for Community Edition](#) describes the available functionality in detail.

8.1. Introduction

This section provides an overview of authentication and authorization in Neo4j.

Security in Neo4j is controlled by authentication and authorization. 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). The core of the Neo4j security model is centred around a number of predefined graph-global data-access roles. Each role includes a set of authorized actions permitted on the Neo4j data graph and its schema. A user can be assigned to none, one or more of these roles, as well as other custom roles.

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 locally on disk. With

this option, full user management is available as procedures described in [Native user and role management](#).

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](#).

Custom-built plugin auth providers

For clients with specific requirements not satisfied with either native or LDAP, Neo4j provides a plugin option for building custom integrations. It is recommended that this option is used as part of a custom delivery as negotiated with Neo4j Professional Services. The plugin is described in [Java Reference](#) [Authentication and authorization plugins](#).

Kerberos authentication and single sign-on

In addition to LDAP, Native and custom providers, Neo4j supports Kerberos for authentication and single sign-on. Kerberos support is provided via the [Neo4j Kerberos Add-On](#) (<https://neo4j.com/docs/add-on/kerberos/1.0>).

8.2. 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. There are two types of roles in Neo4j:

- Native roles are described in [Native roles](#).
- Custom roles are described in [Custom roles](#).

suspended user

A [user](#) who has been suspended is not able to access the database in any capacity, regardless of any assigned [roles](#).

user

- A user is composed of a username and credentials, where the latter is a unit of information, such as a password, verifying the identity of a user.
- A user may represent a human, an application etc.

8.3. Configuration

This section describes how to enable and disable authentication and authorization in Neo4j.

Authentication and authorization is enabled by default. It is possible to turn off all authentication and authorization. This is done using the setting `dbms.security.auth_enabled`.

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

This configuration makes the database vulnerable to malicious activities. Disabling authentication and authorization is not recommended.

8.4. Native user and role management

This section describes native user and role management in Neo4j.

This section describes the following:

- [Native roles](#)
- [Custom roles](#)
- [Propagate users and roles](#)
- [Procedures for native user and role management](#)



Native user and role management is available with Neo4j Enterprise Edition. A subset of this functionality is also available in Community Edition. Refer to [User management for Community Edition](#) for a description of the functionality available in Community Edition.

8.4.1. Native roles

This section describes native roles in Neo4j.

Neo4j provides the following native roles:

reader

- Read-only access to the data graph (all nodes, relationships, properties).

editor

- Read/write access to the data graph.
- Write access limited to creating and changing existing properties key, node labels, and relationship types of the graph.

publisher

- Read/write access to the data graph.

architect

- Read/write access to the data graph.
- Set/delete access to indexes along with any other future schema constructs.

admin

- Read/write access to the data graph.
- Set/delete access to indexes along with any other future schema constructs.
- View/terminate queries.

We detail below the set of actions on the data and database prescribed by each role. The subset of the functionality which is available with Community Edition is also included:

Table 15. 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 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		

Action	reader	editor	publisher	architect	admin	(no role)	Available in Community Edition
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		

A [user](#) who has no assigned roles will not have 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.
- Deleting a user terminates all of the user's running queries and sessions.
- All queries currently running for the deleted user are rolled back.
- 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.

8.4.2. Custom roles

This section describes custom roles in Neo4j.

Custom roles may be [created](#) and [deleted](#) by an [administrator](#). Custom roles are created for the sole purpose of controlling the ability to execute certain custom developed procedures. In contrast to the native roles, a custom role will not have any permissions other than to execute procedures which have been explicitly permitted in [neo4j.conf](#).

More details regarding how to use custom roles to allow for subgraph access control may be found in [Subgraph access control](#).

8.4.3. Propagate users and roles

This section describes how to propagate native users, roles, and role assignments across a Neo4j cluster.

Native users, and user passwords recorded in SHA-256 encrypted format, are stored in a file named `auth`. Roles and role assignments are stored in a file named `roles`. The files are located in the [data](#) directory, in a subdirectory called `dbms`. Neo4j automatically reloads the stored users and assigned roles from disk every five seconds.

Changes to users and roles are applied only to the Neo4j instance on which the commands are

executed. This means that changes are not automatically propagated across a cluster of Neo4j instances, but have to be specifically provided for. A number of options are available to propagate changes to native users, custom roles, and role assignments across a cluster:

- Manually copy users and roles files on disk to all other cluster instances
- Use a shared network folder to store users and roles files
- Create an automated process that synchronizes the stored data across the cluster using, for example, a combination of `rsync` and `crontab`

The recommended solution for clustered security is to use the LDAP or plugin auth provider.

8.4.4. Procedures for native user and role management

This section describes procedures for native user and role management in Neo4j.

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 [User management for 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]	□
<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]	
<code>dbms.showCurrentUser</code>	Show the current user	[reader,editor,publisher,architect,admin]	□

Procedure name	Description	Executable by role(s)	Available in Community Edition
dbms.procedures	List roles per procedure	[reader,editor,publisher,architect,admin]	Not applicable

Activate a suspended user

An [administrator](#) is able to activate a suspended [user](#) so that the user is once again able to access the data in their original capacity.

Syntax:

```
CALL dbms.security.activateUser(username, requirePasswordChange)
```

Arguments:

Name	Type	Description
username	String	This is the username of the user to be activated.
requirePasswordChange	Boolean	This is optional, with a default of true . If this is true , (i) the user will be forced to change their password when they next log in, and (ii) until the user has changed their password, they will be forbidden from performing any other operation.

Exceptions:

The current user is not an administrator.

The username does not exist in the system.

The username matches that of the current user (i.e. activating the current user is not permitted).

Considerations:

This is an idempotent procedure.

Example 36. Activate a suspended user

The following example activates a [user](#) with the username 'jackgreen'. When the user 'jackgreen' next logs in, he will be required to [change his password](#).

```
CALL dbms.security.activateUser('jackgreen')
```

Assign a role to the user

An [administrator](#) is able to assign a [role](#) to any [user](#) in the system, thus allowing the user to perform a series of actions upon the data.

Syntax:

```
CALL dbms.security.addRoleToUser(roleName, username)
```

Arguments:

Name	Type	Description
roleName	String	This is the name of the role to be assigned to the user.
username	String	This is the username of the user who is to be assigned the role.

Exceptions:

The current user is not an administrator.

The username does not exist in the system.

The username contains characters other than alphanumeric characters and the '_' character.

The role name does not exist in the system.

The role name contains characters other than alphanumeric characters and the '_' character.

Considerations:

This is an idempotent procedure.

Example 37. Assign a role to the user

The following example assigns the [role publisher](#) to the user with username '[johnsmith](#)'.

```
CALL dbms.security.addRoleToUser('publisher', 'johnsmith')
```

Change the current user's password

Any [active user](#) is able to change their own password at any time.

Syntax:

```
CALL dbms.security.changePassword(password, requirePasswordChange)
```

Arguments:

Name	Type	Description
password	String	This is the new password for the current user .
requirePasswordChange	Boolean	This is optional, with a default of false . If this is true , (i) the current user will be forced to change their password when they next log in, and (ii) until the current user has changed their password, they will be forbidden from performing any other operation.

Exceptions:

The password is the empty string.

The password is the same as the current user's previous password.

Example 38. Change the current user's password

The following example changes the password of the current user to 'h6u4%kr'.

```
CALL dbms.security.changePassword('h6u4%kr')
```

Change the given user's password

An [administrator](#) is able to change the password of any [user](#) within the system. Alternatively, the [current user](#) may change their own password.

Syntax:

```
CALL dbms.security.changeUserPassword(username, newPassword, requirePasswordChange)
```

Arguments:

Name	Type	Description
username	String	This is the username of the user whose password is to be changed.
newPassword	String	This is the new password for the user.
requirePasswordChange	Boolean	This is optional, with a default of <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 39. Change a given user's password

The following example changes the password of the [user](#) with the username 'joebloggs' to 'h6u4%kr'. When the user 'joebloggs' next logs in, he will be required to [change his password](#).

```
CALL dbms.security.changeUserPassword('joebloggs', 'h6u4%kr')
```

Create a new role

An [administrator](#) is able to create custom roles in the system.

Syntax:

```
CALL dbms.security.createRole(roleName)
```

Arguments:

Name	Type	Description
roleName	String	This is the name of the role to be created.

Exceptions:

The current user is not an administrator.

The role name already exists in the system.

The role name is empty.

The role name contains characters other than alphanumeric characters and the '_' character.

The role name matches one of the native roles: [reader](#), [publisher](#), [architect](#), and [admin](#).

Example 40. Create a new role

The following example creates a new custom role.

```
CALL dbms.security.createRole('operator')
```

Create a new user

An [administrator](#) is able to create a new [user](#). This action ought to be followed by assigning a [role](#) to the user, which is described [here](#).

Syntax:

```
CALL dbms.security.createUser(username, password, requirePasswordChange)
```

Arguments:

Name	Type	Description
username	String	This is the user's username.
password	String	This is the user's password.
requirePasswordChange	Boolean	This is optional, with a default of true . If this is true , (i) the user will be forced to change their password when they log in for the first time, and (ii) until the user has changed their password, they will be forbidden from performing any other operation.

Exceptions:

The current user is not an administrator.

The username either contains characters other than the ASCII characters between ! and ~, or contains : and ,.

The username is already in use within the system.

The password is the empty string.

Example 41. Create a new user

The following example creates a [user](#) with the username 'johnsmith' and password 'h6u4%kr'. When the user 'johnsmith' logs in for the first time, he will be required to [change his password](#).

```
CALL dbms.security.createUser('johnsmith', 'h6u4%kr')
```

Delete the specified role

An [administrator](#) is able to delete custom roles from the system. The native roles [reader](#), [publisher](#), [architect](#), and [admin](#) (see [Native roles](#)) cannot be deleted.

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 42. Delete the specified role

The following example deletes the custom role 'operator' from the system.

```
CALL dbms.security.deleteRole('operator')
```

Delete the specified user

An [administrator](#) is able to delete permanently a [user](#) from the system. It is not possible to undo this action, so, if in any doubt, consider [suspending the user](#) instead.

Syntax:

```
CALL dbms.security.deleteUser(username)
```

Arguments:

Name	Type	Description
username	String	This is the username of the user to be deleted.

Exceptions:

The current user is not an administrator.

The username does not exist in the system.

The username matches that of the current user (i.e. deleting the current user is not permitted).

Considerations:

It is not necessary to remove any assigned [roles](#) from the user prior to deleting the user.

Deleting a user will terminate with immediate effect all of the user's sessions and roll back any running transactions.

As it is not possible for the current user to delete themselves, there will always be at least one administrator in the system.

Example 43. Delete the specified user

The following example deletes a [user](#) with the username 'janebrown'.

```
CALL dbms.security.deleteUser('janebrown')
```

List all available roles

An [administrator](#) is able to view all assigned users for each role in the system.

Syntax:

```
CALL dbms.security.listRoles()
```

Returns:

Name	Type	Description
role	String	This is the name of the role.
users	List<String>	This is a list of the usernames of all users who have been assigned the role.

Exceptions:

The current user is not an administrator.

Example 44. List all available roles

The following example shows, for each role in the system, the name of the role and the usernames of all assigned users.

```
CALL dbms.security.listRoles()
```

```
+-----+  
| role | users |  
+-----+  
| "reader" | ["bill"] |  
| "architect" | [] |  
| "admin" | ["neo4j"] |  
| "publisher" | ["john", "bob"] |  
+-----+  
4 rows
```

List all roles assigned to the specified user

Any [active user](#) is able to view all of their assigned [roles](#). An [administrator](#) is able to view all assigned roles for any [user](#) in the system.

Syntax:

```
CALL dbms.security.listRolesForUser(username)
```

Arguments:

Name	Type	Description
username	String	This is the username of the user.

Returns:

Name	Type	Description
value	String	This returns all roles assigned to the requested user.

Exceptions:

The current user is not an administrator and the username does not match that of the current user.

The username does not exist in the system.

Considerations:

This procedure may be invoked by the current user to view their roles, irrespective of whether or not the current user is an administrator.

This procedure may be invoked by an administrator to view the roles for another user.

Example 45. List all roles assigned to the specified user

The following example lists all the roles for the user with username '`johnsmith`', who has the `roles reader` and `publisher`.

```
CALL dbms.security.listRolesForUser('johnsmith')
```

```
+-----+  
| value |  
+-----+  
| "reader" |  
| "publisher" |  
+-----+  
2 rows
```

List all local users

An `administrator` is able to view the details of every `user` in the system.

Syntax:

```
CALL dbms.security.listUsers()
```

Returns:

Name	Type	Description
<code>username</code>	String	This is the user's username.
<code>roles</code>	List<String>	This is a list of roles assigned to the user.
<code>flags</code>	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 46. List all local users

The following example shows, for each `user` in the system, the username, the `roles` assigned to the user, and whether the user is suspended or needs to change their password.

```
CALL dbms.security.listUsers()
```

```
+-----+-----+  
| username | roles | flags |  
+-----+-----+  
| "neo4j" | ["admin"] | [] |  
| "anne" | [] | ["password_change_required"] |  
| "bill" | ["reader"] | ["is_suspended"] |  
| "john" | ["architect", "publisher"] | [] |  
+-----+-----+  
4 rows
```

List all users currently assigned the specified role

An [administrator](#) is able to view all assigned [users](#) for a [role](#).

Syntax:

```
CALL dbms.security.listUsersForRole(roleName)
```

Arguments:

Name	Type	Description
roleName	String	This is the name of the role.

Returns:

Name	Type	Description
value	String	This returns all assigned users for the requested role.

Exceptions:

The current user is not an administrator.

The role name does not exist in the system.

Example 47. List all users currently assigned the specified role

The following example lists all the assigned users - 'bill' and 'anne' - for the [role publisher](#).

```
CALL dbms.security.listUsersForRole('publisher')
```

```
+-----+
| value   |
+-----+
| "bill"  |
| "anne"  |
+-----+
2 rows
```

Unassign a role from the user

An [administrator](#) is able to remove a [role](#) from any [user](#) in the system, thus preventing the user from performing upon the data any actions prescribed by the role.

Syntax:

```
CALL dbms.security.removeRoleFromUser(roleName, username)
```

Arguments:

Name	Type	Description
roleName	String	This is the name of the role which is to be removed from the user.
username	String	This is the username of the user from which the role is to be removed.

Exceptions:

The current user is not an administrator.
The username does not exist in the system.
The role name does not exist in the system.
The username is that of the current user and the role is admin .

Considerations:

If the username is that of the current user and the role name provided is admin , an error will be thrown; i.e. the current user may not be demoted from being an administrator.
As it is not possible for the current user to remove the admin role from themselves, there will always be at least one administrator in the system.
This is an idempotent procedure.

Example 48. Unassign a role from the user

The following example removes the [role publisher](#) from the user with username 'johnsmith'.

```
CALL dbms.security.removeRoleFromUser('publisher', 'johnsmith')
```

Suspend the specified user

An [administrator](#) is able to suspend a [user](#) from the system. The suspended user may be [activated](#) at a later stage.

Syntax:

```
CALL dbms.security.suspendUser(username)
```

Arguments:

Name	Type	Description
username	String	This is the username of the user to be suspended.

Exceptions:

The current user is not an administrator.
The username does not exist in the system.
The username matches that of the current user (i.e. suspending the current user is not permitted).

Considerations:

Suspending a user will terminate with immediate effect all of the user's sessions and roll back any running transactions.
All of the suspended user's attributes — assigned roles and password — will remain intact.
A suspended user will not be able to log on to the system.
As it is not possible for the current user to suspend themselves, there will always be at least one active administrator in the system.

This is an idempotent procedure.

Example 49. Suspend the specified user

The following example suspends a [user](#) with the username '**billjones**'.

```
CALL dbms.security.suspendUser('billjones')
```

Show the current user

The [current user](#) is able to view whether or not they need to change their password.

Syntax:

```
CALL dbms.showCurrentUser()
```

Returns:

Name	Type	Description
username	String	This is the user's username.
flags	List<String>	This is a flag indicating whether the user needs change their password.

Example 50. Show the current user

The following example shows that the [current user](#) — with the username '**johnsmith**' — does not need to change his password.

```
CALL dbms.showCurrentUser()
```

```
+-----+-----+-----+
| username | roles           | flags |
+-----+-----+-----+
| "johnsmith" | ["architect","publisher"] | []    |
+-----+-----+-----+
1 row
```

List roles per procedure

Any [active user](#) is able to view all procedures in the system, including which role(s) have the privilege to execute them.

Syntax:

```
CALL dbms.procedures()
```

Returns:

Name	Type	Description
name	String	This is the name of the procedure.
signature	String	This is the signature of the procedure.

Name	Type	Description
description	String	This is a description of the procedure.
roles	List<String>	This is a list of roles having the privilege to execute the procedure.

Example 51. List roles per procedure

The following example shows, for four of the security procedures, the procedure name, the description, and which roles have the privilege to execute the procedure.

```
CALL dbms.procedures()
YIELD name, signature, description, roles
WITH name, description, roles
WHERE name contains 'security'
RETURN name, description, roles
ORDER BY name
LIMIT 4
```

name	description	roles
"dbms.security.activateUser"	"Activate a suspended user."	["admin"]
"dbms.security.addRoleToUser"	"Assign a role to the user."	["admin"]
"dbms.security.changePassword"	"Change the current user's password."	
["reader", "editor", "publisher", ...		
"dbms.security.changeUserPassword"	"Change the given user's password."	["admin"]

4 rows

8.5. Integration with LDAP

This section describes Neo4j support for integrating with LDAP systems.

This section describes the following:

- [Introduction](#)
- [Configure the LDAP auth provider](#)
 - Configuration for Active Directory
 - Configuration for openLDAP
- [Use 'ldapsearch' to verify the configuration](#)
- [The auth cache](#)
- [Available methods of encryption](#)
 - Use LDAP with encryption via StartTLS
 - Use LDAP with encrypted LDAPS
- [Use a self-signed certificate in a test environment](#)

8.5.1. Introduction

Neo4j supports the LDAP protocol which allows for integration with Active Directory, OpenLDAP or other LDAP-compatible authentication services. We will show example configurations where management of federated users is deferred to the LDAP service, using that service's facilities for administration. This means that we completely turn off native Neo4j user and role administration and map LDAP groups to the Neo4j [native roles](#), and to custom roles.

8.5.2. Configure the LDAP auth provider

All settings need to be defined at server startup time in the default configuration file `neo4j.conf`. First configure Neo4j to use LDAP as authentication and authorization provider.

```
# Turn on security
dbms.security.auth_enabled=true

# Choose LDAP connector as security provider for both authentication and authorization
dbms.security.auth_provider=ldap
```

Configuration for Active Directory

See below for an example configuration for Active Directory:

```
# Configure LDAP to point to the AD server
dbms.security.ldap.host=ldap://myactivedirectory.example.com

# Provide details on user structure within the LDAP system:
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

# Configure the actual mapping between groups in the LDAP system and roles in Neo4j
dbms.security.ldap.authorization.group_to_role_mapping=
  "cn=Neo4j Read Only,cn=Users,dc=neo4j,dc=com"      = reader      ;\
  "cn=Neo4j Read-Write,cn=Users,dc=neo4j,dc=com"     = publisher    ;\
  "cn=Neo4j Schema Manager,cn=Users,dc=neo4j,dc=com" = architect   ;\
  "cn=Neo4j Administrator,cn=Users,dc=neo4j,dc=com"  = admin       ;\
  "cn=Neo4j Procedures,cn=Users,dc=neo4j,dc=com"     = allowed_role

# In case defined users are not allowed to search for themselves,
# we can specify credentials for a user with read access to all users and groups:
# Note that this account only needs read-only access to the relevant parts of the LDAP directory
# and does not need to have access rights to Neo4j or any other systems.
#dbms.security.ldap.authorization.use_system_account=true
#dbms.security.ldap.authorization.system_username=cn=search-account,cn=Users,dc=example,dc=com
#dbms.security.ldap.authorization.system_password=secret
```

Below is an alternative configuration for Active Directory that allows for logging in with `sAMAccountName`:

```

# Configure LDAP to point to the AD server
dbms.security.ldap.host=ldap://myactivedirectory.example.com

# Provide details on user structure within the LDAP system:
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

# Configure the actual mapping between groups in the LDAP system and roles in Neo4j
dbms.security.ldap.authorization.group_to_role_mapping=
  "cn=Neo4j Read Only,cn=Users,dc=neo4j,dc=com" = reader ;\
  "cn=Neo4j Read-Write,cn=Users,dc=neo4j,dc=com" = publisher ;\
  "cn=Neo4j Schema Manager,cn=Users,dc=neo4j,dc=com" = architect ;\
  "cn=Neo4j Administrator,cn=Users,dc=neo4j,dc=com" = admin ;\
  "cn=Neo4j Procedures,cn=Users,dc=neo4j,dc=com" = allowed_role

# In case defined users are not allowed to search for themselves,
# we can specify credentials for a user with read access to all users and groups:
# Note that this account only needs read-only access to the relevant parts of the LDAP directory
# and does not need to have access rights to Neo4j or any other systems.
dbms.security.ldap.authorization.use_system_account=true
dbms.security.ldap.authorization.system_username=cn=search-account,cn=Users,dc=example,dc=com
dbms.security.ldap.authorization.system_password=secret

# Perform authentication with sAMAccountName instead of DN.
# Using this setting 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.
dbms.security.ldap.authentication.use_samaccountname=true

```

Below is an alternative configuration for Active Directory that allows for authenticating users from different organizational units by using the Active Directory attribute `sAMAccountName`:

```

# Configure LDAP to point to the AD server
dbms.security.ldap.host=ldap://myactivedirectory.example.com

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

# Configure the actual mapping between groups in the LDAP system and roles in Neo4j
dbms.security.ldap.authorization.group_to_role_mapping=
  "cn=Neo4j Read Only,cn=Users,dc=example,dc=com" = reader ;\
  "cn=Neo4j Read-Write,cn=Users,dc=example,dc=com" = publisher ;\
  "cn=Neo4j Schema Manager,cn=Users,dc=example,dc=com" = architect ;\
  "cn=Neo4j Administrator,cn=Users,dc=example,dc=com" = admin ;\
  "cn=Neo4j Procedures,cn=Users,dc=example,dc=com" = allowed_role

```

Specifying the `{0}@example.com` pattern in the `user_dn_template` enables the authentication to start at the root domain. The whole tree is checked to find the user, regardless of where it is located within the tree.

Note that the setting `dbms.security.ldap.authentication.use_samaccountname` is not configured in this example.

Configuration for openLDAP

See below for an example configuration for openLDAP:

```

# Configure LDAP to point to the OpenLDAP server
dbms.security.ldap.host=myopenldap.example.com

# Provide details on user structure within the LDAP system:
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

# Configure the actual mapping between groups in the OpenLDAP system and roles in Neo4j
dbms.security.ldap.authorization.group_to_role_mapping=\
  101 = reader      ;\
  102 = publisher   ;\
  103 = architect   ;\
  104 = admin        ;\
  105 = allowed_role

# In case defined users are not allowed to search for themselves,
# we can specify credentials for a user with read access to all users and groups:
# Note that this account only needs read-only access to the relevant parts of the LDAP directory
# and does not need to have access rights to Neo4j or any other systems.
#dbms.security.ldap.authorization.use_system_account=true
#dbms.security.ldap.authorization.system_username=cn=search-account,ou=users,dc=example,dc=com
#dbms.security.ldap.authorization.system_password=search-account-password

```

We would like to draw attention to some details in the configuration examples. A comprehensive overview of LDAP configuration options is available in [Configuration settings](#).

Parameter name	Default value	Description
dbms.security.ldap.authentication.user_dn_template	uid={0},ou=users,dc=example,dc=com	Converts usernames into LDAP-specific fully qualified names required for logging in.
dbms.security.ldap.authorization.user_search_base	ou=users,dc=example,dc=com	Sets the base object or named context to search for user objects.
dbms.security.ldap.authorization.user_search_filter	(&(objectClass*)(uid={0}))	Sets up an LDAP search filter to search for a user principal.
dbms.security.ldap.authorization.group_membership_attributes	[memberOf]	Lists attribute names on a user object that contains groups to be used for mapping to roles.
dbms.security.ldap.authorization.group_to_role_mapping		Lists an authorization mapping from groups to the pre-defined built-in roles <code>admin</code> , <code>architect</code> , <code>publisher</code> and <code>reader</code> , or to any other custom-defined roles.

8.5.3. Use 'ldapsearch' to verify the configuration

We can use the LDAP command-line tool `ldapsearch` to verify that the configuration is correct, and that the LDAP server is actually responding. We do this by issuing a search command that includes LDAP configuration setting values.

These example searches verify both the authentication (using the `simple` mechanism) and authorization of user 'john'. See the `ldapsearch` documentation for more advanced usage and how to use SASL authentication mechanisms.

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 secret -b cn=Users,dc=example,dc=com "(&(objectClass=*)(cn=john))" memberOf
```

Then verify that we get a successful response, and 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
```

8.5.4. 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 `dbms.security.ldap.authentication.cache_enabled` and `dbms.security.auth_cache_ttl` parameters.

```
# Turn on authentication caching to ensure performance
dbms.security.ldap.authentication.cache_enabled=true
dbms.security.auth_cache_ttl=10m
```

Parameter name	Default value	Description
<code>dbms.security.ldap.authentication.cache_enabled</code>	<code>true</code>	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. It should be noted that when using the REST API, disabling authentication caching will result in re-authentication and possibly re-authorization of users on every request, which may severely impact performance on production systems, and put heavy load on the LDAP server.

Parameter name	Default value	Description
<code>dbms.security.auth_cache_ttl</code>	<code>10000 minutes</code>	Is the time to live (TTL) for cached authentication and authorization info. Setting the TTL to 0 will disable all auth caching. A short ttl will require more frequent re-authentication and re-authorization, which can impact performance. A very long ttl will also mean that changes to the users settings on an LDAP server may not be reflected in the Neo4j authorization behavior in a timely manner.

An administrator can clear the auth cache to force the re-querying of authentication and authorization information from the federated auth provider system.

Example 52. Clear the auth cache

Use Neo4j Browser or Neo4j Cypher Shell to execute this statement.

```
CALL dbms.security.clearAuthCache()
```

8.5.5. Available methods of encryption

All the following ways of specifying the `dbms.security.ldap.host` parameter are valid. Doing so will configure using LDAP without encryption. Not specifying the protocol or port will result 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. Not specifying the port will result in `ldaps` being used over the default port 636.

```
dbms.security.ldap.host=ldaps://myactivedirectory.example.com
dbms.security.ldap.host=ldaps://myactivedirectory.example.com:636
```

This method of securing Active Directory is being deprecated and is therefore not recommended. Instead, use Active Directory with encryption via [StartTLS](#).

8.5.6. Use a self-signed certificate 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 wish to use a self-signed certificate on the LDAP server. In these scenarios you will have to tell Neo4j about the local certificate. This is done by entering the details of the certificate using `dbms.jvm.additional` in `neo4j.conf`.

Example 53. Specify details for self-signed certificate on LDAP server

This example shows how to specify details for a self-signed certificate on an LDAP server. The path to the certificate file `MyCert.jks` is an absolute path on the Neo4j server.

```
dbms.jvm.additional=-Djavax.net.ssl.keyStore=/path/to/MyCert.jks  
dbms.jvm.additional=-Djavax.net.ssl.keyStorePassword=secret  
dbms.jvm.additional=-Djavax.net.ssl.trustStore=/path/to/MyCert.jks  
dbms.jvm.additional=-Djavax.net.ssl.trustStorePassword=secret
```

8.6. Subgraph access control

This section describes how to configure subgraph access control in Neo4j.

This section describes the following:

- [Introduction](#)
- [Configuration steps](#)
 - [Create a custom role](#)
 - [Manage procedure permissions](#)

8.6.1. Introduction

It is possible to restrict a user's access to, and subsequent actions on, specified portions of the graph. For example, a user can be allowed to read, but not write, nodes with specific labels and relationships with certain types.

To implement subgraph access control, you must complete the following steps:

1. Put a procedure or function in place that performs the reads from, and/or writes to, the portion of the graph that you wish to control. This can either be developed in-house, or be made available as a third-party library. Please refer to [Java Reference](#) [Extending Neo4j](#) for a description on creating and using user-defined procedures and functions.
2. Create one, or several, custom roles, with which to run the procedure described above. These roles can subsequently be assigned the relevant privileges.
3. Configure the procedure so that it can be executed by users with the custom roles.

The steps below assume that the procedure or function is already developed and installed.

8.6.2. Configuration steps

Create a custom role

Create a custom role and manage it either through native user management or through federated user management with LDAP.

Native users scenario

In the native users scenario, a custom role is created and assigned to the relevant user(s).

Example 54. Native users scenario

In this example, we will use Cypher to create a custom `accounting` role and assign it to a pre-existing user, `billsmith`.

```
CALL dbms.security.createRole('accounting')
CALL dbms.security.addRoleToUser('accounting', 'billsmith')
```

Federated users scenario (LDAP)

In the LDAP scenario, the LDAP user group is mapped to a custom role in Neo4j.

Example 55. Federated users scenario (LDAP)

In this example, we will use Cypher to create a custom `accounting` role.

```
CALL dbms.security.createRole('accounting')
```

We will then map the `accounting` role to the LDAP group with groupID `101`.

```
dbms.security.realms.ldap.authorization.group_to_role_mapping=101=accounting
```

Manage procedure permissions

In standard use, procedures and functions are executed according to the same security rules as regular Cypher statements, as described in [Native roles](#). For example, users assigned any one of the native roles `publisher`, `architect` and `admin` will be able to execute a procedure with `mode=WRITE`, whereas a user assigned only the `reader` role will not be allowed to execute the procedure.

For the purpose of subgraph access control, we allow specific roles to execute procedures that they would otherwise be prevented from accessing through their assigned native roles. The user is given the privilege that comes with the mode of the procedure, during the execution of the procedure only. The following two parameters are used to configure the desired behavior:

`dbms.security.procedures.default_allowed`

The setting `dbms.security.procedures.default_allowed` defines a single role that is allowed to execute any procedure or function that is not matched by the `dbms.security.procedures.roles` configuration.

Example 56. Configure a default role that can execute procedures and functions

Assume that we have the following configuration:

```
dbms.security.procedures.default_allowed=superAdmin
```

This will have the following effects:

- If the setting `dbms.security.procedures.roles` is left unconfigured, the role `superAdmin` will be able to execute all custom procedures and functions.
- If the setting `dbms.security.procedures.roles` has some roles and functions defined, the role `superAdmin` will be able to execute all custom procedures and functions that are *not* configured by `dbms.security.procedures.roles`.

`dbms.security.procedures.roles`

The `dbms.security.procedures.roles` setting provides fine-grained control over procedures.

Example 57. Configure roles for the execution of specific procedures

Assume that we have the following configuration:

```
dbms.security.procedures.roles=apoc.convert.*:Converter;apoc.load.json.*:Converter,DataSource;apoc.trigger.add:TriggerHappy
```

This will have the following effects:

- All users with the role `Converter` will be able to execute all procedures in the `apoc.convert` namespace.
- All users with the roles `Converter` and `DataSource` will be able to execute procedures in the `apoc.load.json` namespace.
- All users with the role `TriggerHappy` will be able to execute the specific procedure `apoc.trigger.add`.



A procedure will fail if it attempts to execute database operations that violates its mode. For example, a procedure assigned the mode of `READ` will fail if it is programmed to do write actions. This will happen regardless of user or role configuration.

8.7. Property-level access control

This section describes how to configure property-level access control in Neo4j.

This section describes the following:

- [Introduction](#)
- [Implement a property blacklist](#)

8.7.1. Introduction

You can use role-based, database-wide, property blacklists to limit which properties a user can read.

The table below lists the configuration parameters that control this feature:

Parameter name	Default value	Description
<code>dbms.security.property_level.enabled</code>	<code>false</code>	Enable property level access control.
<code>dbms.security.property_level.blacklist</code>		An authorization mapping for property level access for roles. The map should be formatted as a semicolon-separated list of key-value pairs, where the key is the role name and the value is a comma-separated list of blacklisted properties. The blacklisted properties for a given user is the union of the blacklist for all the roles that user is part of.

Considerations

The property blacklist prevents users from reading properties. A user querying for a blacklisted property will get the same results as if the property did not exist on the node/relationship.

Blacklisting a property will only affect the reading of that property, not the writing. It is therefore recommended to only add users that are assigned the `reader` role to roles that have a property blacklist.

Limitations

All properties with a name corresponding to the ones in the blacklist will be affected. This is regardless of whether it is associated with a node or a relationship, and regardless of node labels and relationship types.

8.7.2. Implement a property blacklist

To enable this feature, the following steps must be taken:

1. Enable property-level access control through the setting `dbms.security.property_level.enabled`.
2. Configure the setting `dbms.security.property_level.blacklist` to restrict specific roles from reading the named properties.
3. Create one, or several, custom roles, which will have restricted property access.
4. Add user to appropriate roles.

Example 58. Implement a property blacklist

First, we enable property-level access control and create the blacklist:

```
dbms.security.property_level.enabled=true
dbms.security.property_level.blacklist=\
roleX=propertyA; \
roleY=propertyB,propertyC
```

Then, we create the custom roles and assign users to them:

```
CALL dbms.security.createRole('roleX')
CALL dbms.security.createRole('roleY')
CALL dbms.security.addRoleToUser('roleX', 'user-1')
CALL dbms.security.addRoleToUser('roleY', 'user-2')
CALL dbms.security.addRoleToUser('roleX', 'user-3')
CALL dbms.security.addRoleToUser('roleY', 'user-3')
```

This will have the following effects:

- The user **user-1** will be unable to read the property **propertyA**.
- The user **user-2** will be unable to read the properties **propertyB** and **propertyC**.
- The user **user-3** will be unable to read the properties **propertyA**, **propertyB** and **propertyC**.

Chapter 9. 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](#).

9.1. Securing extensions

This section describes how 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](#). This section describes how to ensure the security of these additions.

9.1.1. Sandboxing

Neo4j provides sandboxing to ensure that procedures do not inadvertently use insecure APIs. For example, when writing custom code it is possible to access Neo4j APIs that are not publicly supported, and these internal APIs are subject to change, without notice. Additionally, their use comes with the risk of performing insecure actions. The sandboxing functionality limits the use of extensions to publicly supported APIs, which exclusively contain safe operations, or contain security checks.

The configuration setting `dbms.security.procedures.unrestricted` provides the possibility to circumvent the sandboxing functionality by defining a comma-separated list of procedures and/or user-defined functions that are allowed full access to the database. The list may contain fully-qualified procedure names, and partial names with the wildcard `*`.

Any attempt to load an extension which contains an unsupported API, which has not been marked as allowed in `dbms.security.procedures.unrestricted`, will result in a warning in the security log. The warning will point out that the extension does not have access to the components it is trying to load. Additionally, a mocked procedure will be loaded with the procedure's name. Calling the mocked procedure will result in an error, saying that the procedure failed to load due to needing more permissions.

Example 59. Sandboxing

In this example we assume that the procedure `my.extensions.example`, as well as some procedures in the `my.procedures` library, make use of unsupported APIs. In order to allow the running of these procedures we configure the setting as shown below.

```
# Example sandboxing
dbms.security.procedures.unrestricted= my.extensions.example,my.procedures.*
```

9.1.2. White listing

White listing can be used to allow loading only a few extensions from a larger library.

The configuration setting `dbms.security.procedures.whitelist` 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 60. 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.whitelist=apoc.coll.*,apoc.load.*
```

There are a few things that should be noted about `dbms.security.procedures.whitelist`:

- 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.
- If the extensions pointed out by this parameter are programmed to access internal APIs, they also have to be explicitly allowed, as described in [Sandboxing](#).

9.2. SSL framework

This section describes SSL/TLS integration for securing communication channels in Neo4j.

Neo4j supports the securing of communication channels using standard SSL/TLS technology.

This section describes the following:

- [Introduction](#)
- [Prerequisites](#)
 - [Java configuration](#)
 - [Certificates](#)
- [Define SSL policies](#)

- [Apply SSL policies](#)
- [Choosing an SSL provider](#)
- [The legacy SSL system](#)
- [Terminology](#)

9.2.1. Introduction

SSL support is enabled by creating an SSL policy, which consists of an SSL certificate together with a set of parameters. The policy is then applied to the communication channel(s) that you wish to secure. The process is described in the following sections.



If Neo4j is started without any SSL policy definitions, it will default to the *legacy SSL system*. Additionally, if no certificates are installed, the Neo4j process will automatically generate a self-signed SSL certificate and a private key in a default directory. For details, refer to [The legacy SSL system](#).

9.2.2. Prerequisites

Java configuration

The Neo4j SSL framework uses strong cryptography which may need to be specifically enabled on your Java platform. For example, see:

<http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html>

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. Such certificates are required if Neo4j has been configured with multiple connectors that bind to different interfaces.

9.2.3. Define SSL policies

SSL policies are configured by assigning values to parameters of the following format:

`dbms.ssl.policy.<policy-name>.<setting-suffix>`

The [policy-name](#) is the name of the policy that you wish to define.

The basic valid values for [setting-suffix](#) are described below:

Setting suffix	Description	Default value
<code>base_directory</code>	The base directory under which cryptographic objects are searched for by default.	This is a mandatory setting.
<code>private_key</code>	The private key used for authenticating and securing this instance.	<code>private.key</code>
<code>public_certificate</code>	A public certificate matching the private key signed by a Certificate Authority (CA).	<code>public.crt</code>

Setting suffix	Description	Default value
<code>trusted_dir</code>	A directory populated with certificates of trusted parties. If configured, this must be an absolute path. At the very minimum, this must contain this node's public certificate (<code>public.crt</code>).	<code>trusted</code>
<code>revoked_dir</code>	A directory populated with certificate revocation lists (CRLs). If configured, this must be an absolute path.	<code>revoked</code>



The public certificate must be duplicated. One copy is to be placed into the `base_directory`, and the other copy is to be placed into the `trusted_dir`.

The only mandatory setting is the base directory defined by `dbms.ssl.policy.<policy-name>.base_directory`. By defining the base directory, we tell Neo4j to define a policy with the name `<policy-name>`. If no other settings for this policy have been defined, Neo4j will by default be looking for the private key and the certificate file inside the base directory, as well as two subdirectories called `trusted` and `revoked`. If other paths are preferred, all the default values can be overridden.

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 61. Define a policy

In this example we will define and create configuration for a policy called `example_policy`. As the simplest configuration possible, we define the base directory for this policy in `neo4j.conf`:

```
dbms.ssl.policy.example_policy.base_directory=certificates/example_policy
```

Then create the mandatory directories:

```
$neo4j-home> mkdir certificates/example_policy
$neo4j-home> mkdir certificates/example_policy/trusted
$neo4j-home> mkdir certificates/example_policy/revoked
```

Finally, place the files `private.key` and `public.crt` into the base directory, and place another copy of the `public.crt` into the `trusted` directory:

```
$neo4j-home> cp /path/to/certs/private.key certificates/example_policy
$neo4j-home> cp /path/to/certs/public.crt certificates/example_policy
$neo4j-home> cp /path/to/certs/public.crt certificates/example_policy/trusted
```

We will have the following listings:

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

Additionally, the following parameters can be configured for a policy:

Setting suffix	Description	Default
<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>REQUIRED</code>
<code>ciphers</code>	A list of ciphers which will be allowed during cipher negotiation.	Java platform default allowed cipher suites
<code>tls_versions</code>	A list of TLS/SSL protocol versions which will be supported.	<code>TLSv1.2</code>
<code>allow_key_generation</code>	It is <i>strongly recommended</i> to keep this parameter at its default value of <code>false</code> . If set to <code>true</code> , it will enable the auto-generation of a <code>.key/.crt</code> file pair on startup. Additionally, the required directory structure will be generated automatically.	<code>false</code>
<code>trust_all</code>	It is <i>strongly recommended</i> to keep this parameter at its default value of <code>false</code> . Setting it to <code>true</code> means "trust anyone" and essentially disables authentication.	<code>false</code>
<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>

The combination of Neo4j and the Java platform will provide strong cipher suites and protocols.

9.2.4. Apply SSL policies

The different [communication channels](#) can be secured independently from each other, using the configuration settings below:

`bolt.ssl_policy`

The policy to be used for Bolt client traffic.

`https.ssl_policy`

The policy to be used for HTTPS client traffic.

`causal_clustering.ssl_policy`

The policy to be used for intra-cluster communication.

`dbms.backup.ssl_policy`

The policy to be used for encrypting backup traffic.

Example 62. Apply an SSL policy

Assume that we have configured the policies listed below:

- One policy called `client_policy` for encryption of client traffic.
- One policy called `cluster_policy` for intra-cluster encryption.
- One policy called `backup_policy` for encrypting backup traffic.

The following example will configure the encryption accordingly:

```
bolt.ssl_policy=client_policy
https.ssl_policy=client_policy
causal_clustering.ssl_policy=cluster_policy
dbms.backup.ssl_policy=backup_policy
```

9.2.5. 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`.

9.2.6. The legacy SSL system

SSL support can also be provided for Bolt and HTTPS using the *legacy SSL system*. The legacy system is expected to be deprecated at some point in the future, so it is recommended to use the [standard SSL configuration](#) instead.

In order to configure the legacy SSL system with Neo4j, the private key and certificate files must be named `neo4j.key` and `neo4j.cert`, respectively. Note that the key should be unencrypted. Place the files into the assigned directory. The default is a directory named `certificates`, which is located in the `neo4j-home` directory. The directory can also be configured explicitly using `dbms.directories.certificates` in `neo4j.conf`.

If started without any certificates installed, the Neo4j process will automatically generate a self-signed SSL certificate and a private key in the default directory. Using auto-generation of self-signed SSL certificates will not work if Neo4j has been configured with multiple `connectors` that bind to different IP addresses. If you need to use multiple IP addresses, please configure certificates manually and use multi-host or wildcard certificates instead.

The legacy policy is available in the SSL framework under the special `legacy` policy name, but it does not allow the full flexibility of the framework. It is essentially equivalent to the following SSL policy definition:

```
bolt.ssl_policy=legacy
https.ssl_policy=legacy

dbms.ssl.policy.legacy.base_directory=certificates
dbms.ssl.policy.legacy.private_key=neo4j.key
dbms.ssl.policy.legacy.public_certificate=neo4j.cert
dbms.ssl.policy.legacy.allow_key_generation=true
dbms.ssl.policy.legacy.client_auth=NONE
```

The HTTPS and Bolt servers do not support client authentication (a.k.a. *mutual authentication*). As a result, `client_auth` has to be turned off explicitly by having `client_auth=NONE` while migrating HTTPS and Bolt servers to the new ssl policy. When client authentication is disabled, values assigned to `trusted_dir`, `revoked_dir` or `trust_all` will be ignored as they are settings used in client authentication.

The `tls_versions` and `ciphers` settings are supported in HTTPS and Bolt servers. The `legacy` policy defaults to the TLS versions and cipher suites supported by the Java platform.

9.2.7. 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 simply 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 version

A version of the [TLS protocol](#).

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".

9.3. 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.

9.4. 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.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 Neo4j 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.whitelist` 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. Do not turn off Neo4j authentication. Refer to [Configuration](#) for details on this setting.
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 10. 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](#)
 - Security events logging
 - Query 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
- [Monitoring a Causal Cluster](#)
 - Procedures for monitoring a Causal Cluster
 - Endpoints for status information
 - Monitoring a multi-cluster

10.1. Metrics

This section describes the Neo4j metrics output facilities.

This section describes the following:

- [Expose metrics](#)
 - [Enable metrics logging](#)
 - [Graphite](#)
 - [Prometheus](#)
 - [CSV files](#)
- [Metrics reference](#)
 - [General-purpose metrics](#)
 - [Metrics specific to Causal Clustering](#)
 - [Java Virtual Machine metrics](#)

10.1.1. 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](#)

Enable metrics logging

By default, metrics logging is enabled and configured to report metrics into CSV files.

Neo4j can expose metrics for the following parts of the database:

```
# Setting for enabling all supported metrics.  
metrics.enabled=true  
  
# Setting for enabling all Neo4j-specific metrics.  
metrics.neo4j.enabled=true  
  
# Setting for exposing metrics about transactions; number of transactions started, committed, etc.  
metrics.neo4j.tx.enabled=true  
  
# Setting for exposing metrics about the Neo4j page cache; page faults, evictions, flushes and exceptions, etc.  
metrics.neo4j.pagecache.enabled=true  
  
# Setting for exposing metrics about approximately entities are in the database; nodes, relationships, properties, etc.  
metrics.neo4j.counts.enabled=true  
  
# Setting for exposing metrics about the network usage of the HA cluster component.  
metrics.neo4j.network.enabled=true
```

Graphite

Send metrics to [Graphite](https://graphiteapp.org/) (<https://graphiteapp.org/>) 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 3 seconds.
metrics.graphite.interval=3s
# 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** (<https://prometheus.io/>) 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 3 seconds.
metrics.csv.interval=3s
# The maximum number of CSV files that will be saved. Default is 7.
metrics.csv.rotation.keep_number
# The file size at which the csv files will auto-rotate. Default is 10M.
metrics.csv.rotation.size=10M
```

10.1.2. 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 16. Database checkpointing metrics

Name	Description
neo4j.check_point.events	The total number of check point events executed so far
neo4j.check_point.total_time	The total time spent in check pointing so far
neo4j.check_point.check_point_duration	The duration of the check point event

Table 17. Database data metrics

Name	Description
neo4j.ids_in_use.relationship_type	The total number of different relationship types stored in the database
neo4j.ids_in_use.property	The total number of different property names used in the database
neo4j.ids_in_use.relationship	The total number of relationships stored in the database
neo4j.ids_in_use.node	The total number of nodes stored in the database

Table 18. Database page cache metrics

Name	Description
neo4j.page_cache.eviction_exceptions	The total number of exceptions seen during the eviction process in the page cache
neo4j.page_cache.flushes	The total number of flushes executed by the page cache
neo4j.page_cache.unpins	The total number of page unpins executed by the page cache
neo4j.page_cache.pins	The total number of page pins executed by the page cache
neo4j.page_cache.evictions	The total number of page evictions executed by the page cache
neo4j.page_cache.page_faults	The total number of page faults happened in the page cache
neo4j.page_cache.hits	The total number of page hits happened in the page cache
neo4j.page_cache.hit_ratio	The ratio of hits to the total number of lookups in the page cache
neo4j.page_cache.usage_ratio	The ratio of number of used pages to total number of available pages

Table 19. Database transaction metrics

Name	Description
neo4j.transaction.started	The total number of started transactions
neo4j.transaction.peak_concurrent	The highest peak of concurrent transactions ever seen on this machine

Name	Description
neo4j.transaction.active	The number of currently active transactions
neo4j.transaction.active_read	The number of currently active read transactions
neo4j.transaction.active_write	The number of currently active write transactions
neo4j.transaction.committed	The total number of committed transactions
neo4j.transaction.committed_read	The total number of committed read transactions
neo4j.transaction.committed_write	The total number of committed write transactions
neo4j.transaction.rolledbacks	The total number of rolled back transactions
neo4j.transaction.rolledbacks_read	The total number of rolled back read transactions
neo4j.transaction.rolledbacks_write	The total number of rolled back write transactions
neo4j.transaction.terminated	The total number of terminated transactions
neo4j.transaction.terminated_read	The total number of terminated read transactions
neo4j.transaction.terminated_write	The total number of terminated write transactions
neo4j.transaction.last_committed_tx_id	The ID of the last committed transaction
neo4j.transaction.last_closed_tx_id	The ID of the last closed transaction

Table 20. Cypher metrics

Name	Description
neo4j.cypher.replan_events	The total number of times Cypher has decided to re-plan a query
neo4j.cypher.replan_wait_time	The total number of seconds waited between query replans

Table 21. Database log rotation metrics

Name	Description
neo4j.log_rotation.events	The total number of transaction log rotations executed so far
neo4j.log_rotation.total_time	The total time spent in rotating transaction logs so far
neo4j.log_rotation.log_rotation_duration	The duration of the log rotation event

Table 22. Bolt metrics

Name	Description
neo4j.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).
neo4j.bolt.connections_opened	The total number of Bolt connections opened since this instance started. This includes both succeeded and failed connections.

Name	Description
neo4j.bolt.connections_closed	The total number of Bolt connections closed since this instance started. This includes both properly and abnormally ended connections.
neo4j.bolt.connections_running	The total number of Bolt connections currently being executed.
neo4j.bolt.connections_idle	The total number of Bolt connections sitting idle.
neo4j.bolt.messages_received	The total number of messages received via Bolt since this instance started.
neo4j.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.
neo4j.bolt.messages_done	The total number of messages that completed processing since this instance started. This includes successful, failed and ignored Bolt messages.
neo4j.bolt.messages_failed	The total number of messages that failed processing since this instance started.
neo4j.bolt.accumulate_d_queue_time	The accumulated time messages have spent waiting for a worker thread.
neo4j.bolt.accumulate_d_processing_time	The accumulated time worker threads have spent processing messages.

Table 23. Server metrics

Name	Description
neo4j.server.threads.jetty.idle	The total number of idle threads in the jetty pool
neo4j.server.threads.jetty.all	The total number of threads (both idle and busy) in the jetty pool

Metrics specific to Causal Clustering

Table 24. Core metrics

Name	Description
neo4j.causal_clustering.core.append_index	Append index of the RAFT log
neo4j.causal_clustering.core.commit_index	Commit index of the RAFT log
neo4j.causal_clustering.core.term	RAFT Term of this server
neo4j.causal_clustering.core.tx_retries	Transaction retries
neo4j.causal_clustering.core.is_leader	Is this server the leader?
neo4j.causal_clustering.core.in_flight_cache.total_bytes	In-flight cache total bytes
neo4j.causal_clustering.core.in_flight_cache.max_bytes	In-flight cache max bytes
neo4j.causal_clustering.core.in_flight_cache.element_count	In-flight cache element count
neo4j.causal_clustering.core.in_flight_cache.max_elements	In-flight cache maximum elements

Name	Description
neo4j.causal_clustering.core.in_flight_cache_hits	In-flight cache hits
neo4j.causal_clustering.core.in_flight_cache_misses	In-flight cache misses
neo4j.causal_clustering.core.message_processing_delay	Delay between RAFT message receive and process
neo4j.causal_clustering.core.message_processing_timer	Timer for RAFT message processing
neo4j.causal_clustering.core.replication_new	Raft replication new request count
neo4j.causal_clustering.core.replication_attempt	Raft replication attempt count
neo4j.causal_clustering.core.replication_success	Raft Replication success count
neo4j.causal_clustering.core.replication_fail	Raft Replication fail count

Table 25. Read Replica Metrics

Name	Description
neo4j.causal_clustering.read_replica.pull_updates	The total number of pull requests made by this instance
neo4j.causal_clustering.read_replica.pull_update_highest_tx_id_requested	The highest transaction id requested in a pull update by this instance
neo4j.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

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.

10.2. Logging

This section describes security and query logging in Neo4j.

Neo4j provides two types of logs for inspection of queries that are run in the database, and of security events that have occurred.

The section describes the following:

- Query logging

- [Security events logging](#)

10.2.1. Query logging

This section describes Neo4j support for query logging.

Neo4j can be configured to log queries executed in the database.

Query logging must be enabled by setting the `dbms.logs.query.enabled` parameter to `true`. The parameter `dbms.logs.query.threshold` determines the threshold for logging a query. If the execution of a query takes a longer time than this threshold, it will be logged. Setting `dbms.logs.query.threshold` to `0` will result in all queries being logged.

Log configuration

The name of the log file is `query.log` and it resides in the `logs` directory (see [File locations](#)).

Rotation of the query log can be configured in the `neo4j.conf` configuration file. The following parameters are available:

Parameter name	Default value	Description
<code>dbms.logs.query.allocation_logging_enabled</code>	<code>false</code>	Log allocated bytes for the executed queries being logged.
<code>dbms.logs.query.enabled</code>	<code>false</code>	Log executed queries that take longer than the configured threshold, <code>dbms.logs.query.threshold</code> .
<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_logging_enabled</code>	<code>true</code>	Log parameters for executed queries that take longer than the configured threshold.
<code>dbms.logs.query.rotation.keep_number</code>	<code>7</code>	Sets number of historical log files kept.
<code>dbms.logs.query.rotation.size</code>	<code>20M</code>	Sets the file size at which the query log will auto-rotate.
<code>dbms.logs.query.threshold</code>	<code>0</code>	If the execution of query takes a longer time than this threshold, the query is logged (provided query logging is enabled).
<code>dbms.logs.query.time_logging_enabled</code>	<code>false</code>	Log detailed time information for the executed queries being logged.

Example 63. Configure for simple query logging

In this example we turn query logging on, but leave all other query log parameters at their defaults.

```
dbms.logs.query.enabled=true
```

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 64. 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 `dbms.setTXMetaData`. This is typically done programmatically, but can be illustrated as follows, using `cypher-shell`.

Example 65. Attach metadata to a query

Start a transaction and call `dbms.setTXMetaData` with a list of meta data.

```
neo4j> :begin
neo4j# CALL dbms.setTXMetaData({ 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 dbms.setTXMetaData({ User: 'jsmith', AppServer: 'app03.dc01.company.com'}); - {} - {}
... CALL dbms.procedures() YIELD name RETURN COUNT(name); - {} - {User: 'jsmith', AppServer:
'app03.dc01.company.com'}
```

10.2.2. Security events logging

This section describes Neo4j support for 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.
- Change of password for user, by administrator and by the user themselves.
- Creation and deletion of users, including failed attempts.
- Creation and deletion of custom roles, including failed attempts.
- Assignment and removal of roles for users, including failed attempts.
- Suspension and activation of users.
- Failed attempts by non-admin users to list users and roles.

Log configuration

The name of the log file is `security.log` and it resides in the `logs` directory (see [File locations](#)).

Rotation of the security events log can be configured in the `neo4j.conf` configuration file. The following parameters are available:

Parameter name	Default value	Description
<code>dbms.logs.security.rotation.size</code>	20M	Sets the file size at which the security event log will auto-rotate.
<code>dbms.logs.security.rotation.delay</code>	300s	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>	7	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, for example using REST, 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:

```
2016-10-27 13:45:00.796+0000 INFO [AsyncLog @ 2016-10-27 ...] [johnsmith]: logged in
2016-10-27 13:47:53.443+0000 ERROR [AsyncLog @ 2016-10-27 ...] [johndoe]: failed to log in: invalid
principal or credentials
2016-10-27 13:48:28.566+0000 INFO [AsyncLog @ 2016-10-27 ...] [johnsmith]: created user `janedoe`
2016-10-27 13:48:32.753+0000 ERROR [AsyncLog @ 2016-10-27 ...] [johnsmith]: tried to create user
`janedoe`: The specified user ...
2016-10-27 13:49:11.880+0000 INFO [AsyncLog @ 2016-10-27 ...] [johnsmith]: added role `admin` to user
`janedoe`
2016-10-27 13:49:34.979+0000 INFO [AsyncLog @ 2016-10-27 ...] [johnsmith]: deleted user `janedoe`
2016-10-27 13:49:37.053+0000 ERROR [AsyncLog @ 2016-10-27 ...] [johnsmith]: tried to delete user
`janedoe`: User 'janedoe' does ...
2016-10-27 14:00:02.050+0000 INFO [AsyncLog @ 2016-10-27 ...] [johnsmith]: created role `operator`
```

10.3. 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](#)

10.3.1. 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
<code>queryId</code>	String	This is the ID of the query.
<code>username</code>	String	This is the username of the user who is executing the query.
<code>metaData</code>	Map	This is any metadata associated with the transaction.
<code>query</code>	String	This is the query itself.
<code>parameters</code>	Map	This is a map containing all the parameters used by the query.
<code>planner</code>	String	Planner used by the query
<code>runtime</code>	String	Runtime used by the query

Name	Type	Description
indexes	List	Indexes used by the query
startTime	String	This is the time at which the query was started.
elapsedTime	String	Deprecated: Use <code>elapsedTimeMillis</code> instead. This is the time that has elapsed since the query was started.
connectionDetails	String	Deprecated: Use <code>requestScheme</code> , <code>clientAddress</code> , <code>requestUri</code> . 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.
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 <code>true</code> .
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 <code>true</code> .
allocatedBytes	Integer	Bytes allocated for the executing query. This number is cumulative over the duration of the query. For memory-intense or long-running queries the value may be larger than the current memory allocation. This field will be null unless the config parameter <code>dbms.track_query_allocation</code> is set to <code>true</code> .
pageHits	Integer	Page hits occurred during the execution.
pageFaults	Integer	Page faults occurred during the execution.

Example 66. 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`, and `status`.

```
CALL dbms.listQueries() YIELD queryId, username, query, elapsedTimeMillis, requestUri, status
```

```
"queryId"  "username"  "query"          "elapsedTimeMillis"  "requestUri"      "status"
```

```
"query-33"  "alwood"    "CALL dbms.listQueries() YIELD "1"  
"127.0.0.1:7687" {"state":"RUNNING"}  
           queryId, username, query, ela  
           psedTime, requestUri, status"
```

```
1 row
```

10.3.2. 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 67. 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  
CALL dbms.listActiveLocks( queryId ) YIELD resourceType, resourceId, mode  
RETURN queryId, query, resourceType, resourceId, mode
```

```
"queryId"    "query"          "resourceType" "resourceId" "mode"  
"query-614"  "match (n), (m), (o), (p), (q) "SCHEMA"     "0"        "SHARED"  
             "return count(*)"  
"query-684"  "CALL dbms.listQueries() YIELD "SCHEMA"   "0"        "SHARED"  
             .."
```

2 rows

10.3.3. 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.

Example 68. 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.

```
CALL dbms.killQueries(['query-378', 'query-765'])
```

```
+-----+  
| queryId | username |  
+-----+  
| "query-378" | "joesmith" |  
| "query-765" | "annebrown" |  
+-----+  
2 rows
```

10.3.4. 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
id	String	This is the ID of the query to be terminated.

Returns:

Name	Type	Description
queryId	String	This is the ID of the terminated query.
username	String	This is the username of the user who was executing the (now terminated) query.
message	String	A message stating whether the query was successfully found.

Example 69. 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

10.4. 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](#)

10.4.1. 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 70. 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*.

10.4.2. 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 71. Configure lock acquisition timeout

Set the timeout to ten seconds.

```
dbms.lock.acquisition.timeout=10s
```

10.4.3. 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.
<code>activeLockCount</code>	Integer	Count of active locks held by transaction.

Name	Type	Description
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	Bytes allocated for the executing transaction. This number is cumulative over the duration of the transaction. For memory-intense or long-running transactions the value may be larger than the current memory allocation.
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.

Example 72. 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"

"transaction-22" "alwood"    "CALL dbms.listTransactions() YIELD      "1"
"127.0.0.1:7687" "Running"   transactionId, username, currentQuery
                                elapsedTime, requestUri, status"
```

1 row

10.5. 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](#)

10.5.1. 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 73. List all network connections

The following example shows that the user '**alwood**' is connected using Java driver and a Firefox web browser. The procedure call yields specific information about the connection, namely **connectionId**, **connectTime**, **connector**, **username**, **userAgent**, and **clientAddress**.

```
CALL dbms.listConnections() YIELD connectionId, connectTime, connector, username, userAgent,  
clientAddress
```

```
"connectionId" "connectTime" "connector" "username" "userAgent"  
"clientAddress" "status"  
  
"bolt-21" "2018-10-10T12:11:42.276Z" "bolt" "alwood" "neo4j-java/1.6.3"  
"127.0.0.1:53929" "Running"  
  
"http-11" "2018-10-10T12:37:19.014Z" "http" null "Mozilla/5.0 (Macintosh; Intel Mac  
OS X 10.13; rv:62.0) Gecko/20100101 Firefox/62.0" "127.0.0.1:54118" "Running"  
  
2 rows
```

10.5.2. 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
ids	List<String>	This is a list of the IDs of all the connections to be terminated.

Returns:

Name	Type	Description
connectionId	String	This is the ID of the terminated connection.
username	String	This is the username of the user who initiated the (now terminated) connection.
message	String	A message stating whether the connection was successfully found.

Considerations:

Bolt connections are stateful. Termination of a Bolt connection results in termination of the ongoing query/transaction.

Termination of an HTTP/HTTPS connection can terminate the ongoing HTTP/HTTPS request.

Example 74. 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
```

10.5.3. 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
id	String	This is the ID of the connection to be terminated.

Returns:

Name	Type	Description
connectionId	String	This is the ID of the terminated connection.
username	String	This is the username of the user who initiated the (now terminated) connection.
message	String	A message stating whether the connection was successfully found.

Considerations:

Bolt connections are stateful. Termination of a Bolt connection results in termination of the ongoing query/transaction.

Termination of an HTTP/HTTPS connection can terminate the ongoing HTTP/HTTPS request.

Example 75. 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
```

10.6. 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
 - [Endpoints for Core Servers](#)
 - [Endpoints for Read Replicas](#)
- [Monitoring a multi-cluster](#)

10.6.1. 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()` can be called on every instance in a Causal Cluster to return the role of the instance.

Syntax:

```
CALL dbms.cluster.role()
```

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 76. Check the role of this instance

The following example shows how to find out the role of the current instance, which in this case is `FOLLOWER`.

```
CALL dbms.cluster.role()
```

```
role
```

```
FOLLOWER
```

Gain an overview over the instances in the cluster

The procedure `dbms.cluster.overview()` provides an overview of cluster topology by returning details on all the instances in the cluster.

Syntax:

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

Returns:

Name	Type	Description
id	String	This is id of the instance.
addresses	List<String>	This is a list of all the addresses for the instance.
role	String	This is the role of the instance, which can be LEADER, FOLLOWER, or READ_REPLICA.
groups	List<String>	This is a list of all the server groups which an instance is part of.
database	String	This is the name of the database which the instance is hosting.

Considerations:

- This procedure can only be called from Core instances, since they are the only ones that have the full view of the cluster.
- The value of the "database" field will always be "default" unless [deploying a multi-cluster](#).

Example 77. Get an overview of the cluster

The following example shows how to explore the cluster topology.

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

id	addresses	role	groups	database
08eb9305-53b9-4394-9237-0f0d63bb05d5	[bolt://neo20:7687, http://neo20:7474, https://neo20:7473]	LEADER	[]	default
cb0c729d-233c-452f-8f06-f2553e08f149	[bolt://neo21:7687, http://neo21:7474, https://neo21:7473]	FOLLOWER	[]	default
ded9eed2-dd3a-4574-bc08-6a569f91ec5c	[bolt://neo22:7687, http://neo22:7474, https://neo22:7473]	FOLLOWER	[]	default
00000000-0000-0000-0000-000000000000	[bolt://neo34:7687, http://neo34:7474, https://neo34:7473]	READ_REPLICA	[]	default
00000000-0000-0000-0000-000000000000	[bolt://neo28:7687, http://neo28:7474, https://neo28:7473]	READ_REPLICA	[]	default
00000000-0000-0000-0000-000000000000	[bolt://neo31:7687, http://neo31:7474, https://neo31:7473]	READ_REPLICA	[]	default

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.cluster.routing.getServer()` provides this information.

Syntax:

```
CALL dbms.cluster.routing.getServer()
```

Example 78. Get routing recommendations

The following example shows how discover which instances in the cluster can provide which services.

```
CALL dbms.cluster.routing.getServers()
```

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) for the information.

The result is not primarily intended for human consumption. Expanding it this is what it looks like.

```
ttl: 300,
server: [
{
  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
}]
```

10.6.2. 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](#)
- [Endpoints for Core Servers](#)
- [Endpoints for Read Replicas](#)

Adjusting security settings for Causal Clustering endpoints

If authentication and authorization is enabled in Neo4j (see [Configuration](#)), the Causal Clustering status endpoints will also require authentication credentials. For some load balancers and proxy servers, providing this 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/manage/server/causalclustering/writable` — Used to direct `write` traffic to specific instances.
- `/db/manage/server/causalclustering/read-only` — Used to direct `read` traffic to specific instances.
- `/db/manage/server/causalclustering/available` — Available for the general case of directing arbitrary request types to instances that are available for processing read transactions.
- `/db/manage/server/causalclustering/status` — Gives a detailed description of this instance's view of its own status within the cluster. Useful for monitoring and coordinating rolling upgrades. See [Status endpoint](#) for further details.

Table 26. Unified HTTP endpoint responses

Endpoint	Instance state	Returned code	Body text
<code>/db/manage/server/causalclustering/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/manage/server/causalclustering/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>
<code>/db/manage/server/causalclustering/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/manage/server/causalclustering/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.

Status endpoint

The status endpoint, available at `/db/manage/server/causalclustering/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 79. 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,
  "healthy": true
}
```

Table 27. 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 <code>memberId</code> that are considered part of the voting set by this core.
healthy	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 <code>memberId</code> to distinguish between core and replica instances.
leader	string	yes	80a7fb7b-c966-4ee7-88a9-35db8b4d68fe	Follows the same format as <code>memberId</code> , 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.

In general, you will want to follow the pattern of first adding a new, updated instance, and then removing an old instance. 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. This process can then be repeated until all old cores have been removed.

Table 28. Measured values, accessed via the status endpoint

Name of check	Method of calculation	Description
allServersAreHealthy	Every core's status endpoint indicates <code>dbHealth==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.
allVotingSetsAreEqual	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.
allVotingSetsContainAtLeastTargetCluster	For all cores (S), excluding core Z (to be switched off), every member in S contains S in their voting set. Membership is determined by using the <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).
hasOneLeader	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.
noMembersLagging	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](#).

Endpoints for Core Servers

[Core Servers](#) provide the following endpoints for status monitoring:

- `/db/manage/server/core/writable` — Used to direct `write` traffic to specific instances.
- `/db/manage/server/core/read-only` — Used to direct `read` traffic to specific instances.
- `/db/manage/server/core/available` — Available for the general case of directing arbitrary request types to instances that are available for processing read transactions.

Table 29. Core HTTP endpoint responses

Endpoint	Instance state	Returned code	Body text
<code>/db/manage/server/core/writable</code>	Leader	<code>200 OK</code>	<code>true</code>
	Follower	<code>404 Not Found</code>	<code>false</code>
<code>/db/manage/server/core/read-only</code>	Leader	<code>404 Not Found</code>	<code>false</code>
	Follower	<code>200 OK</code>	<code>true</code>
<code>/db/manage/server/core/available</code>	Leader	<code>200 OK</code>	<code>true</code>
	Follower	<code>200 OK</code>	<code>true</code>

Example 80. 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/manage/server/core/writable
* About to connect() to localhost port 7474 (#0)
*   Trying ::1...
* connected
* Connected to localhost (::1) port 7474 (#0)
> GET /db/manage/server/core/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(6.1.25)
<
* Connection #0 to host localhost left intact
true* Closing connection #0
```

Endpoints for Read Replicas

[Read Replicas](#) provides the following endpoint for status monitoring:

- `/db/manage/server/read-replica/available` — Available for the general case of directing arbitrary request types to instances that are available for processing read transactions.

Table 30. Read Replica HTTP endpoint responses

Endpoint	Returned code	Body text
/db/manage/server/read-replica/available	200 OK	true

10.6.3. Monitoring a multi-cluster

This section describes specific monitoring facilities for a Neo4j multi-cluster.

A number of procedures are available that provide information about a multi-cluster. This section describes the following:

- [Gain an overview over the instances in a multi-cluster](#)
- [Get routing information for all databases in a multi-cluster](#)
- [Get routing information for a given database in a multi-cluster](#)

The image below gives an overview of the procedures available for finding out information about a multi-cluster:



Figure 22. The procedures available to monitor a multi-cluster.

Gain an overview over all the instances in a multi-cluster

The `dbms.cluster.overview()` procedure is described in detail in [Gain an overview over the instances in the cluster](#). It provides an overview of the whole multi-cluster, regardless of which cluster member is used to run the query, and which constituent cluster that member belongs to. The name of the database managed by each cluster member is provided by the "database" column.

Unlike when [monitoring a single Causal Cluster](#), the cluster overview output will contain multiple

members with the role of Leader. This is to be expected, as each constituent cluster will have its own instance filling the Leader role.

Get routing information for all databases in a multi-cluster

Use this procedure for discovery of routing-capable endpoints for all databases in a multi-cluster.

Syntax:

```
CALL dbms.cluster.routing.getRoutersForAllDatabases()
```

Returns:

A json map containing the following fields:

Name	Description
ttl	Time To Live (TTL) for this information.
routers	A list of routers available for this configuration.
addresses	A list of addresses available for routing.
database	The database name that this information is applicable to.

Considerations:

This information is not primarily intended for humans, but for consumption by a client-side directory program, which can then issue a full routing request (see [bolt+routing](#)) to one of the returned addresses.

Example 81. Get routers for all the databases in a multi-cluster

In order to see routing-capable endpoints in the multi-cluster containing the databases `foo`, `bar` and `baz`, run the following query against any Core Server in the multi-cluster:

```
CALL dbms.cluster.routing.getRoutersForAllDatabases()
```

The result is a list of addresses of members that support routing in the `foo`, `bar` and `baz` clusters, together with the time to live (TTL) for this information.

```
ttl: 300,
routers: [
  {
    addresses: [neo20:7687,neo21:7687,neo22:7687],
    database: "foo"
  },
  {
    addresses: [neo23:7687,neo24:7687,neo25:7687],
    database: "bar"
  },
  {
    addresses: [neo26:7687,neo27:7687,neo28:7687],
    database: "baz"
  }
]
```

Get routing information for a given database in a multi-cluster

Use this procedure for discovery of routing-capable endpoints for a given database in a multi-cluster.

Syntax:

```
CALL dbms.cluster.routing.getRoutersForDatabase(database-name)
```

Returns:

A json map containing the following fields:

Name	Description
ttl	Time To Live (TTL) for this information.
routers	A list of routers available for this configuration.
addresses	A list of addresses available for routing.
database	The database name that this information is applicable to.

Considerations:

This information is not primarily intended for humans, but for consumption by a client-side directory program, which can then issue a full routing request (see [bolt+routing](#)) to one of the returned addresses.

Example 82. Get routers for a given database in a multi-cluster

In order to see routing-capable endpoints (see [bolt+routing](#)) in the cluster hosting the `foo` database, run the following query against any instance in the multi-cluster:

```
CALL dbms.cluster.routing.getRoutersForDatabase("foo")
```

The result is a list of addresses of members that support routing in the `foo` cluster, together with the time to live (TTL) for this information. This information is not primarily intended for humans, but for consumption by a client-side directory program, which can then issue a full routing request to one of the returned addresses.

```
ttl: 300,
routers: [
  {
    addresses: [neo20:7687,neo21:7687,neo22:7687],
    database: "foo"
  }
]
```

Chapter 11. 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.
- [Compressed storage](#) — How to configure the compressed storage of properties.
- [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.

11.1. Memory configuration

This section describes the different aspects of Neo4j memory configuration and use.

11.1.1. Overview

Consider the image below. As you can see, the RAM of the Neo4j server has a number of usage areas, with some sub-areas:

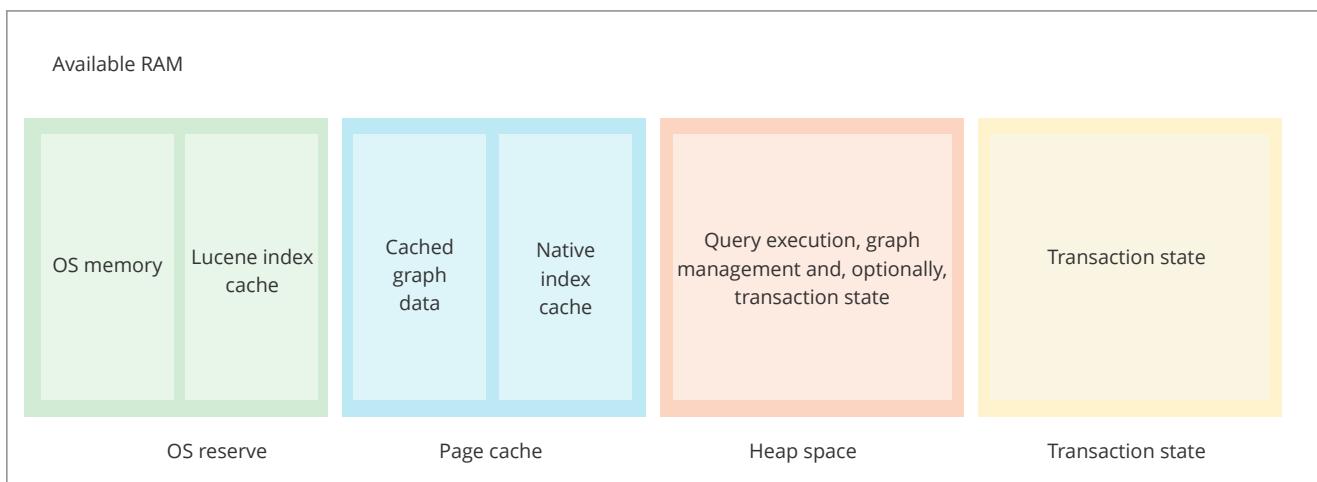


Figure 23. Neo4j memory management

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 page cache and heap space. However, if we do not leave enough space for the OS it will start swapping 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.

Lucene index cache

Neo4j uses [Apache Lucene](https://lucene.apache.org/) (<https://lucene.apache.org/>) for some of its indexing functionality. Index lookup performance is optimized by ensuring that as much as possible of the indexes are cached into memory. Similar to the OS memory, the Lucene index cache can not be explicitly configured. Instead we estimate the memory needed and make sure that there is enough headroom left for Lucene indexes after the page cache and heap cache have been assigned.

Page cache

The page cache is used to cache the Neo4j data and native indexes. The caching of graph data and indexes into memory will help 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`.

Heap size

The heap space is used for query execution, transaction state, management of the graph etc. The size needed for the heap is very dependent on the nature of the usage of Neo4j. For example, long-running queries, or very complicated queries, are likely to require a larger heap than simpler queries.

Generally speaking, in order to aid performance, we want to configure a large enough heap to sustain concurrent operations.

In case of performance issues we may have to tune our queries, and monitor their memory usage, in order to determine whether the heap needs to be increased.

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. This will help avoid unwanted full garbage collection pauses.

Transaction state

Transaction state is the memory that is needed to hold data and intermediate results in transactions that update records in the database. Queries that only read data do not require transaction state memory allocation. By default, transaction state is allocated from the heap, or *on-heap*. Note that when the transaction state is configured inside the heap, its maximum size cannot be specified.

Transaction state can also be configured to be allocated separately from the heap, or *off-heap*, by using the parameter `dbms.tx_state.memory_allocation`. When the transaction state is allocated off-heap, the maximum size of the transaction state can be defined using the parameter `dbms.tx_state.max_off_heap_memory`. Note that the transaction state memory is not pre-allocated; it will grow and shrink as needed by the activity in the database. Keeping transaction state off-heap is particularly beneficial to applications characterized by large, write-intensive transactions.

11.1.2. Considerations

Always use explicit configuration

In order to have good control of a system's behavior, it is recommended that you always define the page cache and heap size parameters explicitly in `neo4j.conf`. If these parameters are not explicitly defined, some heuristic values will be computed at startup based on 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 a database

The `neo4j-admin memrec --database` command is useful for inspecting the current distribution of data and indexes in an existing database.

Example 83. Use `neo4j-admin memrec` to inspect the memory settings of a database

We wish to estimate the total size of the database files.

```
$neo4j-home> bin/neo4j-admin memrec --database=graph.db
...
...
...
# Lucene indexes: 6690m
# Data volume and native indexes: 17050m
```

We 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, we can do a sanity check of our memory configuration:

- Compare the value for data volume and native indexes to the value of `dbms.memory.pagecache.size`.
- Compare the value for Lucene indexes to how much memory is left after assigning `dbms.memory.pagecache.size` and `dbms.memory.heap.initial_size`.

Note that even though we strive for caching as much of our data and indexes as possible, in some production systems the access to memory is limited and must be negotiated between different areas. Then there will be a certain amount of testing and tuning to figure out the optimal division of the available memory.

The effect of index providers on memory usage

After an upgrade from an earlier version of Neo4j, it is advantageous to rebuild certain indexes in order to take advantage of new index features. For details, see [Index configuration](#). The rebuilding of indexes will change the distribution of memory utilization. In a database with many indexes, a significant amount of memory may have been reserved for Lucene. After the rebuild, it could be necessary to allocate some of that memory to the page cache instead. Use `neo4j-admin memrec --database` to inspect the database before and after rebuilding indexes.

11.1.3. 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 we are already running in production or planning for a future deployment:

Example 84. Estimate page cache for an existing Neo4j database

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 --database=graph.db  
...  
...  
...  
# Lucene indexes: 6690m  
# Data volume and native indexes: 35050m
```

We can see that the data volume and native indexes combined take up approximately 35GB. In our specific use case we estimate that 20% will provide sufficient head room for growth.

`dbms.memory.pagecache.size = 1.2 * (35GB) = 42GB`

We configure the page cache by adding the following to `neo4j.conf`:

```
dbms.memory.pagecache.size=42GB
```

Example 85. 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 by that fraction plus some percentage for growth. For example, import 1/100th of the data and measure its data volume and native indexes. Then multiply that number by 120 to size up the result, and allow for 20% growth.

Assume that we have imported 1/100th of the data into a test database.

```
$neo4j-home> bin/neo4j-admin memrec --database=graph.db  
...  
...  
...  
# Lucene indexes: 425.0  
# Data volume and native indexes: 251100k
```

We can see that the data volume and native indexes combined take up approximately 250MB. We size up the result and additionally reserve 20% for growth:

`dbms.memory.pagecache.size = 120 * (250MB) = 30GB`

We configure the page cache by adding the following to `neo4j.conf`:

```
dbms.memory.pagecache.size=30G
```

11.2. Index configuration

This section describes configuration aspects of indexes in Neo4j.

This section contains the following:

- [Introduction](#)

- Schema indexes
 - Introduction
 - Index providers
 - Limitations of native indexes
 - Key size
 - Element size calculations
 - Non-composite indexes
 - Composite indexes
 - Queries using `CONTAINS` and `ENDS WITH`
 - Workarounds to address limitations
 - Limitations of Lucene-backed indexes
 - Upgrade considerations
- Fulltext schema indexes
 - Introduction
 - Configuration
 - Deprecation of explicit indexes

11.2.1. Introduction

This section introduces the concepts of schema indexes and fulltext schema indexes.

Indexes are used to speed up search.

When we talk about *indexes* we often mean *schema indexes*. Schema indexes are 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. Schema indexes are good at exact lookups, range scans, full scans, and prefix searches.

For details on the configuration aspects of schema indexes, see [Schema indexes](#). For details on creating, using and dropping schema indexes, see [Cypher Manual □ Indexes](#).

Fulltext schema indexes are sometimes referred to as *fulltext indexes*. They are used for queries that demand an understanding of language. For example, a fulltext index is useful if we wish to parse a book for a certain term, and take 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, allow for excluding words that are not relevant for the search (for example *if* and *and*) and include conjugations of words that are. In contrast to schema indexes, fulltext indexes are created, used and dropped using built-in procedures. The use of fulltext indexes do require a familiarity with how the indexes operate.

For details on the configuration aspects of schema indexes, see [Fulltext schema indexes](#). For details on creating, using and dropping schema indexes, see [Cypher Manual □ Fulltext schema index](#).

11.2.2. Schema indexes

This section describes schema indexes.

This section describes the following:

- [Introduction](#)
- [Index providers](#)
- [Limitations of native indexes](#)
 - [Key size](#)
 - [Element size calculations](#)
 - [Non-composite indexes](#)
 - [Composite indexes](#)
 - [Queries using CONTAINS and ENDS WITH](#)
 - [Workarounds to address limitations](#)
- [Limitations of Lucene-backed indexes](#)
- [Upgrade considerations](#)

Introduction

Neo4j uses a combination of native indexes and [Apache Lucene](#) (<https://lucene.apache.org/>) for its indexing functionality. The native index is an implementation of the classic [B+Tree](#) (https://en.wikipedia.org/wiki/B%2B_tree).

For performance reasons, it is recommended to use native indexes whenever possible.

For more information on the different index types, refer to [Cypher manual □ Indexes](#).

Index providers

The index provider used when creating new indexes is controlled by the setting `dbms.index.default_schema_provider`. If not configured explicitly, `dbms.index.default_schema_provider` will default to use the newest provider available in that particular version of Neo4j.

The table below lists the available index providers and their support for native indexing:

Index provider	Value types supported for native indexing	Type of native index supported
<code>native-btree-1.0</code>	spatial, temporal, numeric, string, array, boolean	Single-property and composite indexes
<code>lucene+native-2.0</code>	spatial, temporal, numeric, string	Single-property indexes
<code>lucene+native-1.0</code>	spatial, temporal, numeric	Single-property indexes
<code>lucene-1.0</code>	spatial, temporal	Single-property indexes

Deprecated index providers

Index providers `lucene-1.0`, `lucene+native-1.0`, and `lucene+native-2.0` have been deprecated, and will be removed in a future release.



The recommended index provider to use is `native-btree-1.0`.

The only reason to use a deprecated provider should be due to the limitations, as described in [Limitations of native indexes](#). There are currently no alternatives to cover these limitations, and deprecated providers will not be removed until there is.

Limitations of native indexes

Typically, the newest index provider version will provide the best performance. However, the native B+Tree implementation has some limitations which may require special handling.

Key size

The native B+Tree index has a key size limit that manifests itself in different ways depending on whether the key holds a single string, a single array, or multiple values (i.e. is the key in a *composite index*).

If a transaction reaches this limitation for one or more of its changes, that transaction will fail before committing any changes. If this limit is reached during index population, the resulting index will be in a failed state, thus not be usable for any queries.

The following sections describe the different key size limits in detail.

Element size calculations

It is useful to know how to calculate the size of a single value when calculating the total size of the resulting key. In some cases those entry sizes is different based on whether the entry is in an array or not.

Table 31. Element sizes

Type	<code>elementSize_ifSingle</code> *	<code>elementSize_ifInArray</code> **
Byte	2	1
Short	3	2
Int	5	4
Long	9	8
Float	5	4
Double	9	8
Boolean	1	1
Date	8	8
Time	12	12
LocalTime	8	8
DateTime	16	16
LocalDateTime	12	12
Duration	28	28
Period	28	28
Point (Cartesian)	28	24
Point (Cartesian 3D)	36	32
Point (WGS-84)	28	24
Point (WGS-84 3D)	36	32
String	<code>2 + utf8StringSize</code> ***	<code>2 + utf8StringSize</code> ***
Array	†	Nested arrays are not supported

* `elementSize_ifSingle` denotes the size of an element if is a single entry.

** `elementSize_ifInArray` denotes the size of an element if it is part of an array.

*** `utf8StringSize` is the size of the `String` in bytes when encoded with UTF8.

† `elementSizeArray` is the size of an array element, and is calculated using the following formulas:

- If the data type of the array is a numeric data type:

```
elementSizeArray = 3 + ( arrayLength * elementSizeifInArray )
```

- If the data type of the array is a geometry data type:

```
elementSizeArray = 5 + ( arrayLength * elementSizeifInArray )
```

- If the data type of the array is non-numeric:

```
elementSizeArray = 2 + ( arrayLength * elementSizeifInArray )
```

String encoding with UTF8



It is worth noting that common characters, such as letters, digits and some symbols, translate into one byte per character. Non-Latin characters may occupy more than one byte per character. Therefore, for example, a string that contains 100 characters or less may be longer than 100 bytes if it contains multi-byte characters.

More specifically, the relevant length in bytes of a string is when encoded with `UTF8` (<https://en.wikipedia.org/wiki/UTF-8>).

Example 86. Calculate the size of a string when used in an index

Consider the string `Sju sjösjuka sjömän sköttes av sju undersköna sjuksköterskor på skeppet Shang Hai`.

This string has 74 characters that each occupies one `Byte`. Additionally, there are 7 characters that each occupy two bytes per character, which add 14 to the total. Therefore, the size of the `String` in bytes when encoded with `UTF8`, `utf8StringSize`, is 88 bytes.

If this string is part of a native index, we get:

```
elementSize = 2 + utf8StringSize = 2 + 88 = 90 bytes
```

Example 87. Calculate the size of an array when used in an index

Consider the array `[19, 84, 20, 11, 54, 9, 59, 76, 82, 27, 9, 35, 56, 80, 65, 95, 16, 91, 61, 11]`.

This array has 20 elements of the type `Int`. Since they are in an array, we need to use `elementSizeifInArray`, which is 4 for `Int`.

Applying the formula for arrays of numeric data types, we get:

```
elementSizeArray = 3 + ( arrayLength * elementSizeifInArray ) = 3 + ( 20 * 4 ) = 83 bytes
```

Non-composite indexes

The only way that a non-composite index can violate the size limit is if the value is a long string or a large array.

Strings

Applicable to index providers: `native-btree-1.0` and `lucene+native-2.0`.



Please see [Deprecated index providers](#) for information on deprecation of lucene-based index providers.

Strings in non-composite native B+Tree indexes have a key size limit of 4036 bytes for index provider `native-btree-1.0`, and 4039 bytes for index provider `lucene+native-2.0`.

Arrays

Applicable to index provider: `native-btree-1.0`.

The following formula is used for arrays in non-composite indexes:

`1 + elementSizeArray <= 4039`

Here `elementSizeArray` is the number calculated from [Element sizes](#).

If we count backwards, we can get the exact array length limit for each data type:

- `maxArrayLength = FLOOR((4039 - 3 - 1) / elementSizeifInArray)` for numeric types.
- `maxArrayLength = FLOOR((4039 - 2 - 1) / elementSizeifInArray)` for non-numeric types.

These calculations result in the table below:

Table 32. Maximum array length, per data type

Data type	maxArrayLength
Byte	4035
Short	2017
Int	1008
Long	504
Float	1008
Double	504
Boolean	4036
String	See Maximum array length, examples for strings
Date	504
Time	336
LocalTime	504
DateTime	252
LocalDateTime	336
Duration	144
Period	144
Point (Cartesian)	168
Point (Cartesian 3D)	126
Point (WGS-84)	168
Point (WGS-84 3D)	126

Note that in most cases, Cypher will use `Long` or `Double` when working with numbers.

Properties with the type of `String` are a special case because they are dynamically sized. The table below shows the maximum number of array elements in an array, based on certain string sizes:

Table 33. Maximum array length, examples for strings

String size	maxArrayLength
1	1345
10	336
100	39
1000	4

The table can be used as a reference point. For example: if we know that all the strings in an array occupy 100 bytes or less, then arrays of length 39 or lower will definitely fit.

Composite indexes

Applicable to index provider: `native-btree-1.0`.

This limitation only applies if one or more of the following criteria is met:

- Composite index contains strings
- Composite index contains arrays
- Composite index targets many different properties (>50)

We denote a targeted property of a composite index a `slot`, and the number of slots `numberOfSlots`. For example, an index on `:Person(firstName, surName, age)` has three properties and thus `numberOfSlots = 3`.

In the index, each slot is filled by an *element*. In order to calculate the size of the index, we must have the size of each element in the index, i.e. the `elementSize`, as calculated in previous sections.

The following equation can be used to verify that a specific composite index entry is within bounds:

```
numberOfSlots + sum( elementSize ) <= 4039
```

Here, `sum(elementSize)` is the sum of the sizes of all the elements of the composite key as defined in [Element size calculations](#), and `numberOfSlots` is the number of targeted properties for the index.

Example 88. The size of a composite index containing strings

Consider a composite index of five strings that each can occupy the maximum of 500 bytes.

Using the equation above we get:

```
numberOfSlots + sum( elementSize ) = 5 + ( 5 * ( 2 + 500 ) ) = 2515 < 4039
```

We are well within bounds for our composite index.

Example 89. The size of an index containing arrays

Consider a composite index of five arrays of type `Float` that each have a length of 250.

First we calculate the size of each array element:

```
elementSizeArray = 3 + ( arrayLength * elementSizeifInArray ) = 3 + ( 250 * 4 ) = 1003
```

Then we calculate the size of the composite index:

```
numberOfSlots + sum( elementSizeArray ) = 5 + ( 5 * 1003 ) = 5020 > 4039
```

This index key will exceed the key size limit for native indexes.

To work around this, it is possible to create the index using the `lucene+native-2.0` index provider, as described in [Workarounds to address limitations](#), but please note that this index provider has been deprecated.

Queries using `CONTAINS` and `ENDS WITH`

Applicable to index providers: `native-btree-1.0` and `lucene+native-2.0`.



Please see [Deprecated index providers](#) for information on deprecation of lucene-based index providers.

Native B+Tree indexes have limited support for `ENDS WITH` and `CONTAINS` queries. These queries will not be able to do an optimized search the way they do for queries that use `STARTS WITH`, `=` and `<>`. Instead, the index result will be a stream of an index scan with filtering.

For details about execution plans, refer to [Cypher Manual](#) \square [Execution plans](#). For details about string operators, refer to [Cypher Manual](#) \square [Operators](#).

Workarounds to address limitations

If any of the limitations described in this section becomes a problem, a workaround is to specify an index provider that uses Lucene for that particular index. This can be done using either of the following methods:

Option 1; change the config

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.

Option 2; use a built-in procedure

There are built-in procedures that 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](#) \square [Constraints](#). For more information, see [Built-in procedures](#).

Limitations of Lucene-backed indexes

Applicable to index providers: `lucene+native-1.0` and `lucene-1.0`



Please see [Deprecated index providers](#) for information on deprecation of lucene-based index providers.

Lucene has a string size limit of 32766 bytes when string is encoded using UTF-8. In a composite index, this limit is applicable to each individual property. This means that a composite index key can hold values that together are larger than 32766 bytes but no single value can be larger.

Upgrade considerations

When creating an index, the current index provider will be assigned to it and will remain the provider for that index until it is dropped. Therefore, when upgrading to newer versions of Neo4j, an existing index needs to be dropped and recreated in order to take advantage of improved indexing features.

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.

11.2.3. Fulltext schema indexes

This section describes fulltext schema indexes.

This section describes the following:

- [Introduction](#)
- [Configuration](#)
- [Deprecation of explicit indexes](#)

Introduction

Fulltext schema indexes are powered by the [Apache Lucene](http://lucene.apache.org/) (<http://lucene.apache.org/>) indexing and search library. A fulltext schema index enables you to write queries that matches within the *contents* of indexed string properties. A full description on how to create and use fulltext schema indexes is provided in the [Cypher Manual](#) \square [Fulltext schema index](#).

Configuration

The following options are available for configuring fulltext schema indexes:

`dbms.index.fulltext.default_analyzer`

The name of the analyzer that the fulltext schema indexes should use by default. This setting only has effect when a fulltext schema 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 specific, the default analyzer is `standard`, which is the same as the `StandardAnalyzer` from Lucene.

`dbms.index.fulltext.eventually_consistent`

Whether or not fulltext schema indexes should be eventually consistent, or not. This setting only has effect when a fulltext schema index is created, and will be remembered as an index-specific setting from then on. Schema 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 fulltext schema 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 fulltext schema 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 fulltext schema indexes are fully consistent.

`dbms.index.fulltext.eventually_consistent_index_update_queue_max_length`

Eventually consistent fulltext schema 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 fulltext schema 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.



When Neo4j is deployed in Causal Cluster configurations, it is recommended that all cluster members have identical `dbms.index.fulltext.*` settings in their `neo4j.conf` files. This ensures that the indexes always behave predictably, when the cluster switches leader, or when members perform store copies.

Deprecation of explicit indexes

Fulltext indexes have previously been supported in Neo4j via the deprecated *explicit indexes*, but with some limitations that the fulltext schema indexes solve. This section outlines some of the similarities and differences in the two fulltext indexing implementations:

- Both schema and explicit fulltext indexes support indexing of both nodes and relationships.
- Both schema and explicit fulltext indexes support configuring custom analyzers, including analyzers that are not included with Lucene itself.
- Both schema and explicit fulltext indexes can be queried using the Lucene query language.
- Both schema and explicit fulltext indexes can return the *score* for each result from a query.
- The fulltext schema indexes are kept up to date automatically, as nodes and relationships are added, removed, and modified. The explicit *auto indexes* can do this as well, except it can get confused by id and space re-use, and produce wrong results from queries as a consequence. This is not a problem for the new fulltext schema indexes.
- The fulltext schema indexes will automatically populate newly created indexes with the existing data in a store. The explicit *auto indexes* do no such thing when they are enabled, and they will miss updates that occur while they are temporarily disabled or misconfigured.
- The fulltext schema indexes can be checked by the consistency checker, and they can be rebuilt if there is a problem with them. The explicit indexes are ignored by the consistency checker, and they cannot be automatically rebuilt if they develop any issues.
- The explicit indexes can be used to index by keys and values that are not actually in the store, so for instance if you want to index a node by the contents of a book without assigning it to the node as a property value, you can do that. The fulltext schema indexes are a projection of the store, and can only index nodes and relationships by the contents of their properties.
- The explicit indexes suffer from the Lucene limitation of only supporting up to at most 2 billion documents in a single index. The fulltext schema indexes have no such limitation.
- The explicit indexes interact poorly with a Causal Cluster. For instance, the fact that a new explicit index has been created can only be communicated from the leader to the rest of the cluster via a *store copy*. The fulltext schema indexes are created, dropped, and updated transactionally, and is replicated throughout a cluster automatically.
- The explicit indexes can be accessed via dedicated REST end-points and Java APIs, as well as Cypher procedures. The fulltext schema indexes can only be accessed via Cypher procedures.
- The fulltext schema indexes can be configured to be *eventually consistent*, in which index updating is moved from the commit path to a background thread. This removes the slow Lucene writes from the performance critical commit process, which has historically been among the main bottlenecks for Neo4j write performance. This is not possible to do with explicit indexes.

11.3. 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, and is independent of the size of the young generation. In the old generation, pause times roughly correlates with the size of the heap. For this reason, the heap should ideally be sized and tuned such that transaction and query state never makes it to the old generation.

The heap size is configured with the `dbms.memory.heap.max_size` (in MBs) setting in the `neo4j.conf` file. The initial size of the heap is specified by the `dbms.memory.heap.initial_size` setting, or with the `-Xms??m` flag, or chosen heuristically by the JVM itself if left unspecified. The JVM will automatically grow the heap as needed, up to the maximum size. The growing of the heap requires a full garbage collection cycle. It is recommended to set the initial heap size and the maximum heap size to the same value. This way the pause that happens when the garbage collector grows the heap can be avoided.

The ratio of the size between the old generation and the new generation of the heap is controlled by the `-XX:NewRatio=N` flag. `N` is typically between 2 and 8 by default. A ratio of 2 means that the old generation size, divided by the new generation size, is equal to 2. In other words, two thirds of the heap memory will be dedicated to the old generation. A ratio of 3 will dedicate three quarters of the heap to the old generation, and a ratio of 1 will keep the two generations about the same size. A ratio of 1 is quite aggressive, but may be necessary if your transactions changes a lot of data. Having a large new generation can also be important if you run Cypher queries that need to keep a lot of data resident, for example when sorting big result sets.

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 not enabled for heaps larger than 32 GB. Gains from increasing the heap size beyond 32 GB can therefore be small or even negative, unless the increase is significant (64 GB or above).

Neo4j has a number of long-lived objects, that stay around in the old generation, effectively for the lifetime of the Java process. To process them efficiently, and without adversely affecting the garbage collection pause time, we recommend using a concurrent garbage collector.

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.

- Use a concurrent garbage collector. We find that -XX:+UseG1GC works well in most use-cases.
 - The Neo4j JVM needs enough heap memory for the transaction state and query processing, plus some head-room for the garbage collector. Because the heap memory needs are so workload dependent, it is common to see configurations from 1 GB, up to 32 GBs of heap memory.
- Start the JVM with the -server flag and a good sized heap.
 - The operating system on a dedicated server can usually make do with 1 to 2 GBs of memory, but the more physical memory the machine has, the more memory the operating system will need.

Edit the following properties:

Table 34. 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

11.4. 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.

11.4.1. 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](#).

11.4.2. Configuration options

The following configuration options are available for configuring the Bolt connector:

Table 35. Thread pool options

Option name	Default	Description
thread_pool_min_size	5	The minimum number of threads that will always be up even if they are idle.
thread_pool_max_size	400	The maximum number of threads that will be created by the thread pool.
thread_pool_keep_alive	5m	The duration that the thread pool will wait before killing an idle thread from the pool. However, the number of threads will never go below <code>thread_pool_min_size</code> .

11.4.3. 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 `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 `thread_pool_max_size` based on your maximum workload. This should basically be set after the maximum number of active transactions that is expected on the server. You should also account for non-transaction operations that will take place on the thread pool, such as connection and disconnection of clients.

Example 90. Configure the thread pool for a Bolt connector

In this example we configure the Bolt thread pool to be of minimum size `5`, maximum size `100`, and have a keep-alive time of `10 minutes`.

```
dbms.connector.bolt.thread_pool_min_size=10  
dbms.connector.bolt.thread_pool_max_size=100  
dbms.connector.bolt.thread_pool_keep_alive=10m
```

11.5. Compressed storage

This section explains Neo4j property value compression and disk usage.

Neo4j can in many cases compress and inline the storage of property values, such as short arrays and strings, with the purpose of saving disk space and possibly an I/O operation.

Compressed storage of short arrays

Neo4j will try to store your primitive arrays in a compressed way. To do that, it employs a "bit-shaving" algorithm that tries to reduce the number of bits required for storing the members of the array. In particular:

1. For each member of the array, it determines the position of leftmost set bit.
2. Determines the largest such position among all members of the array.
3. It reduces all members to that number of bits.
4. Stores those values, prefixed by a small header.

That means that when even a single negative value is included in the array then the original size of the primitives will be used.

There is a possibility that the result can be inlined in the property record if:

- It is less than 24 bytes after compression.
- It has less than 64 members.

For example, an array `long[] {0L, 1L, 2L, 4L}` will be inlined, as the largest entry (4) will require 3 bits to store so the whole array will be stored in $4 \times 3 = 12$ bits. The array `long[] {-1L, 1L, 2L, 4L}` however will require the whole 64 bits for the -1 entry so it needs $64 \times 4 = 32$ bytes and it will end up in the dynamic store.

Compressed storage of short strings

Neo4j will try to classify your strings in a short string class and if it manages that it will treat it accordingly. In that case, it will be stored without indirection in the property store, inlining it instead in the property record, meaning that the dynamic string store will not be involved in storing that value, leading to reduced disk footprint. Additionally, when no string record is needed to store the property, it can be read and written in a single lookup, leading to performance improvements and less disk space required.

The various classes for short strings are:

- Numerical, consisting of digits 0..9 and the punctuation space, period, dash, plus, comma and apostrophe.
- Date, consisting of digits 0..9 and the punctuation space dash, colon, slash, plus and comma.
- Hex (lower case), consisting of digits 0..9 and lower case letters a..f
- Hex (upper case), consisting of digits 0..9 and upper case letters a..f
- Upper case, consisting of upper case letters A..Z, and the punctuation space, underscore, period, dash, colon and slash.
- Lower case, like upper but with lower case letters a..z instead of upper case
- E-mail, consisting of lower case letters a..z and the punctuation comma, underscore, period, dash, plus and the at sign (@).
- URI, consisting of lower case letters a..z, digits 0..9 and most punctuation available.
- Alpha-numerical, consisting of both upper and lower case letters a..zA..z, digits 0..9 and punctuation space and underscore.
- Alpha-symbolical, consisting of both upper and lower case letters a..zA..Z and the punctuation space, underscore, period, dash, colon, slash, plus, comma, apostrophe, at sign, pipe and semicolon.
- European, consisting of most accented european characters and digits plus punctuation space,

dash, underscore and period — like latin1 but with less punctuation.

- Latin 1.
- UTF-8.

In addition to the string's contents, the number of characters also determines if the string can be inlined or not. Each class has its own character count limits, which are

Table 36. Character count limits

String class	Character count limit
Numerical, Date and Hex	54
Uppercase, Lowercase and E-mail	43
URI, Alphanumeric and Alphasymbolical	36
European	31
Latin1	27
UTF-8	14

That means that the largest inline-able string is 54 characters long and must be of the Numerical class and also that all Strings of size 14 or less will always be inlined.

Also note that the above limits are for the default 41 byte PropertyRecord layout — if that parameter is changed via editing the source and recompiling, the above have to be recalculated.

11.6. 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.

By default, most Linux distributions schedule I/O requests using the Completely Fair Queuing (CFQ) algorithm, which provides a good balance between throughput and latency. The particular I/O workload of a database, however, is better served by the Deadline scheduler. The Deadline scheduler gives preference to *read* requests, and processes them as soon as possible. This tends to decrease the latency of reads, while the latency of writes goes up. Since the writes are usually sequential, their lingering in the I/O queue increases the chance of overlapping or adjacent write requests being merged together. This effectively reduces the number of writes that are sent to the drive.

On Linux, the I/O scheduler for a drive, in this case `sda`, can be changed at runtime like this:

```
$ echo 'deadline' > /sys/block/sda/queue/scheduler
$ cat /sys/block/sda/queue/scheduler
noop [deadline] cfq
```

Another 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. This can be accomplished by setting the `noatime, nodiratime` mount options in `fstab`, or when issuing the disk mount command.

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 ZFS file systems generally cope well with ageing; thus they are recommended as a first choice.

XFS can have a slightly higher write throughput than EXT4, and unlike EXT4 supports files that are larger than 32 TiB. However, it needs additional tuning to improve its ageing characteristics. A careful read of the `xfs` and `mkfs.xfs` man pages is recommended if you wish to run Neo4j on XFS, and that you have at least the `crc`, `finobt`, and `sparse` XFS options enabled.

Running Neo4j on the Btrfs is not advised, since it can behave badly when it is close to running out of storage space.

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. A larger drive gives the wear-levelling algorithm more room to work with. Enterprise drives generally have higher endurance than consumer drives. V-NAND has higher endurance than planer NAND, and TLC NAND lasts longer than QLC. Likewise MLC lasts longer than TLC, but at the cost of reduced drive capacity.

11.7. 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.

11.7.1. 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. Today a typical mechanical drive has an average seek time of about 5ms. This can cause a query or traversal to be very slow when the amount of RAM assigned to the page cache is too small. A new, good SATA enabled SSD has an average read service time of less than 100 microseconds, meaning those scenarios will execute at least 50 times faster, and modern NVMe drives can service reads in 10 to 50 microseconds. However, this is still hundreds of times slower than accessing RAM.

To avoid hitting disk you need more RAM. On a standard mechanical drive you can handle graphs with a few tens of millions of primitives (nodes, relationships and properties) with 2-3 GBs of RAM. A server with 8-16 GBs of RAM can handle graphs with hundreds of millions of primitives, and a good server with 16-64 GBs can handle billions of primitives. However, if you invest in a good SSD you will be able to handle much larger graphs on less RAM.

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 Lucene indexes don't quite fit in memory. In this case, queries that do index lookups will have high latencies.

11.7.2. 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 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, as the database is running. These profiles contain information about what data is and is not in memory, and are stored in a "profiles" sub-directory of the store directory. When Neo4j is later restarted, it will look for these cache profiles and eagerly load in the same data that was in memory when the profile was created. The profiles are also copied as part of online backup and cluster store-copy, and helps warm up new database instances that join a cluster.

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 easy to tune it until you have the right balance between IO usage and checkpoint time.

11.8. 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.

11.8.1. 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 91. 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();
```

11.8.2. Execution plans

Execution plans are cached and 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.

Chapter 12. 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 for a database.](#)
- [How to use the Cypher Shell](#)

12.1. Neo4j Admin

This section describes the Neo4j Admin tool.

This section describes the following:

- [Introduction](#)
- [Syntax and commands](#)
- [Environment variables](#)
- [Exit codes](#)

12.1.1. 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.

12.1.2. 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 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 Report tool .
	<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 .
Authentication	<code>set-default-admin</code>	Sets the default admin user when no roles are present.
	<code>set-initial-password</code>	Sets the initial password of the initial admin user (<code>neo4j</code>).
Offline backup For details see Dump and load databases .	<code>dump</code>	Dump a database into a single-file archive.
	<code>load</code>	Load a database from an archive created with the dump command.
Online backup For details see Backup .	<code>backup</code>	Perform an online backup from a running Neo4j server.
	<code>restore</code>	Restore a backed-up database.
Clustering	<code>unbind</code>	Removes cluster state data for the specified database. For details, see Unbind a Core Server .

Limitations

`neo4j-admin` must be invoked as the `neo4j` user in order to ensure the appropriate file permissions.

12.1.3. 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.

12.1.4. 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](#).

12.2. Consistency checker

This section 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.

12.2.1. Check consistency of a database or a backup

The `neo4j-admin` tool is located in the `bin` directory. Run it with the `check-consistency` argument in order to check the consistency of a database.

Syntax

```
neo4j-admin check-consistency [--database=<name>]
[backup=</path/to/backup>] [--verbose[=<true|false>]]
[--report-dir=<directory>] [--additional-config=<config-file->
path>] [--check-graph[=<true|false>]] [--check-label-scan-
[--check-indexes[=<true|false>]] [--check-property-owners[=<true|false>]]]
store[=<true|false>]]
```

Options

Option	Default	Description
--database	graph.db	Name of database.
--backup		Path to backup to check consistency of. Cannot be used together with --database.
--additional-config		Configuration file to supply additional configuration in.
--verbose	false	Enable verbose output.
--report-dir	.	Directory to write report file in.
--check-graph	true	Perform checks between nodes, relationships, properties, types and tokens.
--check-indexes	true	Perform checks on indexes.
--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.

Limitations

The consistency checker cannot be used with a database which is currently in use. If used with a running database, it will stop and print an error.

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 92. 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 stop
$neo4j-home> bin/neo4j-admin check-consistency --database=graph.db

2017-02-02 09:45:31.719+0000 INFO [o.n.k.i.s.f.RecordFormatSelector] Selected
RecordFormat:StandardV3_0[v0.A.7] record format from store /Users/maria/neo4j-enterprise-3.2.0-
alpha03/data/databases/graph.db
2017-02-02 09:45:31.719+0000 INFO [o.n.k.i.s.f.RecordFormatSelector] Format not configured. Selected
format from the store: RecordFormat:StandardV3_0[v0.A.7]
2017-02-02 09:45:31.963+0000 INFO [o.n.m.MetricsExtension] Initiating metrics...
..... 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/graph.db-backup
```

12.3. Report tool

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 [--list] [--to=</tmp>] [--verbose] [--force] [all][<classifier1>
<classifier2> ...]
```

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

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.
activetxs	X	Include the output of <code>dbms.listTransactions()</code> .
ccstate		Include the current cluster state.
config		Include the <code>neo4j.conf</code> file.
env	X	Include a list of all environment variables.
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.
tree		Include a text view of the folder structure of the data directory (no files are collected).
tx		Include transaction logs.

The classifiers marked as *Online* 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`, `env`, `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](#) (<https://docs.oracle.com/javase/8/docs/technotes/guides/attach/index.html>) to gather data from a running Neo4j instance. Therefore, it requires the Java JDK in order to execute properly.

12.4. Display store information

This chapter describes the `store-info` command of Neo4j Admin.

The version of a Neo4j database can be displayed using the `store-info` command of `neo4j-admin`:

```
neo4j-admin store-info --store=<path-to-store>
```

The `store` argument takes a path to a Neo4j database or backup. We will know if the store will need to go through a format migration as part of restoring a backup, by the presence of the message `Store format superseded in:` in the output.



The Neo4j Admin `store-info` command can not be used on a running database. The store is locked for protection while the database is running, and using the `store-info` command on it will fail with an error message indicating this.

Example 93. Example usage with older backup

You have an old backup of Neo4j a folder called `/backups/graph-db.2016-11-11` which you would like to restore. It is unclear what the version of Neo4j was at the time of backup. To find out, you would run:

```
$neo4j-home> bin/neo4j-admin store-info --store=/backups/graph-db.2016-11-11
Store format version:      VE.H.0
Introduced in version:    3.0.0
Store format superseded in: 3.0.6
```

The output reveals that the database was configured to use the high-limit format, which is Enterprise Edition only. This means the backup can only be used with Neo4j Enterprise Edition.

The output also explains that the format was introduced in Neo4j 3.0.0, but a newer format was shipped in Neo4j 3.0.6. This means that a format migration must be performed if the backup is restored (see [Upgrade](#) for details).

Example 94. Example usage with newer backup

As with the previous example, let us assume that another backup of Neo4j is present in `/backups/graph-db.2016-11-11/`. In this case the output is:

```
$neo4j-home> bin/neo4j-admin store-info --store=/backups/graph-db.2016-11-11
Store format version:      v0.A.7
Introduced in version:    3.0.0
```

The output tells us that the backup is made with the standard store format, which all editions of Neo4j support. It is also the newest version of the format known to Neo4j. This means that no format migration is necessary for the backup to be restored.

12.5. 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>] [--database=<name>]
```

The recommendations will be given in a format such that it can be copied and pasted straight into `neo4j.conf`.

Options

Option	Default	Description
<code>--memory</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>--database</code>	<code>graph.db</code>	The name of the database. This option will generate numbers for Lucene indexes, and for data volume and native indexes in the database. These can be used as an input into more detailed memory analysis.

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 95. 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
```

12.6. 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. 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 □ Schema](#)).

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`.

12.6.1. Syntax

The syntax for importing a set of CSV files is:

```
neo4j-admin import [--mode=csv] [--database=<name>]
                  [--additional-config=<config-file-path>]
                  [--report-file=<filename>]
                  [--nodes[:Label1:Label2]=<"file1,file2,... ">]
                  [--relationships[:RELATIONSHIP_TYPE]=<"file1,file2,... ">]
                  [--id-type=<STRING|INTEGER|ACTUAL>]
                  [--input-encoding=<character-set>]
                  [--ignore-extra-columns[=<true|false>]]
                  [--ignore-duplicate-nodes[=<true|false>]]
                  [--ignore-missing-nodes[=<true|false>]]
                  [--multiline-fields[=<true|false>]]
                  [--delimiter=<delimiter-character>]
                  [--array-delimiter=<array-delimiter-character>]
                  [--quote=<quotation-character>]
                  [--max-memory=<max-memory-that-importer-can-use>]
                  [--f=<File containing all arguments to this import>]
                  [--high-io=<true/false>]
```

Example 96. 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"
```

The syntax for importing a pre-3.0 database is:

```
neo4j-admin import --mode=database [--database=<name>]
[--additional-config=<config-file-path>]
[--from=<source-directory>]
```

Details on how to use `neo4j-admin import` for upgrading a pre-3.0 database can be found in the [upgrade chapter](#).

12.6.2. CSV file header format

This section explains the header format of CSV files when using the Neo4j import tool.

This section describes the following:

- [Header files](#)
- [Properties](#)
- [Node files](#)
- [Relationship files](#)
- [Using ID spaces](#)
- [Skipping columns](#)
- [Compressed files](#)

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`.

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 97. 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`.

```
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](#). It is possible to specify a default time zone for `Time` values, for example: `time{timezone:+02:00}`, and `DateTime` values, for example: `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 value position in the data file.

Example 98. 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

For files containing node data, there is one mandatory field; the `ID`, and one optional field; the `LABEL`:

`ID`

Each node must have a unique ID which is used during 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.

`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 99. 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

For files containing relationship data, there are three mandatory fields:

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 100. 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 101. 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 102. 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 103. 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
```

12.6.3. Options

This section describes in details the options available when using the Neo4j import tool to import data from CSV files.

--database=<name>

Name of database. Default: `graph.db`

--additional-config=<config-file-path>

Configuration file to supply additional configuration in. Default:

--mode=<database|csv>

Import a collection of CSV files or a pre-3.0 installation. Default: `csv`

--from=<source-directory>

The location of the pre-3.0 database (e.g. <neo4j-root>/data/graph.db). Default:

--report-file=<filename>

File in which to store the report of the csv-import. Default: `import.report`

--nodes[:Label1:Label2]=<"headerfile,file1,file2,...">

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. Note that file groups must be enclosed in quotation marks. Files can also be specified using regular expressions. For an example, see [Using regular expressions for specifying multiple input files](#). Default:

--relationships[:RELATIONSHIP_TYPE]=<"headerfile,file1,file2,...">

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. Note that file groups must be enclosed in quotation marks. Files can also be specified using regular expressions. For an example, see [Using regular expressions for specifying multiple input files](#). Default:

--id-type=<STRING|INTEGER|ACTUAL>

Each node must provide a unique id. This is used to find the correct nodes when creating relationships. Possible values are: `STRING`: arbitrary strings for identifying nodes, `INTEGER`: arbitrary integer values for identifying nodes, `ACTUAL`: (advanced) actual node ids. 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`

--ignore-duplicate-nodes=<true/false>

If duplicate nodes should be ignored during the import. Default: `false`

--ignore-missing-nodes=<true/false>

If relationships referring to missing nodes should be ignored during the import. Default: `false`

--multiline-fields=<true/false>

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`

--delimiter=<delimiter-character>

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=<array-delimiter-character>

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=<quotation-character>

Character to treat as quotation character for values in CSV data. Quotes can be escaped by doubling them, for example `"` would be interpreted as a literal `"`. You cannot escape using \. Default: `"`

--max-memory=<max-memory-that-importer-can-use>

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 e.g. `20G` for 20 gigabyte. It can also be specified as a percentage of the available memory, e.g. `70%`. Default: `90%`

--f=<arguments-file>

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. If this argument is used, no additional arguments are supported.

--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.

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 parameter `dbms.record_format=high_limit` in a configuration file, and supplying that file to the importer with `--additional-config`. 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 setting the environment variable `NEO4J_DEBUG=true`, and rerunning the import.

12.7. 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 should be shutdown before running the `dump` and `load` commands.
- `neo4j-admin` must be invoked as the `neo4j` user in order to ensure the appropriate file permissions.

Examples of usage

- Moving databases from one environment to another.

Using the `dump` and `load` commands is the recommended, and safe, way of transferring databases between environments. They understand which files need to be exported and imported and which should not.

By contrast, file system copy-and-paste of databases is not supported.

- Offline backups.

For a discussion of online versus offline backups, see [Online and offline backups](#).

Examples

Example 104. Use the `dump` command of `neo4j-admin`

Dump the database called `graph.db` into a file called `/backups/graph.db/2016-10-02.dump`. The destination directory for the dump file — in this case `/backups/graph.db` — must exist before calling the command.

```
$neo4j-home> bin/neo4j-admin dump --database=graph.db --to=/backups/graph.db/2016-10-02.dump  
$neo4j-home> ls /backups/graph.db  
$neo4j-home> 2016-10-02.dump
```

Example 105. Use the `load` command of `neo4j-admin`

Load the backed-up database contained in the file `/backups/graph.db/2016-10-02.dump` into database `graph.db`. Since we have a database running, we first have to shut it down. When we use the `--force` option, any existing database gets overwritten.

```
$neo4j-home> bin/neo4j stop  
Stopping Neo4j... stopped  
$neo4j-home> bin/neo4j-admin load --from=/backups/graph.db/2016-10-02.dump --database=graph.db  
--force
```



If using the `load` command to seed a Causal Cluster, you must first perform `neo4j-admin unbind` on each of the cluster instances. The procedure is described in [Seed from an offline backup](#).

12.8. Unbind a Core Server

This section describes how to remove cluster state data for a Core Server.

The cluster state of a Causal Cluster Core member can be removed by using the following command:

```
neo4j-admin unbind [--database=<name>]
```

Limitations

The database should be shutdown before running the `unbind` command.

Examples of usage

- When downgrading a Causal Cluster member:

The `unbind` command can be used to turn a Causal Cluster Core Server into a standalone server. To start the database in single mode after unbinding it from the cluster, first set `dbms.mode=SINGLE` in `neo4j.conf`.

- When seeding a Causal Cluster:

If you wish to seed a Causal Cluster with a database, and that cluster has previously been started, you must first run `neo4j-admin unbind` on each of the Core Servers. For more information about seeding Causal Clusters, see [Seed a cluster](#).

- When recovering a Causal Cluster:

If you need to recover a Causal Cluster from a backup, you must first run `neo4j-admin unbind` on each of the Core Servers. For more information about recovering a database from online backups, see [Restore a backup](#).



Since Read Replicas are not members of the Core cluster, and therefore do not have a cluster state, the `unbind` command is not applicable for Read Replicas.

12.9. Cypher Shell

This section describes the Neo4j Cypher Shell.

Cypher Shell is a command-line tool that is installed as part of the product. You can connect to a Neo4j database and use Cypher to query data, define schema or perform administrative tasks. Cypher Shell exposes explicit transactions allowing multiple operations to be grouped and applied or rolled back together. Cypher Shell communicates via the encrypted binary protocol Bolt.

12.9.1. Invoking Cypher Shell

Cypher Shell is located in the `bin` directory and is invoked with a set of arguments.

```
cypher-shell [-h] [-a ADDRESS] [-u USERNAME] [-p PASSWORD] [--encryption {true,false}] [--format {verbose,plain}] [--debug] [--non-interactive] [-v] [--fail-fast | --fail-at-end] [cypher]
```

Arguments

Positional arguments:	
<code>cypher</code>	An optional string of cypher to execute and then exit.
Optional arguments:	
<code>-h, --help</code>	Show help message and exit.
<code>--fail-fast</code>	Exit and report failure on first error when reading from file (this is the default behavior).
<code>--fail-at-end</code>	Exit and report failures at end of input when reading from file.
<code>--format {auto,verbose,plain}</code>	Desired output format. <code>auto</code> displays results in tabular format if you use the shell interactively and with minimal formatting if you use it for scripting. <code>verbose</code> displays results in tabular format and prints statistics and <code>plain</code> displays data with minimal formatting (default: auto).
<code>--debug</code>	Print additional debug information (default: false).
<code>--non-interactive</code>	Force non-interactive mode; only useful if auto-detection fails (default: false).
<code>-v, --version</code>	Print version of cypher-shell and exit (default: false).
Connection arguments:	
<code>-a ADDRESS, --address ADDRESS</code>	Address and port to connect to (default: bolt://localhost:7687).
<code>-u USERNAME, --username USERNAME</code>	Username to connect as. Can also be specified using environment variable NEO4J_USERNAME (default:).
<code>-p PASSWORD, --password PASSWORD</code>	Password to connect with. Can also be specified using environment variable NEO4J_PASSWORD (default:).

```
--encryption {true,false}
```

Whether the connection to Neo4j should be encrypted; must be consistent with Neo4j's configuration (default: true).

Example 106. Invoke Cypher Shell with username and password

```
$neo4j-home> bin/cypher-shell -u johndoe -p secret
```

```
Connected to Neo4j at bolt://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 107. Invoke help from within Cypher Shell

```
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
```

```
For help on a specific command type:  
:help command
```

Example 108. Execute a query from within Cypher Shell

```
neo4j> MATCH (n) RETURN n;
```

```
+-----+  
| n |  
+-----+  
| (:Person {name: "Bruce Wayne", alias: "Batman"}) |  
| (:Person {name: "Selina Kyle", alias: ["Catwoman", "The Cat"]}) |  
+-----+
```

Example 109. Invoke Cypher Shell with a Cypher script from the command line

Below is 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 will use the `--format plain` flag for a simple output.

```
$neo4j-home> cat examples.cypher | bin/cypher-shell -u neo4j -p secret --format plain
```

```
n  
(:Person {name: "Bruce Wayne", alias: "Batman"})  
(:Person {name: "Selina Kyle", alias: ["Catwoman", "The Cat"]})  
batman  
(:Person {name: "Bruce Wayne", alias: "Batman"})
```

12.9.2. Query parameters

Cypher Shell supports querying based on parameters. This is often used while scripting.

Example 110. 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
```

12.9.3. Transactions

Cypher Shell supports explicit transactions. Transaction states are controlled using the keywords `:begin`, `:commit`, and `:rollback`:

Example 111. 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 now open up 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>
```

12.9.4. Procedures

Cypher Shell supports running any procedures for which the current user is authorized. Here, we are using the natively built-in procedure `dbms.showCurrentUser()`.

Example 112. Call a procedure from within Cypher Shell

```
neo4j> CALL dbms.showCurrentUser();
+-----+
| username | roles      | flags |
+-----+
| "neo4j"  | ["admin"] | []   |
+-----+
1 row available after 66 ms, consumed after another 2 ms
neo4j> :exit
```

Appendix A: Reference

This appendix contains the Neo4j configuration settings reference, the list of built-in procedures bundled with Neo4j, and a description of user management for Community Edition.

This appendix contains the following:

- Configuration settings
- Built-in procedures
- User management for Community Edition

A.1. Configuration settings

This section contains a complete reference of Neo4j configuration settings. They can be set in `neo4j.conf`.

All settings

- `bolt.ssl_policy`: Specify the SSL policy to use for the encrypted bolt connections.
- `browser.allow_outgoing_connections`: Configure the policy for outgoing Neo4j Browser connections.
- `browser.credential_timeout`: Configure the Neo4j Browser to time out logged in users after this idle period.
- `browser.post_connect_cmd`: Commands to be run when Neo4j Browser successfully connects to this server.
- `browser.remote_content_hostname_whitelist`: Whitelist of hosts for the Neo4j Browser to be allowed to fetch content from.
- `browser.retain_connection_credentials`: Configure the Neo4j Browser to store or not store user credentials.
- `causal_clustering.array_block_id_allocation_size`: The size of the ID allocation requests Core servers will make when they run out of ARRAY_BLOCK IDs.
- `causal_clustering.catch_up_client_inactivity_timeout`: The catch up protocol times out if the given duration elapses with no network activity.
- `causal_clustering.catchup_batch_size`: The maximum batch size when catching up (in unit of entries).
- `causal_clustering.cluster_allow_reads_on_followers`: Configure if the `dbms.cluster.routing.getServers()` procedure should include followers as read endpoints or return only read replicas.
- `causal_clustering.cluster_routing_ttl`: How long drivers should cache the data from the `dbms.cluster.routing.getServers()` procedure.
- `causal_clustering.cluster_topology_refresh`: Time between scanning the cluster to refresh current server's view of topology.
- `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.database`: The name of the database being hosted by this server instance.
- `causal_clustering.disable_middleware_logging`: Prevents the network middleware from dumping its

own logs.

- `causal_clustering.discovery_advertised_address`: Advertised cluster member discovery management communication.
- `causal_clustering.discovery_listen_address`: Host and port to bind the cluster member discovery management communication.
- `causal_clustering.discovery_type`: Configure the discovery type used for cluster name resolution.
- `causal_clustering.enable_pre_voting`: Enable pre-voting extension to the Raft protocol (this is breaking and must match between the core cluster members).
- `causal_clustering.expected_core_cluster_size`: Expected number of Core machines in the cluster before startup.
- `causal_clustering.global_session_tracker_state_size`: The maximum file size before the global session tracker state file is rotated (in unit of entries).
- `causal_clustering.handshake_timeout`: Time out for protocol negotiation handshake.
- `causal_clustering.id_alloc_state_size`: The maximum file size before the ID allocation file is rotated (in unit of entries).
- `causal_clustering.in_flight_cache.max_bytes`: The maximum number of bytes in the in-flight cache.
- `causal_clustering.in_flight_cache.max_entries`: The maximum number of entries in the in-flight cache.
- `causal_clustering.in_flight_cache.type`: Type of in-flight cache.
- `causal_clustering.initial_discovery_members`: A comma-separated list of other members of the cluster to join.
- `causal_clustering.join_catch_up_timeout`: Time out for a new member to catch up.
- `causal_clustering.kubernetes.address`: Address for Kubernetes API.
- `causal_clustering.kubernetes.ca_crt`: File location of CA certificate for Kubernetes API.
- `causal_clustering.kubernetes.label_selector`: LabelSelector for Kubernetes API.
- `causal_clustering.kubernetes.namespace`: File location of namespace for Kubernetes API.
- `causal_clustering.kubernetes.service_port_name`: Service port name for discovery for Kubernetes API.
- `causal_clustering.kubernetes.token`: File location of token for Kubernetes API.
- `causal_clustering.label_token_id_allocation_size`: The size of the ID allocation requests Core servers will make when they run out of LABEL_TOKEN IDs.
- `causal_clustering.label_token_name_id_allocation_size`: The size of the ID allocation requests Core servers will make when they run out of LABEL_TOKEN_NAME IDs.
- `causal_clustering.last_applied_state_size`: The maximum file size before the storage file is rotated (in unit of entries).
- `causal_clustering.leader_election_timeout`: The time limit within which a new leader election will occur if no messages are received.
- `causal_clustering.load_balancing.config`: The configuration must be valid for the configured plugin and usually exists under matching subkeys, e.g.
- `causal_clustering.load_balancing.plugin`: The load balancing plugin to use.
- `causal_clustering.load_balancing.shuffle`: Enables shuffling of the returned load balancing result.
- `causal_clustering.log_shipping_max_lag`: The maximum lag allowed before log shipping pauses (in unit of entries).
- `causal_clustering.middleware_logging.level`: The level of middleware logging.
- `causal_clustering.minimum_core_cluster_size_atFormation`: Minimum number of Core machines in the cluster at formation.

- `causal_clustering.minimum_core_cluster_size_at_runtime`: Minimum number of Core machines required to be available at runtime.
- `causal_clustering.multi_dc_license`: Enable multi-data center features.
- `causal_clustering.neostore_block_id_allocation_size`: The size of the ID allocation requests Core servers will make when they run out of NEOSTORE_BLOCK IDs.
- `causal_clustering.node_id_allocation_size`: The size of the ID allocation requests Core servers will make when they run out of NODE IDs.
- `causal_clustering.node_labels_id_allocation_size`: The size of the ID allocation requests Core servers will make when they run out of NODE_LABELS IDs.
- `causal_clustering.property_id_allocation_size`: The size of the ID allocation requests Core servers will make when they run out of PROPERTY IDs.
- `causal_clustering.property_key_token_id_allocation_size`: The size of the ID allocation requests Core servers will make when they run out of PROPERTY_KEY_TOKEN IDs.
- `causal_clustering.property_key_token_name_id_allocation_size`: The size of the ID allocation requests Core servers will make when they run out of PROPERTY_KEY_TOKEN_NAME IDs.
- `causal_clustering.protocol_implementations.catchup`: Catchup protocol implementation versions that this instance will allow in negotiation as a comma-separated list.
- `causal_clustering.protocol_implementations.compression`: Network compression algorithms that this instance will allow in negotiation as a comma-separated list.
- `causal_clustering.protocol_implementations.raft`: Raft protocol implementation versions that this instance will allow in negotiation as a comma-separated list.
- `causal_clustering.pull_interval`: Interval of pulling updates from cores.
- `causal_clustering.raft_advertised_address`: Advertised hostname/IP address and port for the RAFT server.
- `causal_clustering.raft_in_queue_max_batch_bytes`: Largest batch processed by RAFT in bytes.
- `causal_clustering.raft_in_queue_max_bytes`: Maximum number of bytes in the RAFT in-queue.
- `causal_clustering.raft_listen_address`: Network interface and port for the RAFT server to listen on.
- `causal_clustering.raft_logImplementation`: RAFT log implementation.
- `causal_clustering.raft_log_prune_strategy`: RAFT log pruning strategy.
- `causal_clustering.raft_log_pruning_frequency`: RAFT log pruning frequency.
- `causal_clustering.raft_log_reader_pool_size`: RAFT log reader pool size.
- `causal_clustering.raft_log_rotation_size`: RAFT log rotation size.
- `causal_clustering.raft_membership_state_size`: The maximum file size before the membership state file is rotated (in unit of entries).
- `causal_clustering.raft_term_state_size`: The maximum file size before the term state file is rotated (in unit of entries).
- `causal_clustering.raft_vote_state_size`: The maximum file size before the vote state file is rotated (in unit of entries).
- `causal_clustering.read_replica_time_to_live`: Time To Live before read replica is considered unavailable.
- `causal_clustering.refuse_to_be_leader`: Prevents the current instance from volunteering to become Raft leader.
- `causal_clustering.relationship_group_id_allocation_size`: The size of the ID allocation requests Core servers will make when they run out of RELATIONSHIP_GROUP IDs.
- `causal_clustering.relationship_id_allocation_size`: The size of the ID allocation requests Core servers will make when they run out of RELATIONSHIP IDs.

- `causal_clustering.relationship_type_token_id_allocation_size`: The size of the ID allocation requests Core servers will make when they run out of RELATIONSHIP_TYPE_TOKEN IDs.
- `causal_clustering.relationship_type_token_name_id_allocation_size`: The size of the ID allocation requests Core servers will make when they run out of RELATIONSHIP_TYPE_TOKEN_NAME IDs.
- `causal_clustering.replicated_lock_token_state_size`: The maximum file size before the replicated lock token state file is rotated (in unit of entries).
- `causal_clustering.replication_retry_timeout_base`: The initial timeout until replication is retried.
- `causal_clustering.replication_retry_timeout_limit`: The upper limit for the exponentially incremented retry timeout.
- `causal_clustering.schema_id_allocation_size`: The size of the ID allocation requests Core servers will make when they run out of SCHEMA IDs.
- `causal_clustering.server_groups`: A list of group names for the server used when configuring load balancing and replication policies.
- `causal_clustering.ssl_policy`: Name of the SSL policy to be used by the clustering, as defined under the dbms.ssl.policy.* settings.
- `causal_clustering.state_machine_apply_max_batch_size`: The maximum number of operations to be batched during applications of operations in the state machines.
- `causal_clustering.state_machine_flush_window_size`: The number of operations to be processed before the state machines flush to disk.
- `causal_clustering.store_copy_max_retry_time_per_request`: Maximum retry time per request during store copy.
- `causal_clustering.string_block_id_allocation_size`: The size of the ID allocation requests Core servers will make when they run out of STRING_BLOCK IDs.
- `causal_clustering.transaction_advertised_address`: Advertised hostname/IP address and port for the transaction shipping server.
- `causal_clustering.transaction_listen_address`: Network interface and port for the transaction shipping server to listen on.
- `causal_clustering.unknown_address_logging_throttle`: Throttle limit for logging unknown cluster member address.
- `causal_clustering.upstream_selection_strategy`: An ordered list in descending preference of the strategy which read replicas use to choose the upstream server from which to pull transactional updates.
- `causal_clustering.user_defined_upstream_strategy`: Configuration of a user-defined upstream selection strategy.
- `cypher.default_language_version`: Set this to specify the default parser (language version).
- `cypher.forbid_exhaustive_shortestpath`: This setting is associated with performance optimization.
- `cypher.forbid_shortestpath_common_nodes`: This setting is associated with performance optimization.
- `cypher.hints_error`: Set this to specify the behavior when Cypher planner or runtime hints cannot be fulfilled.
- `cypher.lenient_create_relationship`: Set this to change the behavior for Cypher create relationship when the start or end node is missing.
- `cypher.min_replan_interval`: The minimum time between possible cypher query replanning events.
- `cypher.planner`: Set this to specify the default planner for the default language version.
- `cypher.statistics_divergence_threshold`: The threshold when a plan is considered stale.
- `db.temporal.timezone`: Database timezone for temporal functions.
- `dbms.active_database`: Name of the database to load.

- **dbms.allow_format_migration**: Whether to allow a store upgrade in case the current version of the database starts against an older store version.
- **dbms.allow_upgrade**: Whether to allow an upgrade in case the current version of the database starts against an older version.
- **dbms.backup.address**: Listening server for online backups.
- **dbms.backup.enabled**: Enable support for running online backups.
- **dbms.backup.ssl_policy**: Name of the SSL policy to be used by backup, as defined under the dbms.ssl.policy.* settings.
- **dbms.checkpoint**: Configures the general policy for when check-points should occur.
- **dbms.checkpoint.interval.time**: Configures the time interval between check-points.
- **dbms.checkpoint.interval.tx**: Configures the transaction interval between check-points.
- **dbms.checkpoint.iops.limit**: Limit the number of IOs the background checkpoint process will consume per second.
- **dbms.config.strict_validation**: A strict configuration validation will prevent the database from starting up if unknown configuration options are specified in the neo4j settings namespace (such as dbms., ha., cypher., etc).
- **dbms.connector.bolt.enabled**: Enable this connector.
- **dbms.connector.http.enabled**: Enable this connector.
- **dbms.connector.https.enabled**: Enable this connector.
- **dbms.connectors.default_advertised_address**: Default hostname or IP address the server uses to advertise itself to its connectors.
- **dbms.connectors.default_listen_address**: Default network interface to listen for incoming connections.
- **dbms.db.timezone**: Database timezone.
- **dbms.directories.certificates**: Directory for storing certificates to be used by Neo4j for TLS connections.
- **dbms.directories.data**: Path of the data directory.
- **dbms.directories.import**: Sets the root directory for file URLs used with the Cypher `LOAD CSV` clause.
- **dbms.directories.lib**: Path of the lib directory.
- **dbms.directories.logs**: Path of the logs directory.
- **dbms.directories.metrics**: The target location of the CSV files: a path to a directory wherein a CSV file per reported field will be written.
- **dbms.directories.plugins**: Location of the database plugin directory.
- **dbms.directories.run**: Path of the run directory.
- **dbms.directories.tx_log**: Location where Neo4j keeps the logical transaction logs.
- **dbms.filewatcher.enabled**: Allows the enabling or disabling of the file watcher service.
- **dbms.ids.reuse.types.override**: Specified names of id types (comma separated) that should be reused.
- **dbms.import.csv.buffer_size**: The size of the internal buffer in bytes used by `LOAD CSV`.
- **dbms.import.csv.legacy_quoteEscaping**: Selects whether to conform to the standard <https://tools.ietf.org/html/rfc4180> for interpreting escaped quotation characters in CSV files loaded using `LOAD CSV`.
- **dbms.index.default_schema_provider**: Index provider to use for newly created schema indexes.
- **dbms.index.fulltext.default_analyzer**: The name of the analyzer that the fulltext indexes should use

by default.

- **dbms.index.fulltext.eventually_consistent**: Whether or not fulltext indexes should be eventually consistent by default or not.
- **dbms.index.fulltext.eventually_consistent_index_update_queue_max_length**: The eventually_consistent mode of the fulltext indexes works by queueing up index updates to be applied later in a background thread.
- **dbms.index_sampling.background_enabled**: Enable or disable background index sampling.
- **dbms.index_sampling.buffer_size**: Size of buffer used by index sampling.
- **dbms.index_sampling.sample_size_limit**: Index sampling chunk size limit.
- **dbms.index_sampling.update_percentage**: Percentage of index updates of total index size required before sampling of a given index is triggered.
- **dbms.index_searcher_cache_size**: The maximum number of open Lucene index searchers.
- **dbms.jvm.additional**: Additional JVM arguments.
- **dbms.lock.acquisition.timeout**: The maximum time interval within which lock should be acquired.
- **dbms.logs.debug.level**: Debug log level threshold.
- **dbms.logs.debug.path**: Path to the debug log file.
- **dbms.logs.debug.rotation.delay**: Minimum time interval after last rotation of the debug log before it may be rotated again.
- **dbms.logs.debug.rotation.keep_number**: Maximum number of history files for the debug log.
- **dbms.logs.debug.rotation.size**: Threshold for rotation of the debug log.
- **dbms.logs.gc.enabled**: Enable GC Logging.
- **dbms.logs.gc.options**: GC Logging Options.
- **dbms.logs.gc.rotation.keep_number**: Number of GC logs to keep.
- **dbms.logs.gc.rotation.size**: Size of each GC log that is kept.
- **dbms.logs.http.enabled**: Enable HTTP request logging.
- **dbms.logs.http.path**: Path to HTTP request log.
- **dbms.logs.http.rotation.keep_number**: Number of HTTP logs to keep.
- **dbms.logs.http.rotation.size**: Size of each HTTP log that is kept.
- **dbms.logs.query.allocation_logging_enabled**: Log allocated bytes for the executed queries being logged.
- **dbms.logs.query.enabled**: Log executed queries that take longer than the configured threshold, dbms.logs.query.threshold.
- **dbms.logs.query.page_logging_enabled**: Log page hits and page faults for the executed queries being logged.
- **dbms.logs.query.parameter_logging_enabled**: Log parameters for the executed queries being logged.
- **dbms.logs.query.path**: Path to the query log file.
- **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 - provided query logging is enabled.
- **dbms.logs.query.time_logging_enabled**: Log detailed time information for the executed queries being logged.

- `dbms.logs.security.level`: Security log level threshold.
- `dbms.logs.security.path`: Path to the security log file.
- `dbms.logs.security.rotation.delay`: Minimum time interval after last rotation of the security log before it may be rotated again.
- `dbms.logs.security.rotation.keep_number`: Maximum number of history files for the security log.
- `dbms.logs.security.rotation.size`: Threshold for rotation of the security log.
- `dbms.logs.timezone`: Database logs timezone.
- `dbms.logs.user.path`: Path to the user log file.
- `dbms.logs.user.rotation.delay`: Minimum time interval after last rotation of the user log before it may be rotated again.
- `dbms.logs.user.rotation.keep_number`: Maximum number of history files for the user log.
- `dbms.logs.user.rotation.size`: Threshold for rotation of the user log.
- `dbms.logs.user.stdout_enabled`: Send user logs to the process stdout.
- `dbms.memory.heap.initial_size`: Initial heap size.
- `dbms.memory.heap.max_size`: Maximum heap size.
- `dbms.memory.pagecache.size`: The amount of memory to use for mapping the store files, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g').
- `dbms.memory.pagecache.swapper`: Specify which page swapper to use for doing paged IO.
- `dbms.mode`: Configure the operating mode of the database — 'SINGLE' for stand-alone operation, 'HA' for operating as a member in an HA cluster, 'ARBITER' for a cluster member with no database in an HA cluster, 'CORE' for operating as a core member of a Causal Cluster, or 'READ_REPLICA' for operating as a read replica member of a Causal Cluster.
- `dbms.netty.ssl.provider`: Netty SSL provider.
- `dbms.procedures.kill_query_verbose`: Specifies whether or not dbms.killQueries produces a verbose output, with information about which queries were not found.
- `dbms.query_cache_size`: The number of Cypher query execution plans that are cached.
- `dbms.read_only`: Only allow read operations from this Neo4j instance.
- `dbms.record_format`: Database record format.
- `dbms.relationship_grouping_threshold`: Relationship count threshold for considering a node to be dense.
- `dbms.rest.transaction.idle_timeout`: Timeout for idle transactions in the REST endpoint.
- `dbms.security.allow_csv_import_from_file_urls`: Determines if Cypher will allow using file URLs when loading data using `LOAD CSV`.
- `dbms.security.auth_cache_max_capacity`: The maximum capacity for authentication and authorization caches (respectively).
- `dbms.security.auth_cache_ttl`: The time to live (TTL) for cached authentication and authorization info when using external auth providers (LDAP or plugin).
- `dbms.security.auth_cache_use_ttl`: Enable time-based eviction of the authentication and authorization info cache for external auth providers (LDAP or plugin).
- `dbms.security.auth_enabled`: Enable auth requirement to access Neo4j.
- `dbms.security.auth_lock_time`: The amount of time user account should be locked after a configured number of unsuccessful authentication attempts.
- `dbms.security.auth_max_failed_attempts`: The maximum number of unsuccessful authentication attempts before imposing a user lock for the configured amount of time. The locked out user will not be able to log in until the lock period expires, even if correct credentials are provided.

- `dbms.security.auth_provider`: The authentication and authorization provider that contains both the users and roles.
- `dbms.security.causal_clustering_status_auth_enabled`: Require authorization for access to the Causal Clustering status endpoints.
- `dbms.security.ha_status_auth_enabled`: Require authorization for access to the HA status endpoints.
- `dbms.security.http_access_control_allow_origin`: Value of the Access-Control-Allow-Origin header sent over any HTTP or HTTPS connector.
- `dbms.security.http_authorization_classes`: Comma-separated list of custom security rules for Neo4j to use.
- `dbms.security.http_strict_transport_security`: Value of the HTTP Strict-Transport-Security (HSTS) response header.
- `dbms.security.ldap.authentication.cache_enabled`: Determines if the result of authentication via the LDAP server should be cached or not.
- `dbms.security.ldap.authentication.mechanism`: LDAP authentication mechanism.
- `dbms.security.ldap.authentication.use_samaccountname`: Perform authentication with sAMAccountName instead of DN. Using this setting 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.
- `dbms.security.ldap.authentication.user_dn_template`: LDAP user DN template.
- `dbms.security.ldap.authorization.group_membership_attributes`: A list of attribute names on a user object that contains groups to be used for mapping to roles when LDAP authorization is enabled.
- `dbms.security.ldap.authorization.group_to_role_mapping`: An authorization mapping from LDAP group names to Neo4j role names.
- `dbms.security.ldap.authorization.system_password`: An LDAP system account password to use for authorization searches when `dbms.security.ldap.authorization.use_system_account` is `true`.
- `dbms.security.ldap.authorization.system_username`: An LDAP system account username to use for authorization searches when `dbms.security.ldap.authorization.use_system_account` is `true`.
- `dbms.security.ldap.authorization.use_system_account`: Perform LDAP search for authorization info using a system account instead of the user's own account.

If this is set to `false` (default), the search for group membership will be performed directly after authentication using the LDAP context bound with the user's own account. *

`dbms.security.ldap.authorization.user_search_base`: The name of the base object or named context to search for user objects when LDAP authorization is enabled. *

`dbms.security.ldap.authorization.user_search_filter`: The LDAP search filter to search for a user principal when LDAP authorization is enabled. * `dbms.security.ldap.connection_timeout`: The timeout for establishing an LDAP connection. *

`dbms.security.ldap.host`: URL of LDAP server to use for authentication and authorization. *

`dbms.security.ldap.read_timeout`: The timeout for an LDAP read request (i.e. *)

`dbms.security.ldap.referral`: The LDAP referral behavior when creating a connection. *

`dbms.security.ldap.use_starttls`: Use secure communication with the LDAP server using opportunistic TLS. *

`dbms.security.log_successful_authentication`: Set to log successful authentication events to the security log. *

`dbms.security.procedures.default_allowed`: The default role that can execute all procedures and user-defined functions that are not covered by the `dbms.security.procedures.roles` setting. *

`dbms.security.procedures.roles`: This provides a finer level of control over which roles can execute procedures than the `dbms.security.procedures.default_allowed` setting. *

`dbms.security.procedures.unrestricted`: A list of procedures and user defined functions (comma separated) that are allowed full access to the database. *

`dbms.security.procedures.whitelist`: A list of procedures (comma separated) that are to be loaded. *

authorization mapping for property level access for roles. * `dbms.security.property_level.enabled`: Set to true to enable property level security. * `dbms.shutdown_transaction_end_timeout`: The maximum amount of time to wait for running transactions to complete before allowing initiated database shutdown to continue. * `dbms.ssl.policy.<policyname>.allow_key_generation`: Allows the generation of a private key and associated self-signed certificate. * `dbms.ssl.policy.<policyname>.base_directory`: The mandatory base directory for cryptographic objects of this policy. * `dbms.ssl.policy.<policyname>.ciphers`: Restrict allowed ciphers. * `dbms.ssl.policy.<policyname>.client_auth`: Client authentication stance. * `dbms.ssl.policy.<policyname>.private_key`: Private PKCS#8 key in PEM format. * `dbms.ssl.policy.<policyname>.public_certificate`: X.509 certificate (chain) of this server in PEM format. * `dbms.ssl.policy.<policyname>.revoked_dir`: Path to directory of CRLs (Certificate Revocation Lists) in PEM format. * `dbms.ssl.policy.<policyname>.tls_versions`: Restrict allowed TLS protocol versions. * `dbms.ssl.policy.<policyname>.trust_all`: Makes this policy trust all remote parties. * `dbms.ssl.policy.<policyname>.trusted_dir`: Path to directory of X.509 certificates in PEM format for trusted parties. * `dbms.ssl.policy.<policyname>.verify_hostname`: When true, this node will verify the hostname of every other instance it connects to by comparing the address it used to connect with it and the patterns described in the remote hosts public certificate Subject Alternative Names. * `dbms.threads.worker_count`: Number of Neo4j worker threads, your OS might enforce a lower limit than the maximum value specified here. * `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.bookmark_ready_timeout`: The maximum amount of time to wait for the database state represented by the bookmark. * `dbms.transaction.monitor.check.interval`: Configures the time interval between transaction monitor checks. * `dbms.transaction.timeout`: The maximum time interval of a transaction within which it should be completed. * `dbms.tx_log.rotation.retention_policy`: Make Neo4j keep the logical transaction logs for being able to backup the database. * `dbms.tx_log.rotation.size`: Specifies at which file size the logical log will auto-rotate. * `dbms.tx_state.max_off_heap_memory`: The maximum amount of off-heap memory that can be used to store transaction state data; it's a total amount of memory shared across all active transactions. * `dbms.tx_state.memory_allocation`: Defines whether memory for transaction state should be allocated on- or off-heap. * `dbms.tx_state.off_heap.block_cache_size`: Defines the size of the off-heap memory blocks cache. * `dbms.tx_state.off_heap.max_cacheable_block_size`: Defines the maximum size of an off-heap memory block that can be cached to speed up allocations for transaction state data. * `dbms.udc.enabled`: Enable the UDC extension. * `dbms.unmanaged_extension_classes`: Comma-separated list of <classname>=<mount point> for unmanaged extensions. * `dbms.windows_service_name`: Name of the Windows Service. * `ha.allow_init_cluster`: Whether to allow this instance to create a cluster if unable to join. * `ha.branched_data_copying_strategy`: Strategy for how to order handling of branched data on slaves and copying of the store from the master. * `ha.branched_data_policy`: Policy for how to handle branched data. * `ha.broadcast_timeout`: Timeout for broadcasting values in cluster. * `ha.configuration_timeout`: Timeout for waiting for configuration from an existing cluster member during cluster join. * `ha.data_chunk_size`: Max size of the data chunks that flows between master and slaves in HA. * `ha.default_timeout`: Default timeout used for clustering timeouts. * `ha.election_timeout`: Timeout for waiting for other members to finish a role election. * `ha.heartbeat_interval`: How often heartbeat messages should be sent. * `ha.heartbeat_timeout`: How long to wait for heartbeats from other instances before marking them as suspects for failure. * `ha.host.coordination`: Host and port to bind the cluster management communication. * `ha.host.data`: Hostname and port to bind the HA server. * `ha.initial_hosts`: A comma-separated list of other members of the cluster to join. * `ha.internal_role_switch_timeout`: Timeout for waiting for internal conditions during state switch, like for transactions to complete, before switching to master or slave. * `ha.join_timeout`: Timeout for joining a cluster. * `ha.learn_timeout`: Timeout for learning values. * `ha.leave_timeout`: Timeout for waiting for cluster leave to finish. * `ha.max_acceptors`: Maximum number of servers to involve when agreeing to membership changes. * `ha.max_channels_per_slave`: Maximum number of connections a slave can have to the master. * `ha.paxos_timeout`: Default value for all Paxos timeouts. * `ha.phase1_timeout`: Timeout for Paxos phase 1. * `ha.phase2_timeout`: Timeout for Paxos phase 2. * `ha.pull_batch_size`: Size of batches of transactions applied on slaves when pulling from master. * `ha.pull_interval`: Interval of pulling updates from master. * `ha.role_switch_timeout`: Timeout for request threads waiting for instance to become master or slave. * `ha.server_id`: Id for a cluster instance. * `ha.slave_lock_timeout`: Timeout for taking remote (write) locks on slaves. * `ha.slave_only`: Whether this instance should only

participate as slave in cluster. * `ha.slave_read_timeout`: How long a slave will wait for response from master before giving up. * `ha.tx_push_factor`: The amount of slaves the master will ask to replicate a committed transaction. * `ha.tx_push_strategy`: Push strategy of a transaction to a slave during commit. * `https.ssl_policy`: SSL policy name. * `metrics.bolt.messages.enabled`: Enable reporting metrics about Bolt Protocol message processing. * `metrics.csv.enabled`: Set to `true` to enable exporting metrics to CSV files. * `metrics.csv.interval`: The reporting interval for the CSV files. * `metrics.csv.rotation.keep_number`: Maximum number of history files for the csv files. * `metrics.csv.rotation.size`: The file size in bytes at which the csv files will auto-rotate. * `metrics.cypher.replanning.enabled`: Enable reporting metrics about number of occurred replanning events. * `metrics.enabled`: The default enablement value for all the supported metrics. * `metrics.graphite.enabled`: Set to `true` to enable exporting metrics to Graphite. * `metrics.graphite.interval`: The reporting interval for Graphite. * `metrics.graphite.server`: The hostname or IP address of the Graphite server. * `metrics.jvm.buffers.enabled`: Enable reporting metrics about the buffer pools. * `metrics.jvm.gc.enabled`: Enable reporting metrics about the duration of garbage collections. * `metrics.jvm.memory.enabled`: Enable reporting metrics about the memory usage. * `metrics.jvm.threads.enabled`: Enable reporting metrics about the current number of threads running. * `metrics.neo4j.causal_clustering.enabled`: Enable reporting metrics about Causal Clustering mode. * `metrics.neo4j.checkpointing.enabled`: Enable reporting metrics about Neo4j check pointing. * `metrics.neo4j.cluster.enabled`: Enable reporting metrics about HA cluster info. * `metrics.neo4j.counts.enabled`: Enable reporting metrics about approximately how many entities are in the database. * `metrics.neo4j.enabled`: The default enablement value for all Neo4j specific support metrics. * `metrics.neo4j.logrotation.enabled`: Enable reporting metrics about the Neo4j log rotation. * `metrics.neo4j.network.enabled`: Enable reporting metrics about the network usage. * `metrics.neo4j.pagecache.enabled`: Enable reporting metrics about the Neo4j page cache. * `metrics.neo4j.server.enabled`: Enable reporting metrics about Server threading info. * `metrics.neo4j.tx.enabled`: Enable reporting metrics about transactions. * `metrics.prefix`: A common prefix for the reported metrics field names. * `metrics.prometheus.enabled`: Set to `true` to enable the Prometheus endpoint. * `metrics.prometheus.endpoint`: The hostname and port to use as Prometheus endpoint. * `tools.consistency_checker.check_graph`: This setting is deprecated. See commandline arguments for neo4j-admin check-consistency instead. * `tools.consistency_checker.check_indexes`: This setting is deprecated. See commandline arguments for neo4j-admin check-consistency instead. * `tools.consistency_checker.check_label_scan_store`: This setting is deprecated. See commandline arguments for neo4j-admin check-consistency instead. * `tools.consistency_checker.check_property_owners`: This setting is deprecated. See commandline arguments for neo4j-admin check-consistency instead.

Table 37. bolt.ssl_policy

Description	Specify the SSL policy to use for the encrypted bolt connections.
Valid values	bolt.ssl_policy is a string
Default value	<code>legacy</code>

Table 38. browser.allow_outgoing_connections

Description	Configure the policy for outgoing Neo4j Browser connections.
Valid values	browser.allow_outgoing_connections is a boolean
Default value	<code>true</code>

Table 39. browser.credential_timeout

Description	Configure the Neo4j Browser to time out logged in users after this idle period. Setting this to 0 indicates no limit.
Valid values	browser.credential_timeout is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	<code>0s</code>

Table 40. browser.post_connect_cmd

Description	Commands to be run when Neo4j Browser successfully connects to this server. Separate multiple commands with semi-colon.
Valid values	browser.post_connect_cmd is a string
Default value	

Table 41. `browser.remote_content_hostname_whitelist`

Description	Whitelist of hosts for the Neo4j Browser to be allowed to fetch content from.
Valid values	browser.remote_content_hostname_whitelist is a string
Default value	guides.neo4j.com,localhost

Table 42. `browser.retain_connection_credentials`

Description	Configure the Neo4j Browser to store or not store user credentials.
Valid values	browser.retain_connection_credentials is a boolean
Default value	true

Table 43. `causal_clustering.array_block_id_allocation_size`

Description	The size of the ID allocation requests Core servers will make when they run out of ARRAY_BLOCK IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.
Valid values	causal_clustering.array_block_id_allocation_size is an integer
Default value	1024

Table 44. `causal_clustering.catch_up_client_inactivity_timeout`

Description	The catch up protocol times out if the given duration elapses with no network activity. Every message received by the client from the server extends the time out duration.
Valid values	causal_clustering.catch_up_client_inactivity_timeout is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	600s

Table 45. `causal_clustering.catchup_batch_size`

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

Table 46. `causal_clustering.cluster_allow_reads_on_followers`

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

Table 47. `causal_clustering.cluster_routing_ttl`

Description	How long drivers should cache the data from the <code>dbms.cluster.routing.getServer()</code> procedure.
Valid values	causal_clustering.cluster_routing_ttl is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's') which is minimum PT1S
Default value	300s

Table 48. causal_clustering.cluster_topology_refresh

Description	Time between scanning the cluster to refresh current server's view of topology.
Valid values	causal_clustering.cluster_topology_refresh is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's') which is minimum PT1S
Default value	5s

Table 49. causal_clustering.connect-randomly-to-server-group

Description	Comma separated list of groups to be used by the connect-randomly-to-server-group selection strategy. The connect-randomly-to-server-group strategy is used if the list of strategies (causal_clustering.upstream_selection_strategy) includes the value connect-randomly-to-server-group .
Valid values	causal_clustering.connect-randomly-to-server-group is a list separated by "," where items are a string
Default value	[]

Table 50. causal_clustering.database

Description	The name of the database being hosted by this server instance. This configuration setting may be safely ignored unless deploying a multicluster. Instances may be allocated to distinct sub-clusters by assigning them distinct database names using this setting. For instance if you had 6 instances you could form 2 sub-clusters by assigning half the database name "foo", half the name "bar". The setting value must match exactly between members of the same sub-cluster. This setting is a one-off: once an instance is configured with a database name it may not be changed in future without using neo4j-admin unbind.
Valid values	causal_clustering.database is a string
Default value	default

Table 51. causal_clustering.disable_middleware_logging

Description	Prevents the network middleware from dumping its own logs. Defaults to true.
Valid values	causal_clustering.disable_middleware_logging is a boolean
Default value	true

Table 52. causal_clustering.discovery_advertised_address

Description	Advertised cluster member discovery management communication.
Valid values	an advertised socket address
Default value	localhost:5000

Table 53. causal_clustering.discovery_listen_address

Description	Host and port to bind the cluster member discovery management communication.
Valid values	a listen socket address
Default value	127.0.0.1:5000

Table 54. causal_clustering.discovery_type

Description	Configure the discovery type used for cluster name resolution.
Valid values	causal_clustering.discovery_type is one of DNS , LIST , SRV , K8S
Default value	LIST

Table 55. causal_clustering.enable_pre_voting

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

Default value	false
----------------------	-------

Table 56. causal_clustering.expected_core_cluster_size

Description	Expected number of Core machines in the cluster before startup.
Valid values	causal_clustering.expected_core_cluster_size is an integer
Default value	3
Deprecated	The causal_clustering.expected_core_cluster_size configuration setting has been deprecated.
Replaced by	causal_clustering.minimum_core_cluster_size_atFormation, causal_clustering.minimum_core_cluster_size_atRuntime

Table 57. causal_clustering.global_session_tracker_state_size

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

Table 58. causal_clustering.handshake_timeout

Description	Time out for protocol negotiation handshake.
Valid values	causal_clustering.handshake_timeout is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	20s

Table 59. causal_clustering.id_alloc_state_size

Description	The maximum file size before the ID allocation file is rotated (in unit of entries)
Valid values	causal_clustering.id_alloc_state_size is an integer
Default value	1000

Table 60. causal_clustering.in_flight_cache.max_bytes

Description	The maximum number of bytes in the in-flight cache.
Valid values	causal_clustering.in_flight_cache.max_bytes is a byte size (valid multipliers are k, m, g, K, M, G)
Default value	2147483648

Table 61. causal_clustering.in_flight_cache.max_entries

Description	The maximum number of entries in the in-flight cache.
Valid values	causal_clustering.in_flight_cache.max_entries is an integer
Default value	1024

Table 62. causal_clustering.in_flight_cache.type

Description	Type of in-flight cache.
Valid values	causal_clustering.in_flight_cache.type is one of NONE, CONSECUTIVE, UNBOUNDED
Default value	CONSECUTIVE

Table 63. causal_clustering.initial_discovery_members

Description	A comma-separated list of other members of the cluster to join.
Valid values	causal_clustering.initial_discovery_members is a list separated by "," where items are an advertised socket address

Table 64. causal_clustering.join_up_timeout

Description	Time out for a new member to catch up.
Valid values	causal_clustering.join_up_timeout is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	600s

Table 65. causal_clustering.kubernetes.address

Description	Address for Kubernetes API.
Valid values	causal_clustering.kubernetes.address is an advertised socket address
Default value	kubernetes.default.svc:443

Table 66. causal_clustering.kubernetes.ca_crt

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

Table 67. causal_clustering.kubernetes.label_selector

Description	LabelSelector for Kubernetes API.
Valid values	causal_clustering.kubernetes.label_selector is a string

Table 68. causal_clustering.kubernetes.namespace

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

Table 69. causal_clustering.kubernetes.service_port_name

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

Table 70. causal_clustering.kubernetes.token

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

Table 71. causal_clustering.label_token_id_allocation_size

Description	The size of the ID allocation requests Core servers will make when they run out of LABEL_TOKEN IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.
Valid values	causal_clustering.label_token_id_allocation_size is an integer
Default value	32

Table 72. causal_clustering.label_token_name_id_allocation_size

Description	The size of the ID allocation requests Core servers will make when they run out of LABEL_TOKEN_NAME IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.
Valid values	causal_clustering.label_token_name_id_allocation_size is an integer

Default value	1024
---------------	------

Table 73. causal_clustering.last_applied_state_size

Description	The maximum file size before the storage file is rotated (in unit of entries)
Valid values	causal_clustering.last_applied_state_size is an integer
Default value	1000

Table 74. causal_clustering.leader_election_timeout

Description	The time limit within which a new leader election will occur if no messages are received.
Valid values	causal_clustering.leader_election_timeout is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	7s

Table 75. causal_clustering.load_balancing.config

Description	The configuration must be valid for the configured plugin and usually exists under matching subkeys, e.g. ..config.server_policies.* This is just a top-level placeholder for the plugin-specific configuration.
Valid values	causal_clustering.load_balancing.config is a string
Default value	

Table 76. causal_clustering.load_balancing.plugin

Description	The load balancing plugin to use.
Valid values	causal_clustering.load_balancing.plugin is a string
Default value	server_policies

Table 77. causal_clustering.load_balancing.shuffle

Description	Enables shuffling of the returned load balancing result.
Valid values	causal_clustering.load_balancing.shuffle is a boolean
Default value	true

Table 78. causal_clustering.log_shipping_max_lag

Description	The maximum lag allowed before log shipping pauses (in unit of entries)
Valid values	causal_clustering.log_shipping_max_lag is an integer
Default value	256

Table 79. causal_clustering.middleware_logging.level

Description	The level of middleware logging.
Valid values	causal_clustering.middleware_logging.level is an integer
Default value	500

Table 80. causal_clustering.minimum_core_cluster_size_atFormation

Description	Minimum number of Core machines in the cluster at formation. The expected_core_cluster_size setting is used when bootstrapping the cluster on first formation. A cluster will not form without the configured amount of cores and this should in general be configured to the full and fixed amount. When using multi-clustering (configuring multiple distinct database names across core hosts), this setting is used to define the minimum size of each sub-cluster at formation.
Valid values	causal_clustering.minimum_core_cluster_size_atFormation is an integer which is minimum 2

Default value	3
----------------------	---

Table 81. causal_clustering.minimum_core_cluster_size_at_runtime

Description	Minimum number of Core machines required to be available at runtime. The consensus group size (core machines successfully voted into the Raft) can shrink and grow dynamically but bounded on the lower end at this number. The intention is in almost all cases for users to leave this setting alone. If you have 5 machines then you can survive failures down to 3 remaining, e.g. with 2 dead members. The three remaining can still vote another replacement member in successfully up to a total of 6 (2 of which are still dead) and then after this, one of the superfluous dead members will be immediately and automatically voted out (so you are left with 5 members in the consensus group, 1 of which is currently dead). Operationally you can now bring the last machine up by bringing in another replacement or repairing the dead one. When using multi-clustering (configuring multiple distinct database names across core hosts), this setting is used to define the minimum size of each sub-cluster at runtime.
Valid values	causal_clustering.minimum_core_cluster_size_at_runtime is an integer which is minimum 2
Default value	3

Table 82. causal_clustering.multi_dc_license

Description	Enable multi-data center features. Requires appropriate licensing.
Valid values	causal_clustering.multi_dc_license is a boolean
Default value	false

Table 83. causal_clustering.neostore_block_id_allocation_size

Description	The size of the ID allocation requests Core servers will make when they run out of NEOSTORE_BLOCK IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.
Valid values	causal_clustering.neostore_block_id_allocation_size is an integer
Default value	1024

Table 84. causal_clustering.node_id_allocation_size

Description	The size of the ID allocation requests Core servers will make when they run out of NODE IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.
Valid values	causal_clustering.node_id_allocation_size is an integer
Default value	1024

Table 85. causal_clustering.node_labels_id_allocation_size

Description	The size of the ID allocation requests Core servers will make when they run out of NODE_LABELS IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.
Valid values	causal_clustering.node_labels_id_allocation_size is an integer
Default value	1024

Table 86. causal_clustering.property_id_allocation_size

Description	The size of the ID allocation requests Core servers will make when they run out of PROPERTY IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.
Valid values	causal_clustering.property_id_allocation_size is an integer
Default value	1024

Table 87. causal_clustering.property_key_token_id_allocation_size

Description	The size of the ID allocation requests Core servers will make when they run out of PROPERTY_KEY_TOKEN IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.
Valid values	causal_clustering.property_key_token_id_allocation_size is an integer
Default value	32

Table 88. causal_clustering.property_key_token_name_id_allocation_size

Description	The size of the ID allocation requests Core servers will make when they run out of PROPERTY_KEY_TOKEN_NAME IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.
Valid values	causal_clustering.property_key_token_name_id_allocation_size is an integer
Default value	1024

Table 89. causal_clustering.protocol_implementations.catchup

Description	Catchup protocol implementation versions that this instance will allow in negotiation as a comma-separated list. Order is not relevant: the greatest value will be preferred. An empty list will allow all supported versions.
Valid values	causal_clustering.protocol_implementations.catchup is a list separated by "," where items are an integer
Default value	[]

Table 90. causal_clustering.protocol_implementations.compression

Description	Network compression algorithms that this instance will allow in negotiation as a comma-separated list. Listed in descending order of preference for incoming connections. An empty list implies no compression. For outgoing connections this merely specifies the allowed set of algorithms and the preference of the remote peer will be used for making the decision. Allowable values: [Gzip,Snappy,Snappy_validating,LZ4,LZ4_high_compression,LZ_validating,LZ4_high_compression_validating]
Valid values	causal_clustering.protocol_implementations.compression is a list separated by "," where items are a string
Default value	[]

Table 91. causal_clustering.protocol_implementations.raft

Description	Raft protocol implementation versions that this instance will allow in negotiation as a comma-separated list. Order is not relevant: the greatest value will be preferred. An empty list will allow all supported versions.
Valid values	causal_clustering.protocol_implementations.raft is a list separated by "," where items are an integer
Default value	[]

Table 92. causal_clustering.pull_interval

Description	Interval of pulling updates from cores.
Valid values	causal_clustering.pull_interval is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	1s

Table 93. causal_clustering.raft_advertised_address

Description	Advertised hostname/IP address and port for the RAFT server.
Valid values	an advertised socket address
Default value	localhost:7000

Table 94. causal_clustering.raft_in_queue_max_batch_bytes

Description	Largest batch processed by RAFT in bytes.
Valid values	causal_clustering.raft_in_queue_max_batch_bytes is a byte size (valid multipliers are k, m, g, K, M, G)
Default value	8388608

Table 95. causal_clustering.raft_in_queue_max_bytes

Description	Maximum number of bytes in the RAFT in-queue.
Valid values	causal_clustering.raft_in_queue_max_bytes is a byte size (valid multipliers are k, m, g, K, M, G)
Default value	2147483648

Table 96. causal_clustering.raft_listen_address

Description	Network interface and port for the RAFT server to listen on.
Valid values	a listen socket address
Default value	127.0.0.1:7000

Table 97. causal_clustering.raft_logImplementation

Description	RAFT log implementation.
Valid values	causal_clustering.raft_logImplementation is a string
Default value	SEGMENTED

Table 98. causal_clustering.raft_log_prune_strategy

Description	RAFT log pruning strategy.
Valid values	causal_clustering.raft_log_prune_strategy is a string
Default value	1g_size

Table 99. causal_clustering.raft_log_pruning_frequency

Description	RAFT log pruning frequency.
Valid values	causal_clustering.raft_log_pruning_frequency is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	600s

Table 100. causal_clustering.raft_log_reader_pool_size

Description	RAFT log reader pool size.
Valid values	causal_clustering.raft_log_reader_pool_size is an integer
Default value	8

Table 101. causal_clustering.raft_log_rotation_size

Description	RAFT log rotation size.
Valid values	causal_clustering.raft_log_rotation_size is a byte size (valid multipliers are k, m, g, K, M, G) which is minimum 1024
Default value	262144000

Table 102. causal_clustering.raft_membership_state_size

Description	The maximum file size before the membership state file is rotated (in unit of entries)
Valid values	causal_clustering.raft_membership_state_size is an integer

Default value	1000
----------------------	------

Table 103. causal_clustering.raft_term_state_size

Description	The maximum file size before the term state file is rotated (in unit of entries)
Valid values	causal_clustering.raft_term_state_size is an integer
Default value	1000

Table 104. causal_clustering.raft_vote_state_size

Description	The maximum file size before the vote state file is rotated (in unit of entries)
Valid values	causal_clustering.raft_vote_state_size is an integer
Default value	1000

Table 105. causal_clustering.read_replica_time_to_live

Description	Time To Live before read replica is considered unavailable.
Valid values	causal_clustering.read_replica_time_to_live is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's') which is minimum PT1M
Default value	60s

Table 106. causal_clustering.refuse_to_be_leader

Description	Prevents the current instance from volunteering to become Raft leader. Defaults to false, and should only be used in exceptional circumstances by expert users. Using this can result in reduced availability for the cluster.
Valid values	causal_clustering.refuse_to_be_leader is a boolean
Default value	false

Table 107. causal_clustering.relationship_group_id_allocation_size

Description	The size of the ID allocation requests Core servers will make when they run out of RELATIONSHIP_GROUP IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.
Valid values	causal_clustering.relationship_group_id_allocation_size is an integer
Default value	1024

Table 108. causal_clustering.relationship_id_allocation_size

Description	The size of the ID allocation requests Core servers will make when they run out of RELATIONSHIP IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.
Valid values	causal_clustering.relationship_id_allocation_size is an integer
Default value	1024

Table 109. causal_clustering.relationship_type_token_id_allocation_size

Description	The size of the ID allocation requests Core servers will make when they run out of RELATIONSHIP_TYPE_TOKEN IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.
Valid values	causal_clustering.relationship_type_token_id_allocation_size is an integer
Default value	32

Table 110. causal_clustering.relationship_type_token_name_id_allocation_size

Description	The size of the ID allocation requests Core servers will make when they run out of RELATIONSHIP_TYPE_TOKEN_NAME IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.
Valid values	causal_clustering.relationship_type_token_name_id_allocation_size is an integer
Default value	1024

Table 111. causal_clustering.replicated_lock_token_state_size

Description	The maximum file size before the replicated lock token state file is rotated (in unit of entries)
Valid values	causal_clustering.replicated_lock_token_state_size is an integer
Default value	1000

Table 112. causal_clustering.replication_retry_timeout_base

Description	The initial timeout until replication is retried. The timeout will increase exponentially.
Valid values	causal_clustering.replication_retry_timeout_base is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	10s

Table 113. causal_clustering.replication_retry_timeout_limit

Description	The upper limit for the exponentially incremented retry timeout.
Valid values	causal_clustering.replication_retry_timeout_limit is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	60s

Table 114. causal_clustering.schema_id_allocation_size

Description	The size of the ID allocation requests Core servers will make when they run out of SCHEMA IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.
Valid values	causal_clustering.schema_id_allocation_size is an integer
Default value	1024

Table 115. causal_clustering.server_groups

Description	A list of group names for the server used when configuring load balancing and replication policies.
Valid values	causal_clustering.server_groups is a list separated by "," where items are a string
Default value	[]

Table 116. causal_clustering.ssl_policy

Description	Name of the SSL policy to be used by the clustering, as defined under the dbms.ssl.policy.* settings. If no policy is configured then the communication will not be secured.
Valid values	causal_clustering.ssl_policy is a string

Table 117. causal_clustering.state_machine_apply_max_batch_size

Description	The maximum number of operations to be batched during applications of operations in the state machines.
Valid values	causal_clustering.state_machine_apply_max_batch_size is an integer
Default value	16

Table 118. causal_clustering.state_machine_flush_window_size

Description	The number of operations to be processed before the state machines flush to disk.
Valid values	causal_clustering.state_machine_flush_window_size is an integer
Default value	4096

Table 119. causal_clustering.store_copy_max_retry_time_per_request

Description	Maximum retry time per request during store copy. Regular store files and indexes are downloaded in separate requests during store copy. This configures the maximum time failed requests are allowed to resend.
Valid values	causal_clustering.store_copy_max_retry_time_per_request is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	1200s

Table 120. causal_clustering.string_block_id_allocation_size

Description	The size of the ID allocation requests Core servers will make when they run out of STRING_BLOCK IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.
Valid values	causal_clustering.string_block_id_allocation_size is an integer
Default value	1024

Table 121. causal_clustering.transaction_advertised_address

Description	Advertised hostname/IP address and port for the transaction shipping server.
Valid values	an advertised socket address
Default value	localhost:6000

Table 122. causal_clustering.transaction_listen_address

Description	Network interface and port for the transaction shipping server to listen on. Please note that it is also possible to run the backup client against this port so always limit access to it via the firewall and configure an ssl policy.
Valid values	a listen socket address
Default value	127.0.0.1:6000

Table 123. causal_clustering.unknown_address_logging_throttle

Description	Throttle limit for logging unknown cluster member address.
Valid values	causal_clustering.unknown_address_logging_throttle is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	10s

Table 124. causal_clustering.upstream_selection_strategy

Description	An ordered list in descending preference of the strategy which read replicas use to choose the upstream server from which to pull transactional updates.
Valid values	causal_clustering.upstream_selection_strategy is a list separated by "," where items are a string
Default value	[default]

Table 125. causal_clustering.user_defined_upstream_strategy

Description	Configuration of a user-defined upstream selection strategy. The user-defined strategy is used if the list of strategies (causal_clustering.upstream_selection_strategy) includes the value user_defined .
Valid values	causal_clustering.user_defined_upstream_strategy is a string
Default value	

Table 126. `cypher.default_language_version`

Description	Set this to specify the default parser (language version).
Valid values	<code>cypher.default_language_version</code> is one of <code>2.3</code> , <code>3.1</code> , <code>3.4</code> , <code>3.5</code> , <code>default</code>
Default value	<code>default</code>

Table 127. `cypher.forbid_exhaustive_shortestpath`

Description	This setting is associated with performance optimization. Set this to <code>true</code> in situations where it is preferable to have any queries using the 'shortestPath' function terminate as soon as possible with no answer, rather than potentially running for a long time attempting to find an answer (even if there is no path to be found). For most queries, the 'shortestPath' algorithm will return the correct answer very quickly. However there are some cases where it is possible that the fast bidirectional breadth-first search algorithm will find no results even if they exist. This can happen when the predicates in the <code>WHERE</code> clause applied to 'shortestPath' cannot be applied to each step of the traversal, and can only be applied to the entire path. When the query planner detects these special cases, it will plan to perform an exhaustive depth-first search if the fast algorithm finds no paths. However, the exhaustive search may be orders of magnitude slower than the fast algorithm. If it is critical that queries terminate as soon as possible, it is recommended that this option be set to <code>true</code> , which means that Neo4j will never consider using the exhaustive search for shortestPath queries. However, please note that if no paths are found, an error will be thrown at run time, which will need to be handled by the application.
Valid values	<code>cypher.forbid_exhaustive_shortestpath</code> is a boolean
Default value	<code>false</code>

Table 128. `cypher.forbid_shortestpath_common_nodes`

Description	This setting is associated with performance optimization. The shortest path algorithm does not work when the start and end nodes are the same. With this setting set to <code>false</code> no path will be returned when that happens. The default value of <code>true</code> will instead throw an exception. This can happen if you perform a shortestPath search after a cartesian product that might have the same start and end nodes for some of the rows passed to shortestPath. If it is preferable to not experience this exception, and acceptable for results to be missing for those rows, then set this to <code>false</code> . If you cannot accept missing results, and really want the shortestPath between two common nodes, then re-write the query using a standard Cypher variable length pattern expression followed by ordering by path length and limiting to one result.
Valid values	<code>cypher.forbid_shortestpath_common_nodes</code> is a boolean
Default value	<code>true</code>

Table 129. `cypher.hints_error`

Description	Set this to specify the behavior when Cypher planner or runtime hints cannot be fulfilled. If true, then non-conformance will result in an error, otherwise only a warning is generated.
Valid values	<code>cypher.hints_error</code> is a boolean
Default value	<code>false</code>

Table 130. `cypher.lenient_create_relationship`

Description	Set this to change the behavior for Cypher create relationship when the start or end node is missing. By default this fails the query and stops execution, but by setting this flag the create operation is simply not performed and execution continues.
Valid values	<code>cypher.lenient_create_relationship</code> is a boolean
Default value	<code>false</code>

Table 131. `cypher.min_replan_interval`

Description	The minimum time between possible cypher query replanning events. After this time, the graph statistics will be evaluated, and if they have changed by more than the value set by <code>cypher.statistics_divergence_threshold</code> , the query will be replanned. If the statistics have not changed sufficiently, the same interval will need to pass before the statistics will be evaluated again. Each time they are evaluated, the divergence threshold will be reduced slightly until it reaches 10% after 7h, so that even moderately changing databases will see query replanning after a sufficiently long time interval.
-------------	---

Valid values	cypher.min_replan_interval is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	10s

Table 132. `cypher.planner`

Description	Set this to specify the default planner for the default language version.
Valid values	cypher.planner is one of COST, RULE, default
Default value	default

Table 133. `cypher.statistics_divergence_threshold`

Description	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 $\text{abs}(a-b)/\max(a,b)$. This means that a value of 0.75 requires the database to approximately quadruple in size. A value of 0 means replan as soon as possible, with the soonest being defined by the <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.
Valid values	cypher.statistics_divergence_threshold is a double which is in the range 0.0 to 1.0
Default value	0.75

Table 134. `db.temporal.timezone`

Description	Database timezone for temporal functions. All Time and DateTime values that are created without an explicit timezone will use this configured default timezone.
Valid values	db.temporal.timezone is a string describing a timezone, either described by offset (e.g. '+02:00') or by name (e.g. 'Europe/Stockholm')
Default value	Z

Table 135. `dbms.active_database`

Description	Name of the database to load.
Valid values	dbms.active_database is a string which org.neo4j.kernel.configuration.Settings\$\$Lambda\$59/319877175@133b3a95
Default value	graph.db

Table 136. `dbms.allow_format_migration`

Description	Whether to allow a store upgrade in case the current version of the database starts against an older store version. Setting this to true does not guarantee successful upgrade, it just allows an upgrade to be performed.
Valid values	dbms.allow_format_migration is a boolean
Default value	false
Deprecated	The <code>dbms.allow_format_migration</code> configuration setting has been deprecated.
Replaced by	<code>dbms.allow_upgrade</code>

Table 137. `dbms.allow_upgrade`

Description	Whether to allow an upgrade in case the current version of the database starts against an older version.
Valid values	dbms.allow_upgrade is a boolean
Default value	false

Table 138. `dbms.backup.address`

Description	Listening server for online backups. The protocol running varies depending on deployment. In a Causal Clustering environment this is the same protocol that runs on causal_clustering.transaction_listen_address . The port range is only respected in a HA or single instance deployment. In Causal Clustering a single port should be used.
Valid values	dbms.backup.address is a hostname and port
Default value	127.0.0.1:6362-6372

Table 139. dbms.backup.enabled

Description	Enable support for running online backups.
Valid values	dbms.backup.enabled is a boolean
Default value	true

Table 140. dbms.backup.ssl_policy

Description	Name of the SSL policy to be used by backup, as defined under the dbms.ssl.policy.* settings. If no policy is configured then the communication will not be secured.
Valid values	dbms.backup.ssl_policy is a string

Table 141. dbms.checkpoint

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

Table 142. dbms.checkpoint.interval.time

Description	Configures the time interval between check-points. The database will not check-point more often than this (unless check pointing is triggered by a different event), but might check-point less often than this interval, if performing a check-point takes longer time than the configured interval. A check-point is a point in the transaction logs, from which recovery would start from. Longer check-point intervals typically means that recovery will take longer to complete in case of a crash. On the other hand, a longer check-point interval can also reduce the I/O load that the database places on the system, as each check-point implies a flushing and forcing of all the store files.
Valid values	dbms.checkpoint.interval.time is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	900s

Table 143. dbms.checkpoint.interval.tx

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

Table 144. dbms.checkpoint.iops.limit

Description	Limit the number of IOs the background checkpoint process will consume per second. This setting is advisory, is ignored in Neo4j Community Edition, and is followed to best effort in Enterprise Edition. An IO is in this case a 8 KiB (mostly sequential) write. Limiting the write IO in this way will leave more bandwidth in the IO subsystem to service random-read IOs, which is important for the response time of queries when the database cannot fit entirely in memory. The only drawback of this setting is that longer checkpoint times may lead to slightly longer recovery times in case of a database or system crash. A lower number means lower IO pressure, and consequently longer checkpoint times. The configuration can also be commented out to remove the limitation entirely, and let the checkpoint flush data as fast as the hardware will go. Set this to -1 to disable the IOPS limit.
Valid values	dbms.checkpoint.iops.limit is an integer
Dynamic	true
Default value	300

Table 145. dbms.config.strict_validation

Description	A strict configuration validation will prevent the database from starting up if unknown configuration options are specified in the neo4j settings namespace (such as dbms., ha., cypher., etc). This is currently false by default but will be true by default in 4.0.
Valid values	dbms.config.strict_validation is a boolean
Default value	false

Table 146. dbms.connector.bolt.enabled

Description	Enable this connector.
Valid values	dbms.connector.bolt.enabled is a boolean
Default value	true

Table 147. dbms.connector.http.enabled

Description	Enable this connector.
Valid values	dbms.connector.http.enabled is a boolean
Default value	true

Table 148. dbms.connector.https.enabled

Description	Enable this connector.
Valid values	dbms.connector.https.enabled is a boolean
Default value	true

Table 149. dbms.connectors.default_advertised_address

Description	Default hostname or IP address the server uses to advertise itself to its connectors. To advertise a specific hostname or IP address for a specific connector, specify the advertised_address property for the specific connector.
Valid values	dbms.connectors.default_advertised_address is a string
Default value	localhost

Table 150. dbms.connectors.default_listen_address

Description	Default network interface to listen for incoming connections. To listen for connections on all interfaces, use "0.0.0.0". To bind specific connectors to a specific network interfaces, specify the listen_address properties for the specific connector.
Valid values	dbms.connectors.default_listen_address is a string
Default value	127.0.0.1

Table 151. dbms.db.timezone

Description	Database timezone. Among other things, this setting influences which timezone the logs and monitoring procedures use.
Valid values	dbms.db.timezone is one of <code>UTC</code> , <code>SYSTEM</code>
Default value	<code>UTC</code>

Table 152. dbms.directories.certificates

Description	Directory for storing certificates to be used by Neo4j for TLS connections.
Valid values	A filesystem path; relative paths are resolved against the root, <code><unsupported.dbms.directories.neo4j_home></code>
Default value	<code>certificates</code>

Table 153. dbms.directories.data

Description	Path of the data directory. You must not configure more than one Neo4j installation to use the same data directory.
Valid values	A filesystem path; relative paths are resolved against the root, <code><unsupported.dbms.directories.neo4j_home></code>
Default value	<code>data</code>

Table 154. dbms.directories.import

Description	Sets the root directory for file URLs used with the Cypher <code>LOAD CSV</code> clause. This must be set to a single directory, restricting access to only those files within that directory and its subdirectories.
Valid values	A filesystem path; relative paths are resolved against the root, <code><unsupported.dbms.directories.neo4j_home></code>

Table 155. dbms.directories.lib

Description	Path of the lib directory.
Valid values	A filesystem path; relative paths are resolved against the root, <code><unsupported.dbms.directories.neo4j_home></code>
Default value	<code>lib</code>

Table 156. dbms.directories.logs

Description	Path of the logs directory.
Valid values	A filesystem path; relative paths are resolved against the root, <code><unsupported.dbms.directories.neo4j_home></code>
Default value	<code>logs</code>

Table 157. dbms.directories.metrics

Description	The target location of the CSV files: a path to a directory wherein a CSV file per reported field will be written.
Valid values	A filesystem path; relative paths are resolved against the root, <code><unsupported.dbms.directories.neo4j_home></code>
Default value	<code>metrics</code>

Table 158. dbms.directories.plugins

Description	Location of the database plugin directory. Compiled Java JAR files that contain database procedures will be loaded if they are placed in this directory.
Valid values	A filesystem path; relative paths are resolved against the root, <code><unsupported.dbms.directories.neo4j_home></code>
Default value	<code>plugins</code>

Table 159. dbms.directories.run

Description	Path of the run directory. This directory holds Neo4j's runtime state, such as a pidfile when it is running in the background. The pidfile is created when starting neo4j and removed when stopping it. It may be placed on an in-memory filesystem such as tmpfs.
Valid values	A filesystem path; relative paths are resolved against the root, < <i>unsupported.dbms.directories.neo4j_home</i> >
Default value	run

Table 160. dbms.directories.tx_log

Description	Location where Neo4j keeps the logical transaction logs.
Valid values	A filesystem path; relative paths are resolved against the root, < <i>unsupported.dbms.directories.database</i> >
Default value	data/databases/graph.db

Table 161. dbms.filewatcher.enabled

Description	Allows the enabling or disabling of the file watcher service. This is an auxiliary service but should be left enabled in almost all cases.
Valid values	dbms.filewatcher.enabled is a boolean
Default value	true

Table 162. dbms.ids.reuse.types.override

Description	Specified names of id types (comma separated) that should be reused. Currently only 'node' and 'relationship' types are supported.
Valid values	dbms.ids.reuse.types.override is a list separated by "," where items are one of NODE, RELATIONSHIP
Default value	[RELATIONSHIP, NODE]

Table 163. dbms.import.csv.buffer_size

Description	The size of the internal buffer in bytes used by LOAD CSV. If the csv file contains huge fields this value may have to be increased.
Valid values	dbms.import.csv.buffer_size is an integer which is minimum 1
Default value	2097152

Table 164. dbms.import.csv.legacy_quoteEscaping

Description	Selects whether to conform to the standard https://tools.ietf.org/html/rfc4180 for interpreting escaped quotation characters in CSV files loaded using LOAD CSV. Setting this to false will use the standard, interpreting repeated quotes "" as a single in-lined quote, while true will use the legacy convention originally supported in Neo4j 3.0 and 3.1, allowing a backslash to include quotes in-lined in fields.
Valid values	dbms.import.csv.legacy_quoteEscaping is a boolean
Default value	true

Table 165. dbms.index.default_schema_provider

Description	Index provider to use for newly created schema indexes. An index provider may store different value types in separate physical indexes. lucene-1.0: Spatial and temporal value types are stored in native indexes, remaining value types in Lucene index. lucene+native-1.0: Spatial, temporal and number value types are stored in native indexes and remaining value types in Lucene index. lucene+native-2.0: Spatial, temporal, number and string value types are stored in native indexes and remaining value types in Lucene index. native-btree-1.0: All value types and arrays of all value types, even composite keys, are stored in one native index. A native index has faster updates, less heap and CPU usage compared to a Lucene index. A native index has these limitations: Index key (be it single or composite) size limit of 4039 bytes - transaction resulting in index key surpassing that will fail. Reduced performance of CONTAINS and ENDS WITH string index queries, compared to a Lucene index.
-------------	---

Valid values	dbms.index.default_schema_provider is a string
Default value	native-btree-1.0

Table 166. dbms.index.fulltext.default_analyzer

Description	The name of the analyzer that the fulltext indexes should use by default.
Valid values	dbms.index.fulltext.default_analyzer is a string
Default value	standard

Table 167. dbms.index.fulltext.eventually_consistent

Description	Whether or not fulltext indexes should be eventually consistent by default or not.
Valid values	dbms.index.fulltext.eventually_consistent is a boolean
Default value	false

Table 168. dbms.index.fulltext.eventually_consistent_index_update_queue_max_length

Description	The eventually_consistent mode of the fulltext indexes works by queueing up index updates to be applied later in a background thread. This setting sets an upper bound on how many index updates are allowed to be in this queue at any one point in time. When it is reached, the commit process will slow down and wait for the index update applier thread to make some more room in the queue.
Valid values	dbms.index.fulltext.eventually_consistent_index_update_queue_max_length is an integer which is minimum 1, and is maximum 50000000
Default value	10000

Table 169. dbms.index_sampling.background_enabled

Description	Enable or disable background index sampling.
Valid values	dbms.index_sampling.background_enabled is a boolean
Default value	true

Table 170. dbms.index_sampling.buffer_size

Description	Size of buffer used by index sampling. This configuration setting is no longer applicable as from Neo4j 3.0.3. Please use dbms.index_sampling.sample_size_limit instead.
Valid values	dbms.index_sampling.buffer_size is a byte size (valid multipliers are k, m, g, K, M, G) which is in the range 1048576 to 2147483647
Default value	67108864
Deprecated	The <code>dbms.index_sampling.buffer_size</code> configuration setting has been deprecated.
Replaced by	dbms.index_sampling.sample_size_limit

Table 171. dbms.index_sampling.sample_size_limit

Description	Index sampling chunk size limit.
Valid values	dbms.index_sampling.sample_size_limit is an integer which is in the range 1048576 to 2147483647
Default value	8388608

Table 172. dbms.index_sampling.update_percentage

Description	Percentage of index updates of total index size required before sampling of a given index is triggered.
Valid values	dbms.index_sampling.update_percentage is an integer which is minimum 0
Default value	5

Table 173. dbms.index_searcher_cache_size

Description	The maximum number of open Lucene index searchers.
Valid values	dbms.index_searcher_cache_size is an integer which is minimum 1
Default value	2147483647

Table 174. dbms.jvm.additional

Description	Additional JVM arguments. Argument order can be significant. To use a Java commercial feature, the argument to unlock commercial features must precede the argument to enable the specific feature in the config value string. For example, to use Flight Recorder, <code>-XX:+UnlockCommercialFeatures</code> must come before <code>-XX:+FlightRecorder</code> .
Valid values	a string

Table 175. dbms.lock.acquisition.timeout

Description	The maximum time interval within which lock should be acquired.
Valid values	dbms.lock.acquisition.timeout is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	0s

Table 176. dbms.logs.debug.level

Description	Debug log level threshold.
Valid values	dbms.logs.debug.level is one of DEBUG, INFO, WARN, ERROR, NONE
Default value	INFO

Table 177. dbms.logs.debug.path

Description	Path to the debug log file.
Valid values	A filesystem path; relative paths are resolved against the root, <unsupported.dbms.directories.neo4j_home>
Default value	logs/debug.log

Table 178. dbms.logs.debug.rotation.delay

Description	Minimum time interval after last rotation of the debug log before it may be rotated again.
Valid values	dbms.logs.debug.rotation.delay is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	300s

Table 179. dbms.logs.debug.rotation.keep_number

Description	Maximum number of history files for the debug log.
Valid values	dbms.logs.debug.rotation.keep_number is an integer which is minimum 1
Default value	7

Table 180. dbms.logs.debug.rotation.size

Description	Threshold for rotation of the debug log.
Valid values	dbms.logs.debug.rotation.size is a byte size (valid multipliers are k, m, g, K, M, G) which is in the range 0 to 9223372036854775807
Default value	20971520

Table 181. dbms.logs.gc.enabled

Description	Enable GC Logging.
-------------	--------------------

Valid values	dbms.logs.gc.enabled is a boolean
Default value	false

Table 182. dbms.logs.gc.options

Description	GC Logging Options.
Valid values	dbms.logs.gc.options is a string
Default value	-XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:+PrintGCAfterGC -XX:+PrintPromotionFailure -XX:+PrintTenuringDistribution

Table 183. dbms.logs.gc.rotation.keep_number

Description	Number of GC logs to keep.
Valid values	dbms.logs.gc.rotation.keep_number is an integer
Default value	5

Table 184. dbms.logs.gc.rotation.size

Description	Size of each GC log that is kept.
Valid values	dbms.logs.gc.rotation.size is a byte size (valid multipliers are k, m, g, K, M, G) which is in the range 0 to 9223372036854775807
Default value	20971520

Table 185. dbms.logs.http.enabled

Description	Enable HTTP request logging.
Valid values	dbms.logs.http.enabled is a boolean
Default value	false

Table 186. dbms.logs.http.path

Description	Path to HTTP request log.
Valid values	A filesystem path; relative paths are resolved against the root, <unsupported.dbms.directories.neo4j_home>
Default value	logs/http.log

Table 187. dbms.logs.http.rotation.keep_number

Description	Number of HTTP logs to keep.
Valid values	dbms.logs.http.rotation.keep_number is an integer
Default value	5

Table 188. dbms.logs.http.rotation.size

Description	Size of each HTTP log that is kept.
Valid values	dbms.logs.http.rotation.size is a byte size (valid multipliers are k, m, g, K, M, G) which is in the range 0 to 9223372036854775807
Default value	20971520

Table 189. dbms.logs.query.allocation_logging_enabled

Description	Log allocated bytes for the executed queries being logged. The logged number is cumulative over the duration of the query, i.e. for memory intense or long-running queries the value may be larger than the current memory allocation. Requires dbms.track_query_allocation=true
Valid values	dbms.logs.query.allocation_logging_enabled is a boolean

Dynamic	true
Default value	false

Table 190. dbms.logs.query.enabled

Description	Log executed queries that take longer than the configured threshold, dbms.logs.query.threshold . Log entries are by default written to the file <code>query.log</code> located in the Logs directory. For location of the Logs directory, see File locations . This feature is available in the Neo4j Enterprise Edition.
Valid values	dbms.logs.query.enabled is a boolean
Dynamic	true
Default value	false

Table 191. dbms.logs.query.page_logging_enabled

Description	Log page hits and page faults for the executed queries being logged.
Valid values	dbms.logs.query.page_logging_enabled is a boolean
Dynamic	true
Default value	false

Table 192. dbms.logs.query.parameter_logging_enabled

Description	Log parameters for the executed queries being logged.
Valid values	dbms.logs.query.parameter_logging_enabled is a boolean
Dynamic	true
Default value	true

Table 193. dbms.logs.query.path

Description	Path to the query log file.
Valid values	A filesystem path; relative paths are resolved against the root, < unsupported.dbms.directories.neo4j_home >
Default value	logs/query.log

Table 194. dbms.logs.query.rotation.keep_number

Description	Maximum number of history files for the query log.
Valid values	dbms.logs.query.rotation.keep_number is an integer which is minimum 1
Dynamic	true
Default value	7

Table 195. dbms.logs.query.rotation.size

Description	The file size in bytes at which the query log will auto-rotate. If set to zero then no rotation will occur. Accepts a binary suffix k, m or g.
Valid values	dbms.logs.query.rotation.size is a byte size (valid multipliers are k, m, g, K, M, G) which is in the range 0 to 9223372036854775807
Dynamic	true
Default value	20971520

Table 196. dbms.logs.query.runtime_logging_enabled

Description	Logs which runtime that was used to run the query.
--------------------	--

Valid values	dbms.logs.query.runtime_logging_enabled is a boolean
Dynamic	true
Default value	false

Table 197. dbms.logs.query.threshold

Description	If the execution of query takes more time than this threshold, the query is logged - provided query logging is enabled. Defaults to 0 seconds, that is all queries are logged.
Valid values	dbms.logs.query.threshold is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Dynamic	true
Default value	0s

Table 198. dbms.logs.query.time_logging_enabled

Description	Log detailed time information for the executed queries being logged. Requires <code>dbms.track_query_cpu_time=true</code>
Valid values	dbms.logs.query.time_logging_enabled is a boolean
Dynamic	true
Default value	false

Table 199. dbms.logs.security.level

Description	Security log level threshold.
Valid values	dbms.logs.security.level is one of DEBUG, INFO, WARN, ERROR, NONE
Default value	INFO

Table 200. dbms.logs.security.path

Description	Path to the security log file.
Valid values	A filesystem path; relative paths are resolved against the root, < <code>unsupported.dbms.directories.neo4j_home</code> >
Default value	logs/security.log

Table 201. dbms.logs.security.rotation.delay

Description	Minimum time interval after last rotation of the security log before it may be rotated again.
Valid values	dbms.logs.security.rotation.delay is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	300s

Table 202. dbms.logs.security.rotation.keep_number

Description	Maximum number of history files for the security log.
Valid values	dbms.logs.security.rotation.keep_number is an integer which is minimum 1
Default value	7

Table 203. dbms.logs.security.rotation.size

Description	Threshold for rotation of the security log.
Valid values	dbms.logs.security.rotation.size is a byte size (valid multipliers are k, m, g, K, M, G) which is in the range 0 to 9223372036854775807
Default value	20971520

Table 204. dbms.logs.timezone

Description	Database logs timezone.
Valid values	dbms.logs.timezone is one of UTC , SYSTEM
Default value	UTC
Deprecated	The <code>dbms.logs.timezone</code> configuration setting has been deprecated.
Replaced by	dbms.db.timezone

Table 205. dbms.logs.user.path

Description	Path to the user log file.
Valid values	A filesystem path; relative paths are resolved against the root, < <i>unsupported.dbms.directories.neo4j_home</i> >
Default value	logs/neo4j.log

Table 206. dbms.logs.user.rotation.delay

Description	Minimum time interval after last rotation of the user log before it may be rotated again.
Valid values	dbms.logs.user.rotation.delay is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	300s

Table 207. dbms.logs.user.rotation.keep_number

Description	Maximum number of history files for the user log.
Valid values	dbms.logs.user.rotation.keep_number is an integer which is minimum 1
Default value	7

Table 208. dbms.logs.user.rotation.size

Description	Threshold for rotation of the user log. If set to 0 log rotation is disabled.
Valid values	dbms.logs.user.rotation.size is a byte size (valid multipliers are k , m , g , K , M , G) which is in the range 0 to 9223372036854775807
Default value	0

Table 209. dbms.logs.user.stdout_enabled

Description	Send user logs to the process stdout. If this is disabled then logs will instead be sent to the file <code>neo4j.log</code> located in the logs directory. For location of the Logs directory, see File locations .
Valid values	dbms.logs.user.stdout_enabled is a boolean
Default value	true

Table 210. dbms.memory.heap.initial_size

Description	Initial heap size. By default it is calculated based on available system resources.
Valid values	a byte size (valid units are k , K , m , M , g , G)

Table 211. dbms.memory.heap.max_size

Description	Maximum heap size. By default it is calculated based on available system resources.
Valid values	a byte size (valid units are k , K , m , M , g , G)

Table 212. dbms.memory.pagecache.size

Description	The amount of memory to use for mapping the store files, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g'). If Neo4j is running on a dedicated server, then it is generally recommended to leave about 2-4 gigabytes for the operating system, give the JVM enough heap to hold all your transaction state and query context, and then leave the rest for the page cache. If no page cache memory is configured, then a heuristic setting is computed based on available system resources.
Valid values	dbms.memory.pagecache.size is a string

Table 213. dbms.memory.pagecache.swapper

Description	Specify which page swapper to use for doing paged IO. This is only used when integrating with proprietary storage technology.
Valid values	dbms.memory.pagecache.swapper is a string

Table 214. dbms.mode

Description	Configure the operating mode of the database — 'SINGLE' for stand-alone operation, 'HA' for operating as a member in an HA cluster, 'ARBITER' for a cluster member with no database in an HA cluster, 'CORE' for operating as a core member of a Causal Cluster, or 'READ_REPLICA' for operating as a read replica member of a Causal Cluster.
Valid values	dbms.mode is one of SINGLE , HA , ARBITER , CORE , READ_REPLICA
Default value	SINGLE

Table 215. dbms.netty.ssl.provider

Description	Netty SSL provider.
Valid values	dbms.netty.ssl.provider is one of JDK , OPENSSL , OPENSSL_REFCNT
Default value	JDK

Table 216. dbms.procedures.kill_query_verbose

Description	Specifies whether or not dbms.killQueries produces a verbose output, with information about which queries were not found.
Valid values	dbms.procedures.kill_query_verbose is a boolean
Default value	true

Table 217. dbms.query_cache_size

Description	The number of Cypher query execution plans that are cached.
Valid values	dbms.query_cache_size is an integer which is minimum 0
Default value	1000

Table 218. dbms.read_only

Description	Only allow read operations from this Neo4j instance. This mode still requires write access to the directory for lock purposes.
Valid values	dbms.read_only is a boolean
Default value	false

Table 219. dbms.record_format

Description	Database record format. Valid values: standard , high_limit . The high_limit format is available for Enterprise Edition only. It is required if you have a graph that is larger than 34 billion nodes, 34 billion relationships, or 68 billion properties. A change of the record format is irreversible. Certain operations may suffer from a performance penalty of up to 10%, which is why this format is not switched on by default.
Valid values	dbms.record_format is a string
Default value	

Table 220. dbms.relationship_grouping_threshold

Description	Relationship count threshold for considering a node to be dense.
Valid values	dbms.relationship_grouping_threshold is an integer which is minimum 1
Default value	50

Table 221. dbms.rest.transaction.idle_timeout

Description	Timeout for idle transactions in the REST endpoint.
Valid values	dbms.rest.transaction.idle_timeout is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	60s

Table 222. dbms.security.allow_csv_import_from_file_urls

Description	Determines if Cypher will allow using file URLs when loading data using <code>LOAD CSV</code> . Setting this value to <code>false</code> will cause Neo4j to fail <code>LOAD CSV</code> clauses that load data from the file system.
Valid values	dbms.security.allow_csv_import_from_file_urls is a boolean
Default value	<code>true</code>

Table 223. dbms.security.auth_cache_max_capacity

Description	The maximum capacity for authentication and authorization caches (respectively).
Valid values	dbms.security.auth_cache_max_capacity is an integer
Default value	10000

Table 224. dbms.security.auth_cache_ttl

Description	The time to live (TTL) for cached authentication and authorization info when using external auth providers (LDAP or plugin). Setting the TTL to 0 will disable auth caching. Disabling caching while using the LDAP auth provider requires the use of an LDAP system account for resolving authorization information.
Valid values	dbms.security.auth_cache_ttl is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	600s

Table 225. dbms.security.auth_cache_use_ttl

Description	Enable time-based eviction of the authentication and authorization info cache for external auth providers (LDAP or plugin). Disabling this setting will make the cache live forever and only be evicted when <code>dbms.security.auth_cache_max_capacity</code> is exceeded.
Valid values	dbms.security.auth_cache_use_ttl is a boolean
Default value	<code>true</code>

Table 226. dbms.security.auth_enabled

Description	Enable auth requirement to access Neo4j.
Valid values	dbms.security.auth_enabled is a boolean
Default value	<code>true</code>

Table 227. dbms.security.auth_lock_time

Description	The amount of time user account should be locked after a configured number of unsuccessful authentication attempts. The locked out user will not be able to log in until the lock period expires, even if correct credentials are provided. Setting this configuration option to a low value is not recommended because it might make it easier for an attacker to brute force the password.
Valid values	dbms.security.auth_lock_time is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's') which is minimum <code>PT0S</code>

Default value	5s
----------------------	----

Table 228. dbms.security.auth_max_failed_attempts

Description	The maximum number of unsuccessful authentication attempts before imposing a user lock for the configured amount of time. The locked out user will not be able to log in until the lock period expires, even if correct credentials are provided. Setting this configuration option to values less than 3 is not recommended because it might make it easier for an attacker to brute force the password.
Valid values	dbms.security.auth_max_failed_attempts is an integer which is minimum 0
Default value	3

Table 229. dbms.security.auth_provider

Description	The authentication and authorization provider that contains both the users and roles. This can be one of the built-in <code>native</code> or <code>ldap</code> providers, or it can be an externally provided plugin, with a custom name prefixed by <code>plugin-</code> , i.e. <code>plugin-<AUTH_PROVIDER_NAME></code> .
Valid values	dbms.security.auth_provider is a string
Default value	<code>native</code>

Table 230. dbms.security.causal_clustering_status_auth_enabled

Description	Require authorization for access to the Causal Clustering status endpoints.
Valid values	dbms.security.causal_clustering_status_auth_enabled is a boolean
Default value	<code>true</code>

Table 231. dbms.security.ha_status_auth_enabled

Description	Require authorization for access to the HA status endpoints.
Valid values	dbms.security.ha_status_auth_enabled is a boolean
Default value	<code>true</code>
Deprecated	The <code>dbms.security.ha_status_auth_enabled</code> configuration setting has been deprecated.

Table 232. dbms.security.http_access_control_allow_origin

Description	Value of the Access-Control-Allow-Origin header sent over any HTTP or HTTPS connector. This defaults to '*', which allows broadest compatibility. Note that any URI provided here limits HTTP/HTTPS access to that URI only.
Valid values	dbms.security.http_access_control_allow_origin is a string
Default value	*

Table 233. dbms.security.http_authorization_classes

Description	Comma-separated list of custom security rules for Neo4j to use.
Valid values	dbms.security.http_authorization_classes is a list separated by "," where items are a string
Default value	[]

Table 234. dbms.security.http_strict_transport_security

Description	Value of the HTTP Strict-Transport-Security (HSTS) response header. This header tells browsers that a webpage should only be accessed using HTTPS instead of HTTP. It is attached to every HTTPS response. Setting is not set by default so 'Strict-Transport-Security' header is not sent. Value is expected to contain directives like 'max-age', 'includeSubDomains' and 'preload'.
Valid values	dbms.security.http_strict_transport_security is a string

Table 235. dbms.security.ldap.authentication.cache_enabled

Description	Determines if the result of authentication via the LDAP server should be cached or not. Caching is used to limit the number of LDAP requests that have to be made over the network for users that have already been authenticated successfully. A user can be authenticated against an existing cache entry (instead of via an LDAP server) as long as it is alive (see <code>dbms.security.auth_cache_ttl</code>). An important consequence of setting this to <code>true</code> is that Neo4j then needs to cache a hashed version of the credentials in order to perform credentials matching. This hashing is done using a cryptographic hash function together with a random salt. Preferably a conscious decision should be made if this method is considered acceptable by the security standards of the organization in which this Neo4j instance is deployed.
Valid values	<code>dbms.security.ldap.authentication.cache_enabled</code> is a boolean
Default value	<code>true</code>

Table 236. `dbms.security.ldap.authentication.mechanism`

Description	LDAP authentication mechanism. This is one of <code>simple</code> or a SASL mechanism supported by JNDI, for example <code>DIGEST-MD5</code> . <code>simple</code> is basic username and password authentication and SASL is used for more advanced mechanisms. See RFC 2251 LDAPv3 documentation for more details.
Valid values	<code>dbms.security.ldap.authentication.mechanism</code> is a string
Default value	<code>simple</code>

Table 237. `dbms.security.ldap.authentication.use_samaccountname`

Description	Perform authentication with sAMAccountName instead of DN. Using this setting requires <code>dbms.security.ldap.authorization.system_username</code> and <code>dbms.security.ldap.authorization.system_password</code> 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.
Valid values	<code>dbms.security.ldap.authentication.use_samaccountname</code> is a boolean
Default value	<code>false</code>

Table 238. `dbms.security.ldap.authentication.user_dn_template`

Description	LDAP user DN template. An LDAP object is referenced by its distinguished name (DN), and a user DN is an LDAP fully-qualified unique user identifier. This setting is used to generate an LDAP DN that conforms with the LDAP directory's schema from the user principal that is submitted with the authentication token when logging in. The special token <code>{0}</code> is a placeholder where the user principal will be substituted into the DN string.
Valid values	<code>dbms.security.ldap.authentication.user_dn_template</code> is a string
Default value	<code>uid={0},ou=users,dc=example,dc=com</code>

Table 239. `dbms.security.ldap.authorization.group_membership_attributes`

Description	A list of attribute names on a user object that contains groups to be used for mapping to roles when LDAP authorization is enabled.
Valid values	<code>dbms.security.ldap.authorization.group_membership_attributes</code> is a list separated by "," where items are a string
Default value	<code>[memberOf]</code>

Table 240. `dbms.security.ldap.authorization.group_to_role_mapping`

Description	An authorization mapping from LDAP group names to Neo4j role names. The map should be formatted as a semicolon separated list of key-value pairs, where the key is the LDAP group name and the value is a comma separated list of corresponding role names. For example: <code>group1=role1;group2=role2;group3=role3,role4,role5</code> You could also use whitespaces and quotes around group names to make this mapping more readable, for example: ---- <code>dbms.security.ldap.authorization.group_to_role_mapping=\ "cn=Neo4j Read Only,cn=users,dc=example,dc=com" = reader; \ "cn=Neo4j Read-Write,cn=users,dc=example,dc=com" = publisher; \ "cn=Neo4j Schema Manager,cn=users,dc=example,dc=com" = architect; \ "cn=Neo4j Administrator,cn=users,dc=example,dc=com" = admin</code> ----- Deprecated: This will be replaced by dynamic configuration in the system graph in 4.0, including a migration step for the existing setting value.
-------------	---

Valid values	dbms.security.ldap.authorization.group_to_role_mapping is a string
Deprecated	The <code>dbms.security.ldap.authorization.group_to_role_mapping</code> configuration setting has been deprecated.

Table 241. `dbms.security.ldap.authorization.system_password`

Description	An LDAP system account password to use for authorization searches when <code>dbms.security.ldap.authorization.use_system_account</code> is <code>true</code> .
Valid values	dbms.security.ldap.authorization.system_password is a string

Table 242. `dbms.security.ldap.authorization.system_username`

Description	An LDAP system account username to use for authorization searches when <code>dbms.security.ldap.authorization.use_system_account</code> is <code>true</code> . Note that the <code>dbms.security.ldap.authentication.user_dn_template</code> will not be applied to this username, so you may have to specify a full DN.
Valid values	dbms.security.ldap.authorization.system_username is a string

Table 243. `dbms.security.ldap.authorization.use_system_account`

Description	Perform LDAP search for authorization info using a system account instead of the user's own account. If this is set to <code>false</code> (default), the search for group membership will be performed directly after authentication using the LDAP context bound with the user's own account. The mapped roles will be cached for the duration of <code>dbms.security.auth_cache_ttl</code> , and then expire, requiring re-authentication. To avoid frequently having to re-authenticate sessions you may want to set a relatively long auth cache expiration time together with this option. NOTE: This option will only work if the users are permitted to search for their own group membership attributes in the directory. If this is set to <code>true</code> , the search will be performed using a special system account user with read access to all the users in the directory. You need to specify the username and password using the settings <code>dbms.security.ldap.authorization.system_username</code> and <code>dbms.security.ldap.authorization.system_password</code> with this option. Note that this account only needs read access to the relevant parts of the LDAP directory and does not need to have access rights to Neo4j, or any other systems.
Valid values	dbms.security.ldap.authorization.use_system_account is a boolean
Default value	<code>false</code>

Table 244. `dbms.security.ldap.authorization.user_search_base`

Description	The name of the base object or named context to search for user objects when LDAP authorization is enabled. A common case is that this matches the last part of <code>dbms.security.ldap.authentication.user_dn_template</code> .
Valid values	dbms.security.ldap.authorization.user_search_base is a string
Default value	<code>ou=users,dc=example,dc=com</code>

Table 245. `dbms.security.ldap.authorization.user_search_filter`

Description	The LDAP search filter to search for a user principal when LDAP authorization is enabled. The filter should contain the placeholder token <code>{0}</code> which will be substituted for the user principal.
Valid values	dbms.security.ldap.authorization.user_search_filter is a string
Default value	<code>(&(objectClass=*)(uid={0}))</code>

Table 246. `dbms.security.ldap.connection_timeout`

Description	The timeout for establishing an LDAP connection. If a connection with the LDAP server cannot be established within the given time the attempt is aborted. A value of 0 means to use the network protocol's (i.e., TCP's) timeout value.
Valid values	dbms.security.ldap.connection_timeout is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	<code>30s</code>

Table 247. dbms.security.ldap.host

Description	URL of LDAP server to use for authentication and authorization. The format of the setting is <protocol>://<hostname>:<port>, where hostname is the only required field. The supported values for protocol are <code>ldap</code> (default) and <code>ldaps</code> . The default port for <code>ldap</code> is 389 and for <code>ldaps</code> 636. For example: <code>ldaps://ldap.example.com:10389</code> . You may want to consider using STARTTLS (<code>dbms.security.ldap.use_starttls</code>) instead of LDAPS for secure connections, in which case the correct protocol is <code>ldap</code> .
Valid values	dbms.security.ldap.host is a string
Default value	<code>localhost</code>

Table 248. dbms.security.ldap.read_timeout

Description	The timeout for an LDAP read request (i.e. search). If the LDAP server does not respond within the given time the request will be aborted. A value of 0 means wait for a response indefinitely.
Valid values	dbms.security.ldap.read_timeout is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	<code>30s</code>

Table 249. dbms.security.ldap.referral

Description	The LDAP referral behavior when creating a connection. This is one of <code>follow</code> , <code>ignore</code> or <code>throw</code> . * <code>follow</code> automatically follows any referrals * <code>ignore</code> ignores any referrals * <code>throw</code> throws an exception, which will lead to authentication failure.
Valid values	dbms.security.ldap.referral is a string
Default value	<code>follow</code>

Table 250. dbms.security.ldap.use_starttls

Description	Use secure communication with the LDAP server using opportunistic TLS. First an initial insecure connection will be made with the LDAP server, and a STARTTLS command will be issued to negotiate an upgrade of the connection to TLS before initiating authentication.
Valid values	dbms.security.ldap.use_starttls is a boolean
Default value	<code>false</code>

Table 251. dbms.security.log_successful_authentication

Description	Set to log successful authentication events to the security log. If this is set to <code>false</code> only failed authentication events will be logged, which could be useful if you find that the successful events spam the logs too much, and you do not require full auditing capability.
Valid values	dbms.security.log_successful_authentication is a boolean
Default value	<code>true</code>

Table 252. dbms.security.procedures.default_allowed

Description	The default role that can execute all procedures and user-defined functions that are not covered by the <code>dbms.security.procedures.roles</code> setting. If the <code>dbms.security.procedures.default_allowed</code> setting is the empty string (default), procedures will be executed according to the same security rules as normal Cypher statements. Deprecated: This will be replaced by dynamic configuration in the system graph in 4.0, including a migration step for the existing setting value.
Valid values	dbms.security.procedures.default_allowed is a string
Default value	
Deprecated	The <code>dbms.security.procedures.default_allowed</code> configuration setting has been deprecated.

Table 253. dbms.security.procedures.roles

Description	This provides a finer level of control over which roles can execute procedures than the <code>dbms.security.procedures.default_allowed</code> setting. For example: <code>dbms.security.procedures.roles=apoc.convert.*:reader;apoc.load.json*:writer;apoc.trigger.add:TriggerHappy</code> will allow the role <code>reader</code> to execute all procedures in the <code>apoc.convert</code> namespace, the role <code>writer</code> to execute all procedures in the <code>apoc.load</code> namespace that starts with <code>json</code> and the role <code>TriggerHappy</code> to execute the specific procedure <code>apoc.trigger.add</code> . Procedures not matching any of these patterns will be subject to the <code>dbms.security.procedures.default_allowed</code> setting. Deprecated: This will be replaced by dynamic configuration in the system graph in 4.0, including a migration step for the existing setting value.
Valid values	<code>dbms.security.procedures.roles</code> is a string
Default value	
Deprecated	The <code>dbms.security.procedures.roles</code> configuration setting has been deprecated.

Table 254. `dbms.security.procedures.unrestricted`

Description	A list of procedures and user defined functions (comma separated) that are allowed full access to the database. The list may contain both fully-qualified procedure names, and partial names with the wildcard '*'! Note that this enables these procedures to bypass security. Use with caution.
Valid values	<code>dbms.security.procedures.unrestricted</code> is a string
Default value	

Table 255. `dbms.security.procedures.whitelist`

Description	A list of procedures (comma separated) that are to be loaded. The list may contain both fully-qualified procedure names, and partial names with the wildcard '*'! If this setting is left empty no procedures will be loaded.
Valid values	<code>dbms.security.procedures.whitelist</code> is a string
Default value	*

Table 256. `dbms.security.property_level.blacklist`

Description	An authorization mapping for property level access for roles. The map should be formatted as a semicolon separated list of key-value pairs, where the key is the role name and the value is a comma separated list of blacklisted properties. For example: <code>role1=prop1;role2=prop2;role3=prop3,prop4,prop5</code> You could also use whitespaces and quotes around group names to make this mapping more readable, for example: <code>dbms.security.property_level.blacklist=\n "role1" = ssn; \n "role2" =\n ssn,income;\n</code> Deprecated: This will be replaced by dynamic configuration in the system graph in 4.0, including a migration step for the existing setting value.
Valid values	<code>dbms.security.property_level.blacklist</code> is a string
Deprecated	The <code>dbms.security.property_level.blacklist</code> configuration setting has been deprecated.

Table 257. `dbms.security.property_level.enabled`

Description	Set to true to enable property level security.
Valid values	<code>dbms.security.property_level.enabled</code> is a boolean
Default value	<code>false</code>
Deprecated	The <code>dbms.security.property_level.enabled</code> configuration setting has been deprecated.

Table 258. `dbms.shutdown_transaction_end_timeout`

Description	The maximum amount of time to wait for running transactions to complete before allowing initiated database shutdown to continue.
Valid values	<code>dbms.shutdown_transaction_end_timeout</code> is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	<code>10s</code>

Table 259. dbms.ssl.policy.<policynname>.allow_key_generation

Description	Allows the generation of a private key and associated self-signed certificate. Only performed when both objects cannot be found.
Valid values	dbms.ssl.policy.<policynname>.allow_key_generation is a boolean
Default value	false

Table 260. dbms.ssl.policy.<policynname>.base_directory

Description	The mandatory base directory for cryptographic objects of this policy. It is also possible to override each individual configuration with absolute paths.
Valid values	A filesystem path; relative paths are resolved against the root, <unsupported.dbms.directories.neo4j_home>

Table 261. dbms.ssl.policy.<policynname>.ciphers

Description	Restrict allowed ciphers.
Valid values	dbms.ssl.policy.<policynname>.ciphers is a list separated by "," where items are a string

Table 262. dbms.ssl.policy.<policynname>.client_auth

Description	Client authentication stance.
Valid values	dbms.ssl.policy.<policynname>.client_auth is one of NONE, OPTIONAL, REQUIRE
Default value	REQUIRE

Table 263. dbms.ssl.policy.<policynname>.private_key

Description	Private PKCS#8 key in PEM format.
Valid values	A filesystem path; relative paths are resolved against the root, <unsupported.dbms.directories.neo4j_home>
Default value	private.key

Table 264. dbms.ssl.policy.<policynname>.public_certificate

Description	X.509 certificate (chain) of this server in PEM format.
Valid values	A filesystem path; relative paths are resolved against the root, <unsupported.dbms.directories.neo4j_home>
Default value	public.crt

Table 265. dbms.ssl.policy.<policynname>.revoked_dir

Description	Path to directory of CRLs (Certificate Revocation Lists) in PEM format.
Valid values	A filesystem path; relative paths are resolved against the root, <unsupported.dbms.directories.neo4j_home>
Default value	revoked

Table 266. dbms.ssl.policy.<policynname>.tls_versions

Description	Restrict allowed TLS protocol versions.
Valid values	dbms.ssl.policy.<policynname>.tls_versions is a list separated by "," where items are a string
Default value	[TLSv1.2]

Table 267. dbms.ssl.policy.<policynname>.trust_all

Description	Makes this policy trust all remote parties. Enabling this is not recommended and the trusted directory will be ignored.
Valid values	dbms.ssl.policy.<policynname>.trust_all is a boolean

Default value	false
----------------------	-------

Table 268. dbms.ssl.policy.<policynname>.trusted_dir

Description	Path to directory of X.509 certificates in PEM format for trusted parties.
Valid values	A filesystem path; relative paths are resolved against the root, <unsupported.dbms.directories.neo4j_home>
Default value	trusted

Table 269. dbms.ssl.policy.<policynname>.verify_hostname

Description	When true, this node will verify the hostname of every other instance it connects to by comparing the address it used to connect with it and the patterns described in the remote hosts public certificate Subject Alternative Names.
Valid values	dbms.ssl.policy.<policynname>.verify_hostname is a boolean
Default value	false

Table 270. dbms.threads.worker_count

Description	Number of Neo4j worker threads, your OS might enforce a lower limit than the maximum value specified here.
Valid values	dbms.threads.worker_count is an integer which is in the range 1 to 44738
Default value	Number of available processors (max 500).

Table 271. dbms.track_query_allocation

Description	Enables or disables tracking of how many bytes are allocated by the execution of a query. Calling dbms.listQueries will display the time. This can also be logged in the query log by using log_queries_allocation_logging_enabled.
Valid values	dbms.track_query_allocation is a boolean
Dynamic	true
Default value	false

Table 272. dbms.track_query_cpu_time

Description	Enables or disables tracking of how much time a query spends actively executing on the CPU. Calling dbms.listQueries will display the time. This can also be logged in the query log by using log_queries_detailed_time_logging_enabled.
Valid values	dbms.track_query_cpu_time is a boolean
Dynamic	true
Default value	false

Table 273. dbms.transaction.bookmark_ready_timeout

Description	The maximum amount of time to wait for the database state represented by the bookmark.
Valid values	dbms.transaction.bookmark_ready_timeout is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's') which is minimum PT1S
Default value	30s

Table 274. dbms.transaction.monitor.check.interval

Description	Configures the time interval between transaction monitor checks. Determines how often monitor thread will check transaction for timeout.
Valid values	dbms.transaction.monitor.check.interval is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	2s

Table 275. dbms.transaction.timeout

Description	The maximum time interval of a transaction within which it should be completed.
Valid values	dbms.transaction.timeout is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Dynamic	true
Default value	0s

Table 276. dbms.tx_log.rotation.retention_policy

Description	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. For example "10 days" will prune logical logs that only contains transactions older than 10 days from the current time, or "100k txs" will keep the 100k latest transactions and prune any older transactions.
Valid values	dbms.tx_log.rotation.retention_policy is a string which must be <code>true</code> , <code>false</code> or of format <code><number><optional_unit> <type></code> . Valid units are <code>k</code> , <code>M</code> and <code>G</code> . Valid types are <code>files</code> , <code>size</code> , <code>txs</code> , <code>entries</code> , <code>hours</code> and <code>days</code> . For example, <code>100M size</code> will limit logical log space on disk to 100Mb, or <code>200k txs</code> will limiting the number of transactions to keep to 200 000 (matches the pattern <code>^(true keep_all false keep_none (\d+[KkMmGg]?)?((files size txs entries hours days)))\$</code>)
Dynamic	true
Default value	7 days

Table 277. dbms.tx_log.rotation.size

Description	Specifies at which file size the logical log will auto-rotate. Minimum accepted value is 1M.
Valid values	dbms.tx_log.rotation.size is a byte size (valid multipliers are <code>k</code> , <code>m</code> , <code>g</code> , <code>K</code> , <code>M</code> , <code>G</code>) which is minimum <code>1048576</code>
Dynamic	true
Default value	262144000

Table 278. dbms.tx_state.max_off_heap_memory

Description	The maximum amount of off-heap memory that can be used to store transaction state data; it's a total amount of memory shared across all active transactions. Zero means 'unlimited'. Used when <code>dbms.tx_state.memory_allocation</code> is set to 'OFF_HEAP'.
Valid values	dbms.tx_state.max_off_heap_memory is a byte size (valid multipliers are <code>k</code> , <code>m</code> , <code>g</code> , <code>K</code> , <code>M</code> , <code>G</code>) which is minimum <code>0</code>
Default value	2147483648

Table 279. dbms.tx_state.memory_allocation

Description	Defines whether memory for transaction state should be allocated on- or off-heap.
Valid values	dbms.tx_state.memory_allocation is one of <code>ON_HEAP</code> , <code>OFF_HEAP</code>
Default value	<code>ON_HEAP</code>

Table 280. dbms.tx_state.off_heap.block_cache_size

Description	Defines the size of the off-heap memory blocks cache. The cache will contain this number of blocks for each block size that is power of two. Thus, maximum amount of memory used by blocks cache can be calculated as <code>2 * dbms.tx_state.off_heap.max_cacheable_block_size * dbms.tx_state.off_heap.block_cache_size</code>
Valid values	dbms.tx_state.off_heap.block_cache_size is an integer which is minimum <code>16</code>
Default value	128

Table 281. dbms.tx_state.off_heap.max_cacheable_block_size

Description	Defines the maximum size of an off-heap memory block that can be cached to speed up allocations for transaction state data. The value must be a power of 2.
-------------	---

Valid values	dbms.tx_state.off_heap.max_cacheable_block_size is a byte size (valid multipliers are <code>k</code> , <code>m</code> , <code>g</code> , <code>K</code> , <code>M</code> , <code>G</code>) which is minimum <code>4096</code> , and org.neo4j.graphdb.factory.GraphDatabaseSettings\$\$Lambda\$69/886210066@5ab315ed
Default value	<code>524288</code>

Table 282. `dbms.udc.enabled`

Description	Enable the UDC extension.
Valid values	<code>dbms.udc.enabled</code> is a boolean
Default value	<code>true</code>

Table 283. `dbms.unmanaged_extension_classes`

Description	Comma-separated list of <classname>=<mount point> for unmanaged extensions.
Valid values	<code>dbms.unmanaged_extension_classes</code> is a comma-separated list of <classname>=<mount point> strings
Default value	<code>[]</code>

Table 284. `dbms.windows_service_name`

Description	Name of the Windows Service.
Valid values	a string

Table 285. `ha.allow_init_cluster`

Description	Whether to allow this instance to create a cluster if unable to join.
Valid values	<code>ha.allow_init_cluster</code> is a boolean
Default value	<code>true</code>

Table 286. `ha.branded_data_copying_strategy`

Description	Strategy for how to order handling of branched data on slaves and copying of the store from the master. The default is <code>copy_then_branch</code> , which, when combined with the <code>keep_last</code> or <code>keep_none</code> branch handling strategies results in a safer branching strategy, as there is always a store present so store failure to copy a store (for example, because of network failure) does not leave the instance without a store.
Valid values	<code>ha.branded_data_copying_strategy</code> is one of <code>branch_then_copy</code> , <code>copy_then_branch</code>
Default value	<code>branch_then_copy</code>
Deprecated	The <code>ha.branded_data_copying_strategy</code> configuration setting has been deprecated.

Table 287. `ha.branded_data_policy`

Description	Policy for how to handle branched data.
Valid values	<code>ha.branded_data_policy</code> is one of <code>keep_all</code> , <code>keep_last</code> , <code>keep_none</code>
Default value	<code>keep_all</code>
Deprecated	The <code>ha.branded_data_policy</code> configuration setting has been deprecated.

Table 288. `ha.broadcast_timeout`

Description	Timeout for broadcasting values in cluster. Must consider end-to-end duration of Paxos algorithm. This value is the default value for the <code>ha.join_timeout</code> and <code>ha.leave_timeout</code> settings.
Valid values	<code>ha.broadcast_timeout</code> is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	<code>30s</code>

Table 289. `ha.configuration_timeout`

Description	Timeout for waiting for configuration from an existing cluster member during cluster join.
Valid values	ha.configuration_timeout is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	1s

Table 290. ha.data_chunk_size

Description	Max size of the data chunks that flows between master and slaves in HA. Bigger size may increase throughput, but may also be more sensitive to variations in bandwidth, whereas lower size increases tolerance for bandwidth variations.
Valid values	ha.data_chunk_size is a byte size (valid multipliers are K , m , g , K , M , G) which is minimum 1024
Default value	2097152
Deprecated	The <code>ha.data_chunk_size</code> configuration setting has been deprecated.

Table 291. ha.default_timeout

Description	Default timeout used for clustering timeouts. Override specific timeout settings with proper values if necessary. This value is the default value for the <code>ha.heartbeat_interval</code> , <code>ha.paxos_timeout</code> and <code>ha.learn_timeout</code> settings.
Valid values	ha.default_timeout is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	5s

Table 292. ha.election_timeout

Description	Timeout for waiting for other members to finish a role election. Defaults to <code>ha.paxos_timeout</code> .
Valid values	ha.election_timeout is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	5s

Table 293. ha.heartbeat_interval

Description	How often heartbeat messages should be sent. Defaults to <code>ha.default_timeout</code> .
Valid values	ha.heartbeat_interval is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	5s

Table 294. ha.heartbeat_timeout

Description	How long to wait for heartbeats from other instances before marking them as suspects for failure. This value reflects considerations of network latency, expected duration of garbage collection pauses and other factors that can delay message sending and processing. Larger values will result in more stable masters but also will result in longer waits before a failover in case of master failure. This value should not be set to less than twice the <code>ha.heartbeat_interval</code> value otherwise there is a high risk of frequent master switches and possibly branched data occurrence.
Valid values	ha.heartbeat_timeout is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	40s

Table 295. ha.host.coordination

Description	Host and port to bind the cluster management communication.
Valid values	ha.host.coordination is a hostname and port
Default value	0.0.0.0:5001-5099

Table 296. ha.host.data

Description	Hostname and port to bind the HA server.
Valid values	ha.host.data is a hostname and port

Default value	0.0.0.0:6001-6011
Deprecated	The <code>ha.host.data</code> configuration setting has been deprecated.

Table 297. `ha.initial_hosts`

Description	A comma-separated list of other members of the cluster to join.
Valid values	<code>ha.initial_hosts</code> is a list separated by "," where items are a hostname and port

Table 298. `ha.internal_role_switch_timeout`

Description	Timeout for waiting for internal conditions during state switch, like for transactions to complete, before switching to master or slave.
Valid values	<code>ha.internal_role_switch_timeout</code> is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	10s
Deprecated	The <code>ha.internal_role_switch_timeout</code> configuration setting has been deprecated.

Table 299. `ha.join_timeout`

Description	Timeout for joining a cluster. Defaults to <code>ha.broadcast_timeout</code> . Note that if the timeout expires during cluster formation, the operator may have to restart the instance or instances.
Valid values	<code>ha.join_timeout</code> is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	30s

Table 300. `ha.learn_timeout`

Description	Timeout for learning values. Defaults to <code>ha.default_timeout</code> .
Valid values	<code>ha.learn_timeout</code> is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	5s

Table 301. `ha.leave_timeout`

Description	Timeout for waiting for cluster leave to finish. Defaults to <code>ha.broadcast_timeout</code> .
Valid values	<code>ha.leave_timeout</code> is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	30s

Table 302. `ha.max_acceptors`

Description	Maximum number of servers to involve when agreeing to membership changes. In very large clusters, the probability of half the cluster failing is low, but protecting against any arbitrary half failing is expensive. Therefore you may wish to set this parameter to a value less than the cluster size.
Valid values	<code>ha.max_acceptors</code> is an integer which is minimum 1
Default value	21

Table 303. `ha.max_channels_per_slave`

Description	Maximum number of connections a slave can have to the master.
Valid values	<code>ha.max_channels_per_slave</code> is an integer which is minimum 1
Default value	20
Deprecated	The <code>ha.max_channels_per_slave</code> configuration setting has been deprecated.

Table 304. `ha.paxos_timeout`

Description	Default value for all Paxos timeouts. This setting controls the default value for the ha.phase1_timeout , ha.phase2_timeout and ha.election_timeout settings. If it is not given a value it defaults to ha.default_timeout and will implicitly change if ha.default_timeout changes. This is an advanced parameter which should only be changed if specifically advised by Neo4j Professional Services.
Valid values	ha.paxos_timeout is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	5s

Table 305. *ha.phase1_timeout*

Description	Timeout for Paxos phase 1. If it is not given a value it defaults to ha.paxos_timeout and will implicitly change if ha.paxos_timeout changes. This is an advanced parameter which should only be changed if specifically advised by Neo4j Professional Services.
Valid values	ha.phase1_timeout is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	5s

Table 306. *ha.phase2_timeout*

Description	Timeout for Paxos phase 2. If it is not given a value it defaults to ha.paxos_timeout and will implicitly change if ha.paxos_timeout changes. This is an advanced parameter which should only be changed if specifically advised by Neo4j Professional Services.
Valid values	ha.phase2_timeout is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	5s

Table 307. *ha.pull_batch_size*

Description	Size of batches of transactions applied on slaves when pulling from master.
Valid values	ha.pull_batch_size is an integer
Default value	100
Deprecated	The ha.pull_batch_size configuration setting has been deprecated.

Table 308. *ha.pull_interval*

Description	Interval of pulling updates from master.
Valid values	ha.pull_interval is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	0s
Deprecated	The ha.pull_interval configuration setting has been deprecated.

Table 309. *ha.role_switch_timeout*

Description	Timeout for request threads waiting for instance to become master or slave.
Valid values	ha.role_switch_timeout is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	120s
Deprecated	The ha.role_switch_timeout configuration setting has been deprecated.

Table 310. *ha.server_id*

Description	Id for a cluster instance. Must be unique within the cluster.
Valid values	ha.server_id is an instance id, which has to be a valid integer

Table 311. *ha.slave_lock_timeout*

Description	Timeout for taking remote (write) locks on slaves. Defaults to ha.slave_read_timeout .
Valid values	ha.slave_lock_timeout is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')

Default value	20s
Deprecated	The <code>ha.slave_lock_timeout</code> configuration setting has been deprecated.

Table 312. `ha.slave_only`

Description	Whether this instance should only participate as slave in cluster. If set to <code>true</code> , it will never be elected as master.
Valid values	<code>ha.slave_only</code> is a boolean
Default value	<code>false</code>
Deprecated	The <code>ha.slave_only</code> configuration setting has been deprecated.

Table 313. `ha.slave_read_timeout`

Description	How long a slave will wait for response from master before giving up.
Valid values	<code>ha.slave_read_timeout</code> is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	20s
Deprecated	The <code>ha.slave_read_timeout</code> configuration setting has been deprecated.

Table 314. `ha.tx_push_factor`

Description	The amount of slaves the master will ask to replicate a committed transaction.
Valid values	<code>ha.tx_push_factor</code> is an integer which is minimum 0
Default value	1
Deprecated	The <code>ha.tx_push_factor</code> configuration setting has been deprecated.

Table 315. `ha.tx_push_strategy`

Description	Push strategy of a transaction to a slave during commit.
Valid values	<code>ha.tx_push_strategy</code> is one of <code>round_robin</code> , <code>fixed_descending</code> , <code>fixedAscending</code>
Default value	<code>fixed_descending</code>
Deprecated	The <code>ha.tx_push_strategy</code> configuration setting has been deprecated.

Table 316. `https.ssl_policy`

Description	SSL policy name.
Valid values	<code>https.ssl_policy</code> is a string
Default value	<code>legacy</code>

Table 317. `metrics.bolt.messages.enabled`

Description	Enable reporting metrics about Bolt Protocol message processing.
Valid values	<code>metrics.bolt.messages.enabled</code> is a boolean
Default value	<code>true</code>

Table 318. `metrics.csv.enabled`

Description	Set to <code>true</code> to enable exporting metrics to CSV files.
Valid values	<code>metrics.csv.enabled</code> is a boolean
Default value	<code>true</code>

Table 319. `metrics.csv.interval`

Description	The reporting interval for the CSV files. That is, how often new rows with numbers are appended to the CSV files.
Valid values	metrics.csv.interval is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	3s

Table 320. metrics.csv.rotation.keep_number

Description	Maximum number of history files for the csv files.
Valid values	metrics.csv.rotation.keep_number is an integer which is minimum 1
Default value	7

Table 321. metrics.csv.rotation.size

Description	The file size in bytes at which the csv files will auto-rotate. If set to zero then no rotation will occur. Accepts a binary suffix k, m or g.
Valid values	metrics.csv.rotation.size is a byte size (valid multipliers are k, m, g, K, M, G) which is in the range 0 to 9223372036854775807
Default value	10485760

Table 322. metrics.cypher.replanning.enabled

Description	Enable reporting metrics about number of occurred replanning events.
Valid values	metrics.cypher.replanning.enabled is a boolean
Default value	true

Table 323. metrics.enabled

Description	The default enablement value for all the supported metrics. Set this to false to turn off all metrics by default. The individual settings can then be used to selectively re-enable specific metrics.
Valid values	metrics.enabled is a boolean
Default value	true

Table 324. metrics.graphite.enabled

Description	Set to true to enable exporting metrics to Graphite.
Valid values	metrics.graphite.enabled is a boolean
Default value	false

Table 325. metrics.graphite.interval

Description	The reporting interval for Graphite. That is, how often to send updated metrics to Graphite.
Valid values	metrics.graphite.interval is a duration (Valid units are: 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	3s

Table 326. metrics.graphite.server

Description	The hostname or IP address of the Graphite server.
Valid values	metrics.graphite.server is a hostname and port
Default value	:2003

Table 327. metrics.jvm.buffers.enabled

Description	Enable reporting metrics about the buffer pools.
-------------	--

Valid values	metrics.jvm.buffers.enabled is a boolean
Default value	<code>true</code>

Table 328. metrics.jvm.gc.enabled

Description	Enable reporting metrics about the duration of garbage collections.
Valid values	metrics.jvm.gc.enabled is a boolean
Default value	<code>true</code>

Table 329. metrics.jvm.memory.enabled

Description	Enable reporting metrics about the memory usage.
Valid values	metrics.jvm.memory.enabled is a boolean
Default value	<code>true</code>

Table 330. metrics.jvm.threads.enabled

Description	Enable reporting metrics about the current number of threads running.
Valid values	metrics.jvm.threads.enabled is a boolean
Default value	<code>true</code>

Table 331. metrics.neo4j.causal_clustering.enabled

Description	Enable reporting metrics about Causal Clustering mode.
Valid values	metrics.neo4j.causal_clustering.enabled is a boolean
Default value	<code>true</code>

Table 332. metrics.neo4j.checkpointing.enabled

Description	Enable reporting metrics about Neo4j check pointing; when it occurs and how much time it takes to complete.
Valid values	metrics.neo4j.checkpointing.enabled is a boolean
Default value	<code>true</code>

Table 333. metrics.neo4j.cluster.enabled

Description	Enable reporting metrics about HA cluster info.
Valid values	metrics.neo4j.cluster.enabled is a boolean
Default value	<code>true</code>
Deprecated	The <code>metrics.neo4j.cluster.enabled</code> configuration setting has been deprecated.

Table 334. metrics.neo4j.counts.enabled

Description	Enable reporting metrics about approximately how many entities are in the database; nodes, relationships, properties, etc.
Valid values	metrics.neo4j.counts.enabled is a boolean
Default value	<code>true</code>

Table 335. metrics.neo4j.enabled

Description	The default enablement value for all Neo4j specific support metrics. Set this to <code>false</code> to turn off all Neo4j specific metrics by default. The individual <code>metrics.neo4j.*</code> metrics can then be turned on selectively.
Valid values	metrics.neo4j.enabled is a boolean

Default value	<code>true</code>
----------------------	-------------------

Table 336. metrics.neo4j.logrotation.enabled

Description	Enable reporting metrics about the Neo4j log rotation; when it occurs and how much time it takes to complete.
Valid values	metrics.neo4j.logrotation.enabled is a boolean
Default value	<code>true</code>

Table 337. metrics.neo4j.network.enabled

Description	Enable reporting metrics about the network usage.
Valid values	metrics.neo4j.network.enabled is a boolean
Default value	<code>true</code>

Table 338. metrics.neo4j.pagecache.enabled

Description	Enable reporting metrics about the Neo4j page cache; page faults, evictions, flushes, exceptions, etc.
Valid values	metrics.neo4j.pagecache.enabled is a boolean
Default value	<code>true</code>

Table 339. metrics.neo4j.server.enabled

Description	Enable reporting metrics about Server threading info.
Valid values	metrics.neo4j.server.enabled is a boolean
Default value	<code>true</code>

Table 340. metrics.neo4j.tx.enabled

Description	Enable reporting metrics about transactions; number of transactions started, committed, etc.
Valid values	metrics.neo4j.tx.enabled is a boolean
Default value	<code>true</code>

Table 341. metrics.prefix

Description	A common prefix for the reported metrics field names. By default, this is either be 'neo4j', or a computed value based on the cluster and instance names, when running in an HA configuration.
Valid values	metrics.prefix is a string
Default value	<code>neo4j</code>

Table 342. metrics.prometheus.enabled

Description	Set to <code>true</code> to enable the Prometheus endpoint.
Valid values	metrics.prometheus.enabled is a boolean
Default value	<code>false</code>

Table 343. metrics.prometheus.endpoint

Description	The hostname and port to use as Prometheus endpoint.
Valid values	metrics.prometheus.endpoint is a hostname and port
Default value	<code>localhost:2004</code>

Table 344. tools.consistency_checker.check_graph

Description	This setting is deprecated. See commandline arguments for neo4-admin check-consistency instead. Perform checks between nodes, relationships, properties, types and tokens.
Valid values	tools.consistency_checker.check_graph is a boolean
Default value	<code>true</code>
Deprecated	The <code>tools.consistency_checker.check_graph</code> configuration setting has been deprecated.

Table 345. `tools.consistency_checker.check_indexes`

Description	This setting is deprecated. See commandline arguments for neo4-admin check-consistency instead. Perform checks on indexes. Checking indexes is more expensive than checking the native stores, so it may be useful to turn off this check for very large databases.
Valid values	tools.consistency_checker.check_indexes is a boolean
Default value	<code>true</code>
Deprecated	The <code>tools.consistency_checker.check_indexes</code> configuration setting has been deprecated.

Table 346. `tools.consistency_checker.check_label_scan_store`

Description	This setting is deprecated. See commandline arguments for neo4-admin check-consistency instead. Perform checks on the label scan store. Checking this store is more expensive than checking the native stores, so it may be useful to turn off this check for very large databases.
Valid values	tools.consistency_checker.check_label_scan_store is a boolean
Default value	<code>true</code>
Deprecated	The <code>tools.consistency_checker.check_label_scan_store</code> configuration setting has been deprecated.

Table 347. `tools.consistency_checker.check_property_owners`

Description	This setting is deprecated. See commandline arguments for neo4-admin check-consistency instead. Perform optional additional checking on property ownership. This can detect a theoretical inconsistency where a property could be owned by multiple entities. However, the check is very expensive in time and memory, so it is skipped by default.
Valid values	tools.consistency_checker.check_property_owners is a boolean
Default value	<code>false</code>
Deprecated	The <code>tools.consistency_checker.check_property_owners</code> configuration setting has been deprecated.

A.2. Built-in procedures

This section contains a reference of Neo4j built-in procedures.

A.2.1. Procedures, editions and modes

The procedures available depends on the type of installation. Enterprise Edition provides a fuller set of procedures than Community Edition. Cluster members have procedures that are not available in standalone mode.

The cluster-specific procedures are not included in this reference, instead see [Procedures for monitoring a Causal Cluster](#). To check which procedures are available in your Neo4j instance, use the `dbms.procedures()` procedure.

Example 113. List available procedures

To list the procedures available on your particular installation, run the following command in Neo4j Browser or in Cypher Shell:

```
CALL dbms.procedures()
```

A.2.2. Procedure reference

The procedure reference section contains the following:

- [Procedures available in standalone Neo4j Enterprise Edition](#)
- [Procedures available in Neo4j Community Edition](#)

A.2.3. Enterprise Edition procedures

Table 348. Enterprise Edition procedures

Name	Description	Signature	Mode	Roles
db.awaitIndex()	Wait for an index to come online (for example: CALL db.awaitIndex(":Person(name)").)	db.awaitIndex(index :: STRING?, timeOutSeconds = 300 :: INTEGER?) :: VOID	READ	reader, editor, publisher, architect, admin
db.awaitIndexes()	Wait for all indexes to come online (for example: CALL db.awaitIndexes("500")).	db.awaitIndexes(timeOutSeconds = 300 :: INTEGER?) :: VOID	READ	reader, editor, publisher, architect, admin
db.constraints()	List all constraints in the database.	db.constraints() :: (description :: STRING?)	READ	reader, editor, publisher, architect, admin
db.createIndex()	Create a schema index with specified index provider (for example: CALL db.createIndex(":Person(name)", "lucene+native-2.0") - YIELD index, providerName, status	db.createIndex(index :: STRING?, providerName :: STRING?) :: (index :: STRING?, providerName :: STRING?, status :: STRING?)	SCHEMA	architect, admin
db.createLabel()	Create a label	db.createLabel(newLabel :: STRING?) :: VOID	WRITE	editor, publisher, architect, admin
db.createNodeKey()	Create a node key constraint with index backed by specified index provider (for example: CALL db.createNodeKey(":Person(name)", "lucene+native-2.0") - YIELD index, providerName, status	db.createNodeKey(index :: STRING?, providerName :: STRING?) :: (index :: STRING?, providerName :: STRING?, status :: STRING?)	SCHEMA	architect, admin
db.createProperty()	Create a Property	db.createProperty(propertyName :: STRING?) :: VOID	WRITE	editor, publisher, architect, admin
db.createRelationshipType()	Create a RelationshipType	db.createRelationshipType(relationshipType :: STRING?) :: VOID	WRITE	editor, publisher, architect, admin

Name	Description	Signature	Mode	Roles
db.createUniquePropertyConstraint()	Create a unique property constraint with index backed by specified index provider (for example: CALL db.createUniquePropertyConstraint(":Person{name}", "lucene+native-2.0") - YIELD index, providerName, status	db.createUniquePropertyConstraint(index :: STRING?, providerName :: STRING?) :: (index :: STRING?, providerName :: STRING?, status :: STRING?)	SCHEMA	architect, admin
db.index.explicit.addNode()	Add a node to an explicit index based on a specified key and value	db.index.explicit.addNode(indexName :: STRING?, node :: NODE?, key :: STRING?, value :: ANY?) :: (success :: BOOLEAN?)	WRITE	editor, publisher, architect, admin
db.index.explicit.addRelationship()	Add a relationship to an explicit index based on a specified key and value	db.index.explicit.addRelationship(indexName :: STRING?, relationship :: RELATIONSHIP?, key :: STRING?, value :: ANY?) :: (success :: BOOLEAN?)	WRITE	editor, publisher, architect, admin
db.index.explicit.auto.searchNodes()	Search nodes in explicit automatic index. Replaces START n=node:node_auto_index('key:foo*')	db.index.explicit.auto.searchNodes(query :: ANY?) :: (node :: NODE?, weight :: FLOAT?)	READ	reader, editor, publisher, architect, admin
db.index.explicit.auto.searchRelationships()	Search relationship in explicit automatic index. Replaces START r=relationship:relationship_auto_index('key:foo*')	db.index.explicit.auto.searchRelationships(query :: ANY?) :: (relationship :: RELATIONSHIP?, weight :: FLOAT?)	READ	reader, editor, publisher, architect, admin
db.index.explicit.auto.seekNodes()	Get node from explicit automatic index. Replaces START n=node:node_auto_index(key = 'A')	db.index.explicit.auto.seekNodes(key :: STRING?, value :: ANY?) :: (node :: NODE?)	READ	reader, editor, publisher, architect, admin
db.index.explicit.auto.seekRelationships()	Get relationship from explicit automatic index. Replaces START r=relationship:relationship_auto_index(key = 'A')	db.index.explicit.auto.seekRelationships(key :: STRING?, value :: ANY?) :: (relationship :: RELATIONSHIP?)	READ	reader, editor, publisher, architect, admin
db.index.explicit.drop()	Remove an explicit index - YIELD type,name,config	db.index.explicit.drop(indexName :: STRING?) :: (type :: STRING?, name :: STRING?, config :: MAP?)	WRITE	editor, publisher, architect, admin
db.index.explicit.existsForNodes()	Check if a node explicit index exists	db.index.explicit.existsForNodes(indexName :: STRING?) :: (success :: BOOLEAN?)	READ	reader, editor, publisher, architect, admin
db.index.explicit.existsForRelationships()	Check if a relationship explicit index exists	db.index.explicit.existsForRelationships(indexName :: STRING?) :: (success :: BOOLEAN?)	READ	reader, editor, publisher, architect, admin

Name	Description	Signature	Mode	Roles
db.index.explicit.forNodes()	Get or create a node explicit index - YIELD type,name,config	db.index.explicit.forNodes(indexName :: STRING?, config = {} :: MAP?) :: (type :: STRING?, name :: STRING?, config :: MAP?)	WRITE	editor, publisher, architect, admin
db.index.explicit.forRelationships()	Get or create a relationship explicit index - YIELD type,name,config	db.index.explicit.forRelationships(indexName :: STRING?, config = {} :: MAP?) :: (type :: STRING?, name :: STRING?, config :: MAP?)	WRITE	editor, publisher, architect, admin
db.index.explicit.list()	List all explicit indexes - YIELD type,name,config	db.index.explicit.list() :: (type :: STRING?, name :: STRING?, config :: MAP?)	READ	reader, editor, publisher, architect, admin
db.index.explicit.removeNode()	Remove a node from an explicit index with an optional key	db.index.explicit.removeNode(indexName :: STRING?, node :: NODE?, key = <[9895b15e-8693-4a21-a58b-4b7b87e09b8e]> :: STRING?) :: (success :: BOOLEAN?)	WRITE	editor, publisher, architect, admin
db.index.explicit.removeRelationship()	Remove a relationship from an explicit index with an optional key	db.index.explicit.removeRelationship(indexName :: STRING?, relationship :: RELATIONSHIP?, key = <[9895b15e-8693-4a21-a58b-4b7b87e09b8e]> :: STRING?) :: (success :: BOOLEAN?)	WRITE	editor, publisher, architect, admin
db.index.explicit.searchNodes()	Search nodes in explicit index. Replaces START n=node:nodes('key:foo*')	db.index.explicit.searchNodes(indexName :: STRING?, query :: ANY?) :: (node :: NODE?, weight :: FLOAT?)	READ	reader, editor, publisher, architect, admin
db.index.explicit.searchRelationships()	Search relationship in explicit index. Replaces START r=relationship:relIndex('key:foo*')	db.index.explicit.searchRelationships(indexName :: STRING?, query :: ANY?) :: (relationship :: RELATIONSHIP?, weight :: FLOAT?)	READ	reader, editor, publisher, architect, admin
db.index.explicit.searchRelationshipsBetween()	Search relationship in explicit index, starting at the node 'in' and ending at 'out'.	db.index.explicit.searchRelationshipsBetween(indexName :: STRING?, in :: NODE?, out :: NODE?, query :: ANY?) :: (relationship :: RELATIONSHIP?, weight :: FLOAT?)	READ	reader, editor, publisher, architect, admin
db.index.explicit.searchRelationshipsIn()	Search relationship in explicit index, starting at the node 'in'.	db.index.explicit.searchRelationshipsIn(indexName :: STRING?, in :: NODE?, query :: ANY?) :: (relationship :: RELATIONSHIP?, weight :: FLOAT?)	READ	reader, editor, publisher, architect, admin

Name	Description	Signature	Mode	Roles
db.index.explicit.searchRelationshipsOut()	Search relationship in explicit index, ending at the node 'out'.	db.index.explicit.searchRelationshipsOut(indexName :: STRING?, out :: NODE?, query :: ANY?) :: (relationship :: RELATIONSHIP?, weight :: FLOAT?)	READ	reader, editor, publisher, architect, admin
db.index.explicit.seekNodes()	Get node from explicit index. Replaces START n=node:nodes(key = 'A')	db.index.explicit.seekNodes(indexName :: STRING?, key :: STRING?, value :: ANY?) :: (node :: NODE?)	READ	reader, editor, publisher, architect, admin
db.index.explicit.seekRelationships()	Get relationship from explicit index. Replaces START r=relationship:relIndex(key = 'A')	db.index.explicit.seekRelationships(indexName :: STRING?, key :: STRING?, value :: ANY?) :: (relationship :: RELATIONSHIP?)	READ	reader, editor, publisher, architect, admin
db.index.fulltext.awaitEventuallyConsistentIndexRefresh()	Wait for the updates from recently committed transactions to be applied to any eventually-consistent fulltext indexes.	db.index.fulltext.awaitEventuallyConsistentIndexRefresh() :: VOID	READ	reader, editor, publisher, architect, admin
db.index.fulltext.createNodeIndex()	Create a node fulltext index for the given labels and properties. The optional 'config' map parameter can be used to supply settings to the index. Note: index specific settings are currently experimental, and might not replicated correctly in a cluster, or during backup. Supported settings are 'analyzer', for specifying what analyzer to use when indexing and querying. Use the db.index.fulltext.listAvailableAnalyzers procedure to see what options are available. And 'eventually_consistent' which can be set to 'true' to make this index eventually consistent, such that updates from committing transactions are applied in a background thread.	db.index.fulltext.createNodeIndex(indexName :: STRING?, labels :: LIST? OF STRING?, propertyNames :: LIST? OF STRING?, config = {} :: MAP?) :: VOID	SCHEMA	architect, admin

Name	Description	Signature	Mode	Roles
db.index.fulltext.createRelationshipIndex()	Create a relationship fulltext index for the given relationship types and properties. The optional 'config' map parameter can be used to supply settings to the index. Note: index specific settings are currently experimental, and might not replicated correctly in a cluster, or during backup. Supported settings are 'analyzer', for specifying what analyzer to use when indexing and querying. Use the <code>db.index.fulltext.listAvailableAnalyzers</code> procedure to see what options are available. And 'eventually_consistent' which can be set to 'true' to make this index eventually consistent, such that updates from committing transactions are applied in a background thread.	<code>db.index.fulltext.createRelationshipIndex(indexName :: STRING?, relationshipTypes :: LIST? OF STRING?, propertyNames :: LIST? OF STRING?, config = {} :: MAP?) :: VOID</code>	SCHEMA	architect, admin
db.index.fulltext.drop()	Drop the specified index.	<code>db.index.fulltext.drop(indexName :: STRING?) :: VOID</code>	SCHEMA	architect, admin
db.index.fulltext.listAvailableAnalyzers()	List the available analyzers that the fulltext indexes can be configured with.	<code>db.index.fulltext.listAvailableAnalyzers() :: (analyzer :: STRING?)</code>	READ	reader, editor, publisher, architect, admin
db.index.fulltext.queryNodes()	Query the given fulltext index. Returns the matching nodes and their lucene query score, ordered by score.	<code>db.index.fulltext.queryNodes(indexName :: STRING?, queryString :: STRING?) :: (node :: NODE?, score :: FLOAT?)</code>	READ	reader, editor, publisher, architect, admin
db.index.fulltext.queryRelationships()	Query the given fulltext index. Returns the matching relationships and their lucene query score, ordered by score.	<code>db.index.fulltext.queryRelationships(indexName :: STRING?, queryString :: STRING?) :: (relationship :: RELATIONSHIP?, score :: FLOAT?)</code>	READ	reader, editor, publisher, architect, admin
db.indexes()	List all indexes in the database.	<code>db.indexes() :: (description :: STRING?, indexName :: STRING?, tokenNames :: LIST? OF STRING?, properties :: LIST? OF STRING?, state :: STRING?, type :: STRING?, progress :: FLOAT?, provider :: MAP?, id :: INTEGER?, failureMessage :: STRING?)</code>	READ	reader, editor, publisher, architect, admin

Name	Description	Signature	Mode	Roles
db.labels()	List all labels in the database.	db.labels() :: (label :: STRING?)	READ	reader, editor, publisher, architect, admin
db.propertyKeys()	List all property keys in the database.	db.propertyKeys() :: (propertyKey :: STRING?)	READ	reader, editor, publisher, architect, admin
db.relationshipTypes()	List all relationship types in the database.	db.relationshipTypes() :: (relationshipType :: STRING?)	READ	reader, editor, publisher, architect, admin
db.resampleIndex()	Schedule resampling of an index (for example: CALL db.resampleIndex(":Person(name)").)	db.resampleIndex(index :: STRING?) :: VOID	READ	reader, editor, publisher, architect, admin
db.resampleOutdatedIndexes()	Schedule resampling of all outdated indexes.	db.resampleOutdatedIndexes() :: VOID	READ	reader, editor, publisher, architect, admin
db.schema()	Show the schema of the data.	db.schema() :: (nodes :: LIST? OF NODE?, relationships :: LIST? OF RELATIONSHIP?)	READ	reader, editor, publisher, architect, admin
db.schema.nodeTypeProperties()	Show the derived property schema of the nodes in tabular form.	db.schema.nodeTypeProperties() :: (nodeType :: STRING?, nodeLabels :: LIST? OF STRING?, propertyName :: STRING?, propertyTypes :: LIST? OF STRING?, mandatory :: BOOLEAN?)	READ	reader, editor, publisher, architect, admin
db.schema.relTypeProperties()	Show the derived property schema of the relationships in tabular form.	db.schema.relTypeProperties() :: (relType :: STRING?, propertyName :: STRING?, propertyTypes :: LIST? OF STRING?, mandatory :: BOOLEAN?)	READ	reader, editor, publisher, architect, admin
db.schema.visualization()	Visualize the schema of the data. Replaces db.schema.	db.schema.visualization() :: (nodes :: LIST? OF NODE?, relationships :: LIST? OF RELATIONSHIP?)	READ	reader, editor, publisher, architect, admin
dbms.changePassword()	Change the current user's password. Deprecated by dbms.security.changePassword.	dbms.changePassword(password :: STRING?) :: VOID	DBMS	reader, editor, publisher, architect, admin
dbms.clearQueryCaches()	Clears all query caches.	dbms.clearQueryCaches() :: (value :: STRING?)	DBMS	admin
dbms.components()	List DBMS components and their versions.	dbms.components() :: (name :: STRING?, versions :: LIST? OF STRING?, edition :: STRING?)	DBMS	reader, editor, publisher, architect, admin

Name	Description	Signature	Mode	Roles
dbms.functions()	List all user functions in the DBMS.	dbms.functions() :: (name :: STRING?, signature :: STRING?, description :: STRING?, roles :: LIST? OF STRING?)	DBMS	reader, editor, publisher, architect, admin
dbms.getTXMetaData()	Provides attached transaction metadata.	dbms.getTXMetaData() :: (metadata :: MAP?)	DBMS	reader, editor, publisher, architect, admin
dbms.killConnection()	Kill network connection with the given connection id.	dbms.killConnection(id :: STRING?) :: (connectionId :: STRING?, username :: STRING?, message :: STRING?)	DBMS	reader, editor, publisher, architect, admin
dbms.killConnections()	Kill all network connections with the given connection ids.	dbms.killConnections(ids :: LIST? OF STRING?) :: (connectionId :: STRING?, username :: STRING?, message :: STRING?)	DBMS	reader, editor, publisher, architect, admin
dbms.killQueries()	Kill all transactions executing a query with any of the given query ids.	dbms.killQueries(ids :: LIST? OF STRING?) :: (queryId :: STRING?, username :: STRING?, message :: STRING?)	DBMS	reader, editor, publisher, architect, admin
dbms.killQuery()	Kill all transactions executing the query with the given query id.	dbms.killQuery(id :: STRING?) :: (queryId :: STRING?, username :: STRING?, message :: STRING?)	DBMS	reader, editor, publisher, architect, admin
dbms.listActiveLocks()	List the active lock requests granted for the transaction executing the query with the given query id.	dbms.listActiveLocks(queryId :: STRING?) :: (mode :: STRING?, resourceType :: STRING?, resourceId :: INTEGER?)	DBMS	reader, editor, publisher, architect, admin
dbms.listConfig()	List the currently active config of Neo4j.	dbms.listConfig(searchString = :: STRING?) :: (name :: STRING?, description :: STRING?, value :: STRING?, dynamic :: BOOLEAN?)	DBMS	admin
dbms.listConnections()	List all accepted network connections at this instance that are visible to the user.	dbms.listConnections() :: (connectionId :: STRING?, connectTime :: STRING?, connector :: STRING?, username :: STRING?, userAgent :: STRING?, serverAddress :: STRING?, clientAddress :: STRING?)	DBMS	reader, editor, publisher, architect, admin

Name	Description	Signature	Mode	Roles
dbms.listQueries()	List all queries currently executing at this instance that are visible to the user.	dbms.listQueries() :: (queryId :: STRING?, username :: STRING?, metaData :: MAP?, query :: STRING?, parameters :: MAP?, planner :: STRING?, runtime :: STRING?, indexes :: LIST? OF MAP?, startTime :: STRING?, elapsedTime :: STRING?, connectionDetails :: STRING?, protocol :: STRING?, clientAddress :: STRING?, requestUri :: STRING?, status :: STRING?, resourceInformation :: MAP?, activeLockCount :: INTEGER?, elapsedTimeMillis :: INTEGER?, cpuTimeMillis :: INTEGER?, waitTimeMillis :: INTEGER?, idleTimeMillis :: INTEGER?, allocatedBytes :: INTEGER?, pageHits :: INTEGER?, pageFaults :: INTEGER?, connectionId :: STRING?)	DBMS	reader, editor, publisher, architect, admin
dbms.listTransactions()	List all transactions currently executing at this instance that are visible to the user.	dbms.listTransactions() :: (transactionId :: STRING?, username :: STRING?, metaData :: MAP?, startTime :: STRING?, protocol :: STRING?, clientAddress :: STRING?, requestUri :: STRING?, currentQueryId :: STRING?, currentQuery :: STRING?, activeLockCount :: INTEGER?, status :: STRING?, resourceInformation :: MAP?, elapsedTimeMillis :: INTEGER?, cpuTimeMillis :: INTEGER?, waitTimeMillis :: INTEGER?, idleTimeMillis :: INTEGER?, allocatedBytes :: INTEGER?, allocatedDirectBytes :: INTEGER?, pageHits :: INTEGER?, pageFaults :: INTEGER?, connectionId :: STRING?)	DBMS	reader, editor, publisher, architect, admin

Name	Description	Signature	Mode	Roles
dbms.procedures()	List all procedures in the DBMS.	dbms.procedures() :: (name :: STRING?, signature :: STRING?, description :: STRING?, roles :: LIST? OF STRING?, mode :: STRING?)	DBMS	reader, editor, publisher, architect, admin
dbms.queryJmx()	Query JMX management data by domain and name. For instance, "org.neo4j:/*"	dbms.queryJmx(query :: STRING?) :: (name :: STRING?, description :: STRING?, attributes :: MAP?)	DBMS	reader, editor, publisher, architect, admin
dbms.security.activateUser()	Activate a suspended user.	dbms.security.activateUser(username :: STRING?, requirePasswordChange = true :: BOOLEAN?) :: VOID	DBMS	admin
dbms.security.addRoleToUser()	Assign a role to the user.	dbms.security.addRoleToUser(roleName :: STRING?, username :: STRING?) :: VOID	DBMS	admin
dbms.security.changePassword()	Change the current user's password.	dbms.security.changePassword(password :: STRING?, requirePasswordChange = false :: BOOLEAN?) :: VOID	DBMS	reader, editor, publisher, architect, admin
dbms.security.changeUserPassword()	Change the given user's password.	dbms.security.changeUserPassword(username :: STRING?, newPassword :: STRING?, requirePasswordChange = true :: BOOLEAN?) :: VOID	DBMS	admin
dbms.security.clearAuthCache()	Clears authentication and authorization cache.	dbms.security.clearAuthCache() :: VOID	DBMS	admin
dbms.security.createRole()	Create a new role.	dbms.security.createRole(roleName :: STRING?) :: VOID	DBMS	admin
dbms.security.createUser()	Create a new user.	dbms.security.createUser(username :: STRING?, password :: STRING?, requirePasswordChange = true :: BOOLEAN?) :: VOID	DBMS	admin
dbms.security.deleteRole()	Delete the specified role. Any role assignments will be removed.	dbms.security.deleteRole(roleName :: STRING?) :: VOID	DBMS	admin
dbms.security.deleteUser()	Delete the specified user.	dbms.security.deleteUser(username :: STRING?) :: VOID	DBMS	admin
dbms.security.listRoles()	List all available roles.	dbms.security.listRoles() :: (role :: STRING?, users :: LIST? OF STRING?)	DBMS	admin
dbms.security.listRolesForUser()	List all roles assigned to the specified user.	dbms.security.listRolesForUser(username :: STRING?) :: (value :: STRING?)	DBMS	admin

Name	Description	Signature	Mode	Roles
dbms.security.listUsers()	List all local users.	dbms.security.listUsers() :: (username :: STRING?, roles :: LIST? OF STRING?, flags :: LIST? OF STRING?)	DBMS	admin
dbms.security.listUsersForRole()	List all users currently assigned the specified role.	dbms.security.listUsersForRole(roleName :: STRING?) :: (value :: STRING?)	DBMS	admin
dbms.security.removeRoleFromUser()	Unassign a role from the user.	dbms.security.removeRoleFromUser(roleName :: STRING?, username :: STRING?) :: VOID	DBMS	admin
dbms.security.showCurrentUser()	Show the current user. Deprecated by dbms.showCurrentUser.	dbms.security.showCurrentUser() :: (username :: STRING?, roles :: LIST? OF STRING?, flags :: LIST? OF STRING?)	DBMS	reader, editor, publisher, architect, admin
dbms.security.suspendUser()	Suspend the specified user.	dbms.security.suspendUser(username :: STRING?) :: VOID	DBMS	admin
dbms.setConfigValue()	Updates a given setting value. Passing an empty value will result in removing the configured value and falling back to the default value. Changes will not persist and will be lost if the server is restarted.	dbms.setConfigValue(setting :: STRING?, value :: STRING?) :: VOID	DBMS	admin
dbms.setTXMetaData()	Attaches a map of data to the transaction. The data will be printed when listing queries, and inserted into the query log.	dbms.setTXMetaData(data :: MAP?) :: VOID	DBMS	reader, editor, publisher, architect, admin
dbms.showCurrentUser()	Show the current user.	dbms.showCurrentUser() :: (username :: STRING?, roles :: LIST? OF STRING?, flags :: LIST? OF STRING?)	DBMS	reader, editor, publisher, architect, admin

A.2.4. Community Edition procedures

Table 349. Community Edition procedures

Name	Description	Signature	Mode
db.awaitIndex()	Wait for an index to come online (for example: CALL db.awaitIndex(":Person(name)").	db.awaitIndex(index :: STRING?, timeOutSeconds = 300 :: INTEGER?) :: VOID	READ
db.awaitIndexes()	Wait for all indexes to come online (for example: CALL db.awaitIndexes("500").)	db.awaitIndexes(timeOutSeconds = 300 :: INTEGER?) :: VOID	READ
db.constraints()	List all constraints in the database.	db.constraints() :: (description :: STRING?)	READ

Name	Description	Signature	Mode
db.createIndex()	Create a schema index with specified index provider (for example: CALL db.createIndex(":Person(name)", "lucene+native-2.0") - YIELD index, providerName, status	db.createIndex(index :: STRING?, providerName :: STRING?) :: (index :: STRING?, providerName :: STRING?, status :: STRING?)	SCHEMA
db.createLabel()	Create a label	db.createLabel(newLabel :: STRING?) :: VOID	WRITE
db.createProperty()	Create a Property	db.createProperty(newProperty :: STRING?) :: VOID	WRITE
db.createRelationshipType()	Create a RelationshipType	db.createRelationshipType(newRelationshipType :: STRING?) :: VOID	WRITE
db.createUniquePropertyConstraint()	Create a unique property constraint with index backed by specified index provider (for example: CALL db.createUniquePropertyConstraint(":Person(name)", "lucene+native-2.0") - YIELD index, providerName, status	db.createUniquePropertyConstraint(index :: STRING?, providerName :: STRING?) :: (index :: STRING?, providerName :: STRING?, status :: STRING?)	SCHEMA
db.index.explicit.addNode()	Add a node to an explicit index based on a specified key and value	db.index.explicit.addNode(indexName :: STRING?, node :: NODE?, key :: STRING?, value :: ANY?) :: (success :: BOOLEAN?)	WRITE
db.index.explicit.addRelationship()	Add a relationship to an explicit index based on a specified key and value	db.index.explicit.addRelationship(indexName :: STRING?, relationship :: RELATIONSHIP?, key :: STRING?, value :: ANY?) :: (success :: BOOLEAN?)	WRITE
db.index.explicit.auto.searchNodes()	Search nodes in explicit automatic index. Replaces START n=node:node_auto_index('key:foo*')	db.index.explicit.auto.searchNodes(query :: ANY?) :: (node :: NODE?, weight :: FLOAT?)	READ
db.index.explicit.auto.searchRelationships()	Search relationship in explicit automatic index. Replaces START r=relationship:relationships_auto_index('key:foo*')	db.index.explicit.auto.searchRelationships(query :: ANY?) :: (relationship :: RELATIONSHIP?, weight :: FLOAT?)	READ
db.index.explicit.auto.seekNodes()	Get node from explicit automatic index. Replaces START n=node:node_auto_index(key = 'A')	db.index.explicit.auto.seekNodes(key :: STRING?, value :: ANY?) :: (node :: NODE?)	READ
db.index.explicit.auto.seekRelationships()	Get relationship from explicit automatic index. Replaces START r=relationship:relationships_auto_index(key = 'A')	db.index.explicit.auto.seekRelationships(key :: STRING?, value :: ANY?) :: (relationship :: RELATIONSHIP?)	READ
db.index.explicit.drop()	Remove an explicit index - YIELD type,name,config	db.index.explicit.drop(indexName :: STRING?) :: (type :: STRING?, name :: STRING?, config :: MAP?)	WRITE
db.index.explicit.existsForNodes()	Check if a node explicit index exists	db.index.explicit.existsForNodes(indexName :: STRING?) :: (success :: BOOLEAN?)	READ
db.index.explicit.existsForRelationships()	Check if a relationship explicit index exists	db.index.explicit.existsForRelationships(indexName :: STRING?) :: (success :: BOOLEAN?)	READ

Name	Description	Signature	Mode
db.index.explicit.forNodes()	Get or create a node explicit index - YIELD type,name,config	db.index.explicit.forNodes(indexName :: STRING?, config = {} :: MAP?) :: (type :: STRING?, name :: STRING?, config :: MAP?)	WRITE
db.index.explicit.forRelationships()	Get or create a relationship explicit index - YIELD type,name,config	db.index.explicit.forRelationships(indexName :: STRING?, config = {} :: MAP?) :: (type :: STRING?, name :: STRING?, config :: MAP?)	WRITE
db.index.explicit.list()	List all explicit indexes - YIELD type,name,config	db.index.explicit.list() :: (type :: STRING?, name :: STRING?, config :: MAP?)	READ
db.index.explicit.removeNode()	Remove a node from an explicit index with an optional key	db.index.explicit.removeNode(indexName :: STRING?, node :: NODE?, key = <[9895b15e-8693-4a21-a58b-4b7b87e09b8e]> :: STRING?) :: (success :: BOOLEAN?)	WRITE
db.index.explicit.removeRelationship()	Remove a relationship from an explicit index with an optional key	db.index.explicit.removeRelationship(indexName :: STRING?, relationship :: RELATIONSHIP?, key = <[9895b15e-8693-4a21-a58b-4b7b87e09b8e]> :: STRING?) :: (success :: BOOLEAN?)	WRITE
db.index.explicit.searchNodes()	Search nodes in explicit index. Replaces START n=node:node('key:foo*')	db.index.explicit.searchNodes(indexName :: STRING?, query :: ANY?) :: (node :: NODE?, weight :: FLOAT?)	READ
db.index.explicit.searchRelationships()	Search relationship in explicit index. Replaces START r=relationship:relIndex('key:foo*')	db.index.explicit.searchRelationships(indexName :: STRING?, query :: ANY?) :: (relationship :: RELATIONSHIP?, weight :: FLOAT?)	READ
db.index.explicit.searchRelationshipsBetween()	Search relationship in explicit index, starting at the node 'in' and ending at 'out'.	db.index.explicit.searchRelationshipsBetween(indexName :: STRING?, in :: NODE?, out :: NODE?, query :: ANY?) :: (relationship :: RELATIONSHIP?, weight :: FLOAT?)	READ
db.index.explicit.searchRelationshipsIn()	Search relationship in explicit index, starting at the node 'in'.	db.index.explicit.searchRelationshipsIn(indexName :: STRING?, in :: NODE?, query :: ANY?) :: (relationship :: RELATIONSHIP?, weight :: FLOAT?)	READ
db.index.explicit.searchRelationshipsOut()	Search relationship in explicit index, ending at the node 'out'.	db.index.explicit.searchRelationshipsOut(indexName :: STRING?, out :: NODE?, query :: ANY?) :: (relationship :: RELATIONSHIP?, weight :: FLOAT?)	READ
db.index.explicit.seekNodes()	Get node from explicit index. Replaces START n=node:node(key = 'A')	db.index.explicit.seekNodes(indexName :: STRING?, key :: STRING?, value :: ANY?) :: (node :: NODE?)	READ

Name	Description	Signature	Mode
db.index.explicit.seekRelationships()	Get relationship from explicit index. Replaces <code>START r=relationship:relIndex(key = 'A')</code>	<code>db.index.explicit.seekRelationships(indexName :: STRING?, key :: STRING?, value :: ANY?) :: (relationship :: RELATIONSHIP?)</code>	READ
db.index.fulltext.awaitEventuallyConsistentIndexRefresh()	Wait for the updates from recently committed transactions to be applied to any eventually-consistent fulltext indexes.	<code>db.index.fulltext.awaitEventuallyConsistentIndexRefresh() :: VOID</code>	READ
db.index.fulltext.createNodeIndex()	Create a node fulltext index for the given labels and properties. The optional 'config' map parameter can be used to supply settings to the index. Note: index specific settings are currently experimental, and might not replicated correctly in a cluster, or during backup. Supported settings are 'analyzer', for specifying what analyzer to use when indexing and querying. Use the <code>db.index.fulltext.listAvailableAnalyzers</code> procedure to see what options are available. And 'eventually_consistent' which can be set to 'true' to make this index eventually consistent, such that updates from committing transactions are applied in a background thread.	<code>db.index.fulltext.createNodeIndex(indexName :: STRING?, labels :: LIST? OF STRING?, propertyNames :: LIST? OF STRING?, config = {} :: MAP?) :: VOID</code>	SCHEMA
db.index.fulltext.createRelationshipIndex()	Create a relationship fulltext index for the given relationship types and properties. The optional 'config' map parameter can be used to supply settings to the index. Note: index specific settings are currently experimental, and might not replicated correctly in a cluster, or during backup. Supported settings are 'analyzer', for specifying what analyzer to use when indexing and querying. Use the <code>db.index.fulltext.listAvailableAnalyzers</code> procedure to see what options are available. And 'eventually_consistent' which can be set to 'true' to make this index eventually consistent, such that updates from committing transactions are applied in a background thread.	<code>db.index.fulltext.createRelationshipIndex(indexName :: STRING?, relationshipTypes :: LIST? OF STRING?, propertyNames :: LIST? OF STRING?, config = {} :: MAP?) :: VOID</code>	SCHEMA
db.index.fulltext.drop()	Drop the specified index.	<code>db.index.fulltext.drop(indexName :: STRING?) :: VOID</code>	SCHEMA
db.index.fulltext.listAvailableAnalyzers()	List the available analyzers that the fulltext indexes can be configured with.	<code>db.index.fulltext.listAvailableAnalyzers() :: (analyzer :: STRING?)</code>	READ

Name	Description	Signature	Mode
db.index.fulltext.queryNodes()	Query the given fulltext index. Returns the matching nodes and their lucene query score, ordered by score.	db.index.fulltext.queryNodes(indexName :: STRING?, queryString :: STRING?) :: (node :: NODE?, score :: FLOAT?)	READ
db.index.fulltext.queryRelationships()	Query the given fulltext index. Returns the matching relationships and their lucene query score, ordered by score.	db.index.fulltext.queryRelationships(indexName :: STRING?, queryString :: STRING?) :: (relationship :: RELATIONSHIP?, score :: FLOAT?)	READ
db.indexes()	List all indexes in the database.	db.indexes() :: (description :: STRING?, indexName :: STRING?, tokenNames :: LIST? OF STRING?, properties :: LIST? OF STRING?, state :: STRING?, type :: STRING?, progress :: FLOAT?, provider :: MAP?, id :: INTEGER?, failureMessage :: STRING?)	READ
db.labels()	List all labels in the database.	db.labels() :: (label :: STRING?)	READ
db.propertyKeys()	List all property keys in the database.	db.propertyKeys() :: (propertyKey :: STRING?)	READ
db.relationshipTypes()	List all relationship types in the database.	db.relationshipTypes() :: (relationshipType :: STRING?)	READ
db.resampleIndex()	Schedule resampling of an index (for example: CALL db.resampleIndex(":Person.name")).	db.resampleIndex(index :: STRING?) :: VOID	READ
db.resampleOutdatedIndexes()	Schedule resampling of all outdated indexes.	db.resampleOutdatedIndexes() :: VOID	READ
db.schema()	Show the schema of the data.	db.schema() :: (nodes :: LIST? OF NODE?, relationships :: LIST? OF RELATIONSHIP?)	READ
db.schema.nodeTypeProperties()	Show the derived property schema of the nodes in tabular form.	db.schema.nodeTypeProperties() :: (nodeType :: STRING?, nodeLabels :: LIST? OF STRING?, propertyName :: STRING?, propertyTypes :: LIST? OF STRING?, mandatory :: BOOLEAN?)	READ
db.schema.relTypeProperties()	Show the derived property schema of the relationships in tabular form.	db.schema.relTypeProperties() :: (relType :: STRING?, propertyName :: STRING?, propertyTypes :: LIST? OF STRING?, mandatory :: BOOLEAN?)	READ
db.schema.visualization()	Visualize the schema of the data. Replaces db.schema.	db.schema.visualization() :: (nodes :: LIST? OF NODE?, relationships :: LIST? OF RELATIONSHIP?)	READ
dbms.changePassword()	Change the current user's password. Deprecated by dbms.security.changePassword.	dbms.changePassword(password :: STRING?) :: VOID	DBMS
dbms.clearQueryCaches()	Clears all query caches.	dbms.clearQueryCaches() :: (value :: STRING?)	DBMS

Name	Description	Signature	Mode
dbms.components()	List DBMS components and their versions.	dbms.components() :: (name :: STRING?, versions :: LIST? OF STRING?, edition :: STRING?)	DBMS
dbms.functions()	List all user functions in the DBMS.	dbms.functions() :: (name :: STRING?, signature :: STRING?, description :: STRING?)	DBMS
dbms.listConfig()	List the currently active config of Neo4j.	dbms.listConfig(searchString = :: STRING?) :: (name :: STRING?, description :: STRING?, value :: STRING?, dynamic :: BOOLEAN?)	DBMS
dbms.procedures()	List all procedures in the DBMS.	dbms.procedures() :: (name :: STRING?, signature :: STRING?, description :: STRING?, mode :: STRING?)	DBMS
dbms.queryJmx()	Query JMX management data by domain and name. For instance, "org.neo4j:*	dbms.queryJmx(query :: STRING?) :: (name :: STRING?, description :: STRING?, attributes :: MAP?)	DBMS
dbms.security.changePassword()	Change the current user's password.	dbms.security.changePassword(password :: STRING?) :: VOID	DBMS
dbms.security.createUser()	Create a new user.	dbms.security.createUser(username :: STRING?, password :: STRING?, requirePasswordChange = true :: BOOLEAN?) :: VOID	DBMS
dbms.security.deleteUser()	Delete the specified user.	dbms.security.deleteUser(username :: STRING?) :: VOID	DBMS
dbms.security.listUsers()	List all local users.	dbms.security.listUsers() :: (username :: STRING?, flags :: LIST? OF STRING?)	DBMS
dbms.security.showCurrentUser()	Show the current user. Deprecated by dbms.showCurrentUser.	dbms.security.showCurrentUser() :: (username :: STRING?, flags :: LIST? OF STRING?)	DBMS
dbms.showCurrentUser()	Show the current user.	dbms.showCurrentUser() :: (username :: STRING?, flags :: LIST? OF STRING?)	DBMS

A.3. User management for Community Edition

This section describes 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 sections provide comparisons between functionality available in Enterprise Edition and Community Edition:

- [Native roles](#) — Descriptions of roles and privileges.
- [Procedures for native user and role management](#) — Listings of built-in procedures for user management.

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
<code>dbms.security.changePassword</code>	Change the current user's password
<code>dbms.security.createUser</code>	Add a user
<code>dbms.security.deleteUser</code>	Delete a user
<code>dbms.security.listUsers</code>	List all users
<code>dbms.showCurrentUser</code>	Show details for the current user

A.3.1. Change the current user's password

The `current user` is able to change their own password at any time.

Syntax:

```
CALL dbms.security.changePassword(password)
```

Arguments:

Name	Type	Description
<code>password</code>	String	This is the new password for the current user.

Exceptions:

The password is the empty string.

The password is the same as the current user's previous password.

Example 114. Change the current user's password

The following example changes the password of the `current user` to '`h6u4%kr`'.

```
CALL dbms.security.changePassword('h6u4%kr')
```

A.3.2. 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
<code>username</code>	String	This is the user's username.

Name	Type	Description
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 115. 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)
```

A.3.3. 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 <code>user</code> 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 116. Delete a user

The following example deletes a [user](#) with the username 'janebrown'.

```
CALL dbms.security.deleteUser('janebrown')
```

A.3.4. List all 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 117. 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
```

A.3.5. Show details for the current user

The [current user](#) is able to view whether or not they need to change their password.

Syntax:

```
CALL dbms.showCurrentUser()
```

Returns:

Name	Type	Description
username	String	This is the user's username.
flags	List<String>	This is a flag indicating whether the user needs change their password.

Example 118. Show details for the current user

The following example shows that the [current user](#) — with the username 'johnsmith' — does not need to change his password.

```
CALL dbms.showCurrentUser()
```

```
+-----+  
| username | flags |  
+-----+  
| "johnsmith" | [] |  
+-----+  
1 row
```

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](#)
- [Manage users and roles](#)

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.

In this section we will learn how to deploy a Causal Cluster locally, on a single machine. This is useful as a learning exercise and to get started quickly developing an application against a Neo4j Causal Cluster. A cluster on a single machine has no fault tolerance and is not suitable for production use.

We will begin by configuring and starting a cluster of three Core instances. This is the minimal deployment for a fault-tolerant Causal Cluster (for a discussion on the number of servers required for a Causal Cluster, see [Core Servers](#)). The Core instances are responsible for keeping the data safe.

After the Core of the cluster is operational we will add three Read Replicas. The Read Replicas are responsible for scaling the capacity of the cluster.

The core of the Causal Cluster remains stable over time. The roles within the core will change as needed but the core itself is long-lived and stable. At the edge of the cluster, the Read Replicas are cheap and disposable. They can be added as needed to increase the operational capacity of the cluster as a whole.

B.1.1. Download and configure

Some of the configuration for the instances in the cluster will be identical. A convenient way to go about the configuration is therefore:

1. Create a local working directory.
2. Download a copy of Neo4j Enterprise Edition from [the Neo4j download site](#) (<https://neo4j.com/download/other-releases/#releases>).
3. Unpack Neo4j in the working directory.
4. Make a copy of the `neo4j-enterprise-3.5.0` directory and name it `core-01/` or similar. Keep the original directory to use when setting up the Read Replicas later. The `core-01/` directory will contain the first Core instance.
5. Complete the configuration (see below) for the first Core instance. Then make two copies of the `core-01/` directory and name them `core-02/` and `core-03/`.
6. Proceed with the configuration of the two copied instances. Those changes that are common to all three Core instances are now already in place for Cores number two and three.

In this example we are running all instances in the cluster on a single machine. Many of the default configuration settings work well out of the box in a production deployment, with multiple machines.

Some of these we have to change when deploying multiple instances on a single machine, so that instances do not try to use the same network ports. We call out the settings that are specific to this scenario as we go along.

B.1.2. Configure the Core instances

All configuration that we will do takes place in the Neo4j configuration file, `conf/neo4j.conf`. If you used a different package than in the download instructions above, see [File locations](#) to locate the configuration file. Look in the configuration file for a section labeled "Causal Clustering Configuration".

Minimum configuration

The minimum configuration for a Core instance requires setting the following:

`dbms.mode`

The operating mode of this instance. Uncomment this setting and give it the value `CORE`.

`causal_clustering.minimum_core_cluster_size_atFormation`

The minimum number of Core machines in the cluster at formation. Uncomment this setting and give it the value `3`.

`causal_clustering.minimum_core_cluster_size_at_runtime`

The minimum number of Core instances which will exist in the consensus group. Uncomment this setting and give it the value `3`.

`causal_clustering.initial_discovery_members`

The network addresses of Core cluster members to be used to discover the cluster when this instance joins. Uncomment this setting and give it the value

`localhost:5000,localhost:5001,localhost:5002`.

Additional configuration

Since we are setting up the Causal Cluster to run on a single machine, we must do some additional configuration that is not necessary when the instances are running on different servers. We configure the following:

`causal_clustering.discovery_listen_address`

The port used for discovery between machines. Uncomment this setting and give it the value `:5000`. On the other two instances, give them the values `:5001` and `:5002`, respectively.

`causal_clustering.transaction_listen_address`

The internal transaction communication address. Uncomment this setting and give it the value `:6000`. On the other two instances, give them the values `:6001` and `:6002`, respectively.

`causal_clustering.raft_listen_address`

The internal consensus mechanism address. Uncomment this setting and give it the value `:7000`. On the other two instances give them the values `:7001` and `:7002`, respectively.

`dbms.connector.bolt.listen_address`

The Bolt connector address. Uncomment this setting and give it the value `:7687`. On the other two instances give them the values `:7688` and `:7689`, respectively.

`dbms.connector.http.listen_address`

The HTTP connector address. Uncomment this setting and give it the value `:7474`. On the other two instances give them the values `:7475` and `:7476`, respectively.

dbms.connector.https.listen_address

The HTTPS connector address. Uncomment this setting and give it the value :6474. On the other two instances give them the values :6475 and :6476, respectively.

dbms.backup.address

The address for online backups. Uncomment this setting and give it the value :6362. On the other two instances give them the values :6363 and :6364, respectively.

B.1.3. Start the Neo4j servers

Start each Neo4j instance as usual. The startup order does not matter.

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

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

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

Startup Time



If you want to follow along with the startup of a server you can follow the messages in `logs/neo4j.log`. On a Unix system issue the command `tail -f 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.

B.1.4. Check the status of the cluster

Now the minimal cluster of three Core Servers is operational and ready to serve requests. Connect to either of the three instances to check the cluster status. Point your web browser to `http://localhost:7474`. Authenticate with the default `neo4j/neo4j` and set a new password. These credentials are not shared between cluster members. A new password must be set on each instance when connecting for the first time. For production deployment we advise integrating the Neo4j cluster with your directory service. See [Integration with LDAP](#) for more details.

Once you have authenticated you can check the status of the cluster by running the query: `CALL dbms.cluster.overview()`. The output will look similar to the following.

Table 350. Cluster overview with `dbms.cluster.overview()`

id	addresses	role	groups
08eb9305-53b9-4394-9237-0f0d63bb05d5	[bolt://localhost:7687, http://localhost:7474, https://localhost:6474]	LEADER	[]
cb0c729d-233c-452f-8f06-f2553e08f149	[bolt://localhost:7688, http://localhost:7475, https://localhost:6475]	FOLLOWER	[]
ded9eed2-dd3a-4574-bc08-6a569f91ec5c	[bolt://localhost:7689, http://localhost:7476, https://localhost:6476]	FOLLOWER	[]

The three Core instances in the cluster are operational.

B.1.5. Test the cluster

Now you can run queries to create nodes and relationships, and see that the data gets replicated in the cluster.

When developing an application against a Neo4j Causal Cluster it is not necessary to know about the roles of the cluster members. The Neo4j Bolt driver creates sessions with access mode *READ* or *WRITE* on request. It is the driver's responsibility to identify the best cluster member to service the session according to need.

When connecting directly with Neo4j Browser, however, we need to be more aware of the roles that the cluster members have. It is easy to navigate from member to member by running the `:sysinfo` command. The sysinfo view contains information about the Neo4j instance. If the instance participates in a Causal Clustering cluster then this view contains a table: **Causal Clustering Cluster Members**. This table contains the same information as provided by the `dbms.cluster.overview()` procedure, but here you can also take action on the other members of the cluster. Run the `:sysinfo` command and click the `Open` action on the instance that has the *LEADER* role. This opens a new Browser session against the Leader of the cluster.

Authenticate and set a new password, as before. Now you can run a query to create nodes and relationships.

```
UNWIND range(0, 100) AS value
MERGE (person1:Person {id: value})
MERGE (person2:Person {id:toInt(100.0 * rand())})
MERGE (person1)-[:FRIENDS]->(person2)
```

When the query has executed choose an instance with the *FOLLOWER* role from the sysinfo view. Click the `Open` action to connect. Now you can run a query to see that the data has been replicated.

```
MATCH path = (person:Person)-[:FRIENDS]-(friend)
RETURN path
LIMIT 10
```

B.1.6. Configure the Read Replicas

Setting up the Read Replicas is similar to setting up the Cores, but simpler.

1. In your working directory, rename the `neo4j-enterprise-3.5.0` directory and name it `replica-01/` or similar. The `replica-01/` directory will contain the first Read Replica.
2. Complete the configuration (see below) for the first Core instance. Then make two copies of the `replica-01/` directory and name them `replica-02/` and `replica-03/`.
3. Proceed with the configuration of the two copied instances. Those changes that are common to all three Read Replicas are now already in place for replicas number two and three.

Configuring a Read Replica is similar to configuring a Core. Read Replicas instances do not participate in quorum decisions, so their configuration is simpler. All that a Read Replica needs to know is the addresses of Core Servers which they can bind to in order to discover the cluster. See [Discovery protocol](#) for details. Once it has completed the initial discovery, the Read Replica becomes aware of the currently available Core Servers and can choose an appropriate one from which to catch up. See [Catchup protocol](#) for details.

Minimum configuration

The minimum configuration for a Read Replica instance requires setting the following:

`dbms.mode`

The operating mode of this instance. Uncomment this setting and give it the value `READ_REPLICA`.

`causal_clustering.initial_discovery_members`

The network addresses of Core cluster members to be used to discover the cluster when this instance joins. Uncomment this setting and give it the value
`localhost:5000,localhost:5001,localhost:5002`.

Additional configuration

Similarly to the Core Cluster, we have to do some additional configuration to enable all the Read Replicas to run on the same machine. We configure the following:

`causal_clustering.transaction_listen_address`

The internal transaction communication address. Uncomment this setting and give it the value `:6003`. On the other two instances, give them the values `:6004` and `:6005`, respectively.

`dbms.connector.bolt.listen_address`

The Bolt connector address. Uncomment this setting and give it the value `:7690`. On the other two instances, give them the values `:7691` and `:7692`, respectively.

`dbms.connector.http.listen_address`

The HTTP connector address. Uncomment this setting and give it the value `:7477`. On the other two instances, give them the values `:7478` and `:7479`, respectively.

`dbms.connector.https.listen_address`

The HTTPS connector address. Uncomment this setting and give it the value `:6477`. On the other two instances, give them the values `:6478` and `:6479`, respectively.

`dbms.backup.address`

The address for online backups. Uncomment this setting and give it the value `:6365`. On the other two instances give them the values `:6366` and `:6367`, respectively.

B.1.7. Test the cluster with Read Replicas

Connect to any of the instances and run `CALL dbms.cluster.overview()` to see the new overview of the cluster. With Read Replicas added the overview will look similar to:

Table 351. Cluster overview with `dbms.cluster.overview()`

<code>id</code>	<code>addresses</code>	<code>role</code>	<code>groups</code>
08eb9305-53b9-4394-9237-0f0d63bb05d5	[bolt://localhost:7687, http://localhost:7474, https://localhost:6474]	LEADER	[]
cb0c729d-233c-452f-8f06-f2553e08f149	[bolt://localhost:7688, http://localhost:7475, https://localhost:6475]	FOLLOWER	[]
ded9eed2-dd3a-4574-bc08-6a569f91ec5c	[bolt://localhost:7689, http://localhost:7476, https://localhost:6476]	FOLLOWER	[]
00000000-0000-0000-0000-000000000000	[bolt://localhost:7690, http://localhost:7477, https://localhost:6477]	READ_REPLICA	[]
00000000-0000-0000-0000-000000000000	[bolt://localhost:7691, http://localhost:7478, https://localhost:6478]	READ_REPLICA	[]

id	addresses	role	groups
00000000-0000-0000-0000-000000000000	[bolt://localhost:7692, http://localhost:7479, https://localhost:6479]	READ_REPLICA	[]

To test that the Read Replicas have successfully caught up with the cluster use :sysinfo and click the Open action to connect to one of the Read Replicas. Issue the same query as before:

```
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 identifier, a *node identifier*, in order to be able to create relationships between nodes in the same process. Relationships are created by connecting the node identifiers. In the examples below, the node identifiers are stored as properties on the nodes. Node identifiers may be of interest later for cross-reference to other systems, traceability etc., but they are not mandatory. If you do not want the identifiers 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. For doing so simply omit 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 `graph.db` (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 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
```

Finally we have the roles that an actor plays in a movie, which will be represented by relationships in the database. 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
```

Now start up a database from the target directory:

```
neo4j_home$ ./bin/neo4j start
```

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 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"
```

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. See also [Using regular expressions for specifying multiple input files](#).

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"
```

Using regular expressions for specifying multiple input files

To simplify using the command line when there are many data source files, the file names can be specified using regular expressions. Each file name that matches the regular expression will be included. Additionally, the files will be sorted according to name. One particular benefit is that the matching is aware of numbers inside the file names and will sort them accordingly, without the need for padding with zeros.

Consider the file names:

- `movies4-header.csv`
- `movies4-part1.csv`
- `movies4-part2.csv`
- `movies4-part12.csv`

For example, the regular expression `movies4.*` will include all of the above files, and keep them in the displayed order.

The use of regular expressions should not be confused with [file globbing](#) ([https://en.wikipedia.org/wiki/Glob_\(programming\)](https://en.wikipedia.org/wiki/Glob_(programming))).

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, as mentioned above. 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`.

Note that for the import to work correctly, the header file must be specified as the first file in the list. For example, let's assume that the file names are instead the following:

- *movies4-header.csv*
- *movies4-data1.csv*
- *movies4-data2.csv*
- *movies4-data12.csv*

In this case, if we use the regular expression `movies4.*`, the sorting will place the header file last and the import will be incorrect. 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.*"
```

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

Each node processed by `neo4j-admin import` must provide a unique id. This id is used to find the correct nodes when creating 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 `--ignore-missing-nodes` 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 [--ignore-missing-nodes](#) 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 `--ignore-missing-nodes` flag:

```
neo4j_home$ bin/neo4j-admin import --nodes import/movies8a.csv --nodes import/actors8a.csv --relationships
import/roles8a.csv --ignore-missing-nodes
```

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 `--ignore-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 `--ignore-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 `--ignore-duplicate-nodes` flag:

```
neo4j_home$ bin/neo4j-admin import --nodes import/actors8b.csv --ignore-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. Manage users and roles

This section describes scenarios of using the Neo4j Security features for native user and role management.

In this section, we show two cases of how [native user and role management](#) can be combined to cater for various real-world scenarios. These examples are not applicable to scenarios where LDAP is used.

The examples in this section assume the existence of an [administrator](#) and a fictitious developer called Jane, who requires access to the database.

B.3.1. Creating a user and managing roles

Step 1: Administrator creates a user

The administrator creates a user on the system with username '`jane`' and password '`abracadabra`', requiring that as soon as Jane logs in for the first time, she is required to change her password immediately:

```
CALL dbms.security.createUser('jane', 'abracadabra', true)
```

Step 2: Administrator assigns the `publisher` role to the user

The administrator assigns the `publisher` role to Jane allowing her to both read and write data:

```
CALL dbms.security.addRoleToUser('publisher', 'jane')
```

Step 3: User logs in and changes her password

As soon as Jane logs in, she is prompted to change her password.

She changes it to '`R0ckyR0ad88`':

```
CALL dbms.security.changePassword('R0ckyR0ad88')
```

Step 4: User writes data

Jane executes a query which inserts some data:

```
CREATE (:Person {name: 'Sam' age: 19})
```

```
+-----+  
| No data returned. |  
+-----+  
Nodes created: 1  
Properties set: 2  
Labels added: 1
```

Step 5: Administrator removes the **publisher** role from the user

The administrator removes the **publisher** role from Jane.

```
CALL dbms.security.removeRoleFromUser('publisher', 'jane')
```

Step 6: User attempts to read data

Jane tries to execute a read query:

```
MATCH (p:Person)  
RETURN p.name
```

The query fails, as Jane does not have the role allowing her to read data (in fact, she has no assigned roles):

```
Read operations are not allowed for user 'jane' with no roles.
```

Step 7: Administrator assigns the **reader** role to the user

The administrator assigns the **reader** role to Jane:

```
CALL dbms.security.addRoleToUser('reader', 'jane')
```

Step 8: User attempts to write data

Jane tries to execute a write query:

```
CREATE (:Person {name: 'Bob' age: 52})
```

The query fails, as Jane does not have the role allowing her to write data.

```
Write operations are not allowed for user 'jane' with roles ['reader'].
```

Step 9: User attempts to read data

Jane tries to execute a read query:

```
MATCH (p:Person)
RETURN p.name
```

The query succeeds as she is assigned the **reader** role:

```
+-----+
| name   |
+-----+
| "Sam"  |
+-----+
1 row
```

B.3.2. Suspending and reactivating a user

This scenario follows on from the one above.

Step 1: Administrator suspends the user

The administrator suspends Jane.

```
CALL dbms.security.suspendUser('jane')
```

Step 2: Suspended user tries to log in

Jane tries to log in to the system, and will fail to do so.

Step 3: Administrator activates suspended user

The administrator activates Jane.

```
CALL dbms.security.activateUser('jane')
```

Step 4: Activated user logs in

Jane is now able to log in successfully.

Appendix C: HA cluster

The functionality described in this section been deprecated and will be removed in Neo4j 4.0.



The functionality described in this section been deprecated and will be removed in Neo4j 4.0. [Causal Clustering](#) should be used instead.

This appendix contains the following:

- [Architecture](#)
- [Setup and configuration](#)
 - [Basic installation](#)
 - [Important configuration settings](#)
 - [Arbiter instances](#)
 - [HAProxy for load balancing](#)
- [Neo4j in Highly Available mode on Docker](#)
- [Status information](#)
- [Upgrade](#)
- [Backup and restore](#)
- [Metrics](#)
- [Tutorials](#)
 - [Set up an HA cluster](#)
 - [Set up a local HA cluster](#)

C.1. Architecture

This section describes the architecture of a Neo4j HA cluster.

A Neo4j HA cluster is comprised of a single master instance and zero or more slave instances. All instances in the cluster have full copies of the data in their local database files. The basic cluster configuration consists of three instances:

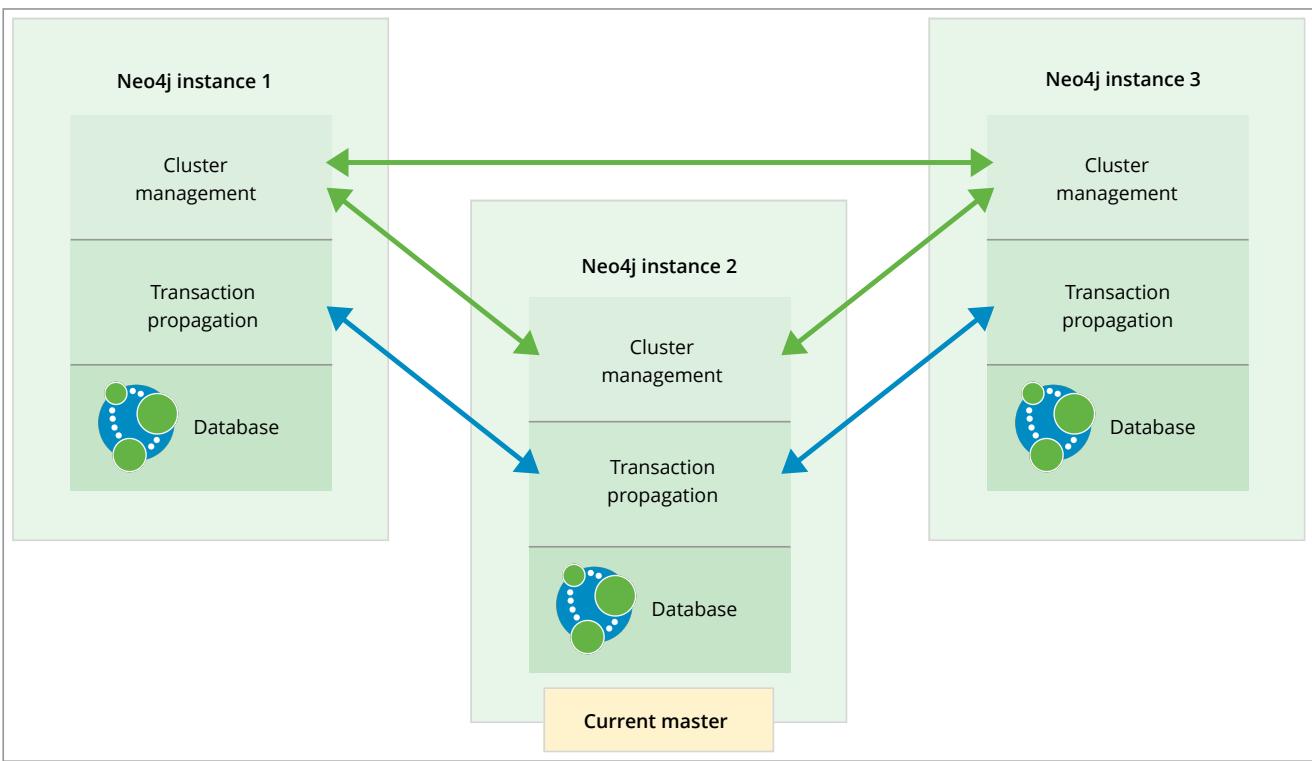


Figure 24. A Neo4j HA cluster

Each instance contains the logic needed in order to coordinate with the other members of the cluster for data replication and election management, represented by the green arrows in the picture above. Each slave instance that is not an arbiter instance (see below) communicates with the master to keep databases up to date, as represented by the blue arrows in the picture above.

C.1.1. Arbiter instance

A special case of a slave instance is the arbiter instance. The arbiter instance comprises the full Neo4j software running in an arbiter mode, such that it participates in cluster communication, but it does not replicate a copy of the datastore.

C.1.2. Transaction propagation

Write transactions performed directly on the master will execute as though the instance were running in non-cluster mode. On success the transaction will be pushed out to a configurable number of slaves. This is done optimistically, meaning that if the push fails, the transaction will still be successful.

When performing a write transaction on a slave each write operation will be synchronized with the master. Locks will be acquired on both master and slave. When the transaction commits it will first be committed on the master and then, if successful, on the slave. To ensure consistency, a slave must be up to date with the master before performing a write operation. The automatic updating of slaves is built into the communication protocol between slave and master.

C.1.3. Failover

Whenever a Neo4j database becomes unavailable, for example caused by hardware failure or network outage, the other instances in the cluster will detect that and mark it as temporarily failed. A database instance that becomes available after an outage will automatically catch up with the cluster.

If the master goes down another member will be elected and have its role switched from slave to master after quorum (see below) has been reached within the cluster. When the new master has performed its role switch, it will broadcast its availability to all the other members of the cluster. Normally a new master is elected and started within seconds. During this time no writes can take

place

Quorum

A cluster must have quorum to elect a new master. Quorum is defined as: *more than 50% of active cluster members*. A simple rule of thumb when designing a cluster is: A cluster that must be able to tolerate n master instance failures requires $2n+1$ instances to satisfy quorum and allow elections to take place. Therefore, the simplest valid cluster size is three instances, which allows for a single master failure.

Election Rules

1. If a master fails, or on a cold-start of the cluster, the slave with the highest committed transaction ID will be elected as the new master. This rule ensures that the slave with the most up-to-date datastore becomes the new master.
2. If a master fails and two or more slaves are tied, i.e. have the same highest committed transaction ID, the slave with the lowest ha.server_id value will be elected the new master. This is a good tie-breaker because the ha.server_id is unique within the cluster, and allows for configuring which instances can become master before others.

C.1.4. Branching

Data branching can be caused in two different ways:

- A slave falls too far behind the master and then leaves or re-joins the cluster. This type of branching is harmless.
- The master re-election happens and the old master has one or more committed transactions that the slaves did not receive before it died. This type of branching is harmful and requires action.

The database makes the best of the situation by creating a directory with the contents of the database files from before branching took place so that it can be reviewed and the situation be resolved. Data branching does not occur under normal operations.

C.1.5. Summary

All this can be summarized as:

- Write transactions can be performed on any database instance in a cluster.
- A Neo4j HA cluster is fault tolerant and can continue to operate from any number of machines down to a single machine.
- Slaves will be automatically synchronized with the master on write operations.
- If the master fails, a new master will be elected automatically.
- The cluster automatically handles instances becoming unavailable (for example due to network issues), and also makes sure to accept them as members in the cluster when they are available again.
- Transactions are atomic, consistent and durable but eventually propagated out to other slaves.
- Updates to slaves are eventually consistent by nature but can be configured to be pushed optimistically from master during commit.
- If the master goes down, any running write transaction will be rolled back and new transactions will block or fail until a new master has become available.
- Reads are highly available and the ability to handle read load scales with more database instances in the cluster.

C.2. Setup and configuration

This section describes how to setup and configure a Neo4j HA cluster.

This section describes the following:

- [Basic installation](#)
- [Important configuration settings](#)
- [Arbiter instances](#)
- [HAProxy for load balancing](#)

C.2.1. Basic installation

This appendix outlines the steps for a basic installation of an HA cluster.

Neo4j can be configured in cluster mode to accommodate differing requirements for load, fault tolerance and available hardware.

Follow these steps in order to configure a Neo4j HA cluster:

1. Download and install the Neo4j Enterprise Edition on each of the servers to be included in the cluster.
2. If applicable, decide which server(s) that are to be configured as arbiter instance(s).
3. Edit the Neo4j configuration file on each of the servers to accommodate the design decisions.
4. Follow installation instructions for a [single instance installation](#).
5. Modify the configuration files on each server as outlined in the section below. There are many parameters that can be modified to achieve a certain behavior. However, the only ones mandatory for an initial cluster are: `dbms.mode`, `ha.server_id` and `ha.initial_hosts`.

C.2.2. Important configuration settings

This section lists the most important configuration options for setting up an HA cluster.

At startup of a Neo4j HA cluster, each Neo4j instance contacts the other instances as configured. When an instance establishes a connection to any other, it determines the current state of the cluster and ensures that it is eligible to join. To be eligible the Neo4j instance must host the same database store as other members of the cluster (although it is allowed to be in an older state), or be a new deployment without a database store.

Please note that IP addresses or hostnames should be explicitly configured for the machines participating in the cluster. In the absence of a specified IP address, Neo4j will attempt to find a valid interface for binding. This is not recommended practice.

`dbms.mode`

`dbms.mode` configures the operating mode of the database.

For cluster mode it is set to: `dbms.mode=HA`

`ha.server_id`

`ha.server_id` is the cluster identifier for each instance. It must be a positive integer and must be

unique among all Neo4j instances in the cluster.

For example, `ha.server_id=1`.

`ha.host.coordination`

`ha.host.coordination` is an address/port setting that specifies where the Neo4j instance will listen for cluster communication. The default port is `5001`.

For example, `ha.host.coordination=192.168.33.22:5001` will listen for cluster communications on port 5001.

`ha.initial_hosts`

`ha.initial_hosts` is a comma separated list of address/port pairs, which specifies how to reach other Neo4j instances in the cluster (as configured via their `ha.host.coordination` option). These hostname/ports will be used when the Neo4j instances start, to allow them to find and join the cluster. When cold starting the cluster, i.e. when no cluster is available yet, the database will be unavailable until all members listed in `ha.initial_hosts` are online and communicating with each other. It is good practice to configure all the instances in the cluster to have the exact same entries in `ha.initial_hosts`, for the cluster to come up quickly and cleanly.

Do not use any whitespace in this configuration option.

For example, `ha.initial_hosts=192.168.33.21:5001,192.168.33.22:5001,192.168.33.23:5001` will initiate a cluster containing the hosts 192.168.33.21-33, all listening on the same port, 5001.

`ha.host.data`

`ha.host.data` is an address/port setting that specifies where the Neo4j instance will listen for transactions from the cluster master. The default port is `6001`.

`ha.host.data` must use a different port than `ha.host.coordination`.

For example, `ha.host.data=192.168.33.22:6001` will listen for transactions from the cluster master on port 6001.

`ha.join_timeout`

`ha.join_timeout` describes the time limit for all members of the `ha.initial_hosts` to start before giving up forming the cluster. The default value is 30 seconds. With the default value each of the instances defined in `ha.initial_hosts` must be started within those 30 seconds for the cluster to successfully form.

Address and port formats

The `ha.host.coordination` and `ha.host.data` configuration options are specified as `<hostname or IP address>:<port>`.

For `ha.host.data` the address must be an address assigned to one of the host's network interfaces.



For `ha.host.coordination` the address must be an address assigned to one of the host's network interfaces, or the value `0.0.0.0`, which will cause Neo4j to listen on every network interface.

Either the address or the port can be omitted, in which case the default for that part will be used. If the hostname or IP address is omitted, then the port must be preceded with a colon (eg. `:5001`).

The syntax for setting a port range is: `<hostname or IP address>:<first port>[-<second port>]`. In this case, Neo4j will test each port in sequence, and select the first that is unused. Note that this usage is not permitted when the hostname is specified as `0.0.0.0` (the "all interfaces" address).

C.2.3. Arbiter instances

This section describes how to configure an arbiter instance for a Neo4j HA cluster.

A typical deployment of Neo4j will use a cluster of three machines to provide fault tolerance and read scalability.

While having at least three instances is necessary for failover to happen in case the master becomes unavailable, it is not required for all instances to run the full Neo4j stack. Instead, something called *arbiter instances* can be deployed. They are regarded as cluster participants in that their role is to take part in master elections with the single purpose of breaking ties in the election process. That makes possible a scenario where you have a cluster of two Neo4j database instances and an additional arbiter instance and still enjoy tolerance of a single failure of either of the three instances.

Arbiter instances are configured in `neo4j.conf` using the same settings as standard Neo4j HA cluster members. The instance is configured to be an arbiter by setting the `dbms.mode` option to `ARBITER`. Settings that are not cluster specific are of course ignored, so you can easily start up an arbiter instance in place of a properly configured Neo4j instance.

To start the arbiter instance, run `neo4j` as normal:

```
neo4j_home$ ./bin/neo4j start
```

You can stop, install and remove it as a service and ask for its status in exactly the same way as for other Neo4j instances.

C.2.4. HAProxy for load balancing

This section describes how to configure HAProxy for load balancing in a Neo4j HA cluster.

In the Neo4j HA architecture, the cluster is typically fronted by a load balancer. In this section we will explore how to set up HAProxy to perform load balancing across the HA cluster.

For this tutorial we will assume a Linux environment with HAProxy already installed. See

<http://www.haproxy.org/> for downloads and installation instructions.

Configuring HAProxy for the Bolt Protocol

In a typical HA deployment, HAProxy will be configured with two open ports, one for routing write operations to the master and one for load balancing read operations over slaves. Each application will have two driver instances, one connected to the master port for performing writes and one connected to the slave port for performing reads.

First we set up the mode and timeouts. The settings below will kill the connection if a server or a client is idle for longer than two hours. Long-running queries may take longer time, but this can be taken care of by enabling HAProxy's TCP heartbeat feature.

```
defaults
  mode      tcp
  timeout connect 30s
  timeout client 2h
  timeout server 2h
```

Set up where drivers wanting to perform writes will connect:

```
frontend neo4j-write
  bind *:7680
  default_backend current-master
```

Now we set up the backend that points to the current master instance.

```
backend current-master
  option httpchk HEAD /db/manage/server/ha/master HTTP/1.0
  server db01 10.0.1.10:7687 check port 7474
  server db02 10.0.1.11:7687 check port 7474
  server db03 10.0.1.12:7687 check port 7474
```

In the example above `httpchk` is configured in the way you would do it if authentication has been disabled for Neo4j. By default however, authentication is enabled and you will need to pass in an authentication header. This would be along the lines of `option httpchk HEAD /db/manage/server/ha/master HTTP/1.0\r\nAuthorization: Basic \ bmVvNGo6bmVvNGo=` where the last part has to be replaced with a base64 encoded value for your username and password.

Configure where drivers wanting to perform reads will connect:

```
frontend neo4j-read
  bind *:7681
  default_backend slaves
```

Finally, configure a backend that points to slaves in a round-robin fashion:

```
backend slaves
  balance roundrobin
  option httpchk HEAD /db/manage/server/ha/slave HTTP/1.0
  server db01 10.0.1.10:7687 check port 7474
  server db02 10.0.1.11:7687 check port 7474
  server db03 10.0.1.12:7687 check port 7474
```

Note that the servers in the `slave` backend are configured the same way as in the `current-master`

backend.

Then by putting all the above configurations into one file, we get a basic workable HAProxy configuration to perform load balancing for applications using the Bolt Protocol.

By default, encryption is enabled between servers and drivers. With encryption turned on, the HAProxy configuration constructed above needs no change to work directly in TLS/SSL passthrough layout for HAProxy. However depending on the driver authentication strategy adopted, some special requirements might apply to the server certificates.

For drivers using trust-on-first-use authentication strategy, each driver would register the HAProxy port it connects to with the first certificate received from the cluster. Then for all subsequent connections, the driver would only establish connections with the server whose certificate is the same as the one registered. Therefore, in order to make it possible for a driver to establish connections with all instances in the cluster, this mode requires all the instances in the cluster sharing the same certificate.

If drivers are configured to run in trusted-certificate mode, then the certificate known to the drivers should be a root certificate to all the certificates installed on the servers in the cluster. Alternatively, for the drivers such as Java driver who supports registering multiple certificates as trusted certificates, the drivers also work well with a cluster if server certificates used in the cluster are all registered as trusted certificates.

To use HAProxy with other encryption layout, please refer to their full documentation at their website.

Configuring HAProxy for the HTTP API

HAProxy can be configured in many ways. The full documentation is available at their website.

For this example, we will configure HAProxy to load balance requests to three HA servers. Simply write the following configuration to `/etc/haproxy/haproxy.cfg`:

```
global
    daemon
    maxconn 256

defaults
    mode http
    timeout connect 5000ms
    timeout client 50000ms
    timeout server 50000ms

frontend http-in
    bind *:80
    default_backend neo4j

backend neo4j
    option httpchk GET /db/manage/server/ha/available
    server s1 10.0.1.10:7474 maxconn 32
    server s2 10.0.1.11:7474 maxconn 32
    server s3 10.0.1.12:7474 maxconn 32

listen admin
    bind *:8080
    stats enable
```

HAProxy can now be started by running:

```
/usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg
```

You can connect to `http://<ha-proxy-ip>:8080/haproxy?stats` to view the status dashboard. This dashboard can be moved to run on port 80, and authentication can also be added. See the HAProxy documentation for details on this.

Optimizing for reads and writes

Neo4j provides a catalogue of *health check URLs* (see [Status information](#)) that HAProxy (or any load balancer for that matter) can use to distinguish machines using HTTP response codes. In the example above we used the `/available` endpoint, which directs requests to machines that are generally available for transaction processing (they are alive!).

However, it is possible to have requests directed to slaves only, or to the master only. If you are able to distinguish in your application between requests that write, and requests that only read, then you can take advantage of two (logical) load balancers: one that sends all your writes to the master, and one that sends all your read-only requests to a slave. In HAProxy you build logical load balancers by adding multiple backends.

The trade-off here is that while Neo4j allows slaves to proxy writes for you, this indirection unnecessarily ties up resources on the slave and adds latency to your write requests. Conversely, you don't particularly want read traffic to tie up resources on the master; Neo4j allows you to scale out for reads, but writes are still constrained to a single instance. If possible, that instance should exclusively do writes to ensure maximum write performance.

The following example excludes the master from the set of machines using the `/slave` endpoint.

```
global
  daemon
  maxconn 256

defaults
  mode http
  timeout connect 5000ms
  timeout client 50000ms
  timeout server 50000ms

frontend http-in
  bind *:80
  default_backend neo4j-slaves

backend neo4j-slaves
  option httpchk GET /db/manage/server/ha/slave
  server s1 10.0.1.10:7474 maxconn 32 check
  server s2 10.0.1.11:7474 maxconn 32 check
  server s3 10.0.1.12:7474 maxconn 32 check

listen admin
  bind *:8080
  stats enable
```

 In practice, writing to a slave is uncommon. While writing to slaves has the benefit of ensuring that data is persisted in two places (the slave and the master), it comes at a cost. The cost is that the slave must immediately become consistent with the master by applying any missing transactions and then synchronously apply the new transaction with the master. This is a more expensive operation than writing to the master and having the master push changes to one or more slaves.

Cache-based sharding with HAProxy

Neo4j HA enables what is called cache-based sharding. If the dataset is too big to fit into the cache of any single machine, then by applying a consistent routing algorithm to requests, the caches on each machine will actually cache different parts of the graph. A typical routing key could be user ID.

In this example, the user ID is a query parameter in the URL being requested. This will route the same user to the same machine for each request.

```

global
  daemon
  maxconn 256

defaults
  mode http
  timeout connect 5000ms
  timeout client 50000ms
  timeout server 50000ms

frontend http-in
  bind *:80
  default_backend neo4j-slaves

backend neo4j-slaves
  balance url_param user_id
  server s1 10.0.1.10:7474 maxconn 32
  server s2 10.0.1.11:7474 maxconn 32
  server s3 10.0.1.12:7474 maxconn 32

listen admin
  bind *:8080
  stats enable

```

Naturally the health check and query parameter-based routing can be combined to only route requests to slaves by user ID. Other load balancing algorithms are also available, such as routing by source IP ([source](#)), the URI ([uri](#)) or HTTP headers([hdr\(\)](#)).

C.3. Highly Available mode on Docker

This section describes how to run Neo4j in Highly Available mode, in a Docker container.

In order to run Neo4j in HA mode under Docker you need to wire up the containers in the cluster so that they can talk to each other. Each container must have a network route to each of the others and the [NE04J_ha_host_coordination](#), [NE04J_ha_host_data](#) and [NE04J_ha_initial_hosts](#) environment variables must be set accordingly:

- [NE04J_dbms_mode](#): the database mode, defaults to [SINGLE](#), set to [HA](#) for Highly Available clusters.
- [NE04J_ha_server__id](#): the id of the server, must be unique within a cluster
- [NE04J_ha_host_coordination](#): the address (including port) used for cluster coordination in HA mode, this must be resolvable by all cluster members
- [NE04J_ha_host_data](#): the address (including port) used for data transfer in HA mode, this must be resolvable by all cluster members
- [NE04J_ha_initial__hosts](#): comma-separated list of other members of the cluster

Within a single Docker host, this can be achieved as follows:

```

docker network create --driver=bridge cluster

docker run --name=instance1 --detach --publish=7474:7474 --publish=7687:7687 --net=cluster --hostname=instance1 \
    --volume=$HOME/neo4j/logs1:/logs \
    --env=NEO4J_dbms_mode=HA --env=NEO4J_ha_server_id=1 \
    --env=NEO4J_ha_host_coordination=instance1:5001 --env=NEO4J_ha_host_data=instance1:6001 \
    --env=NEO4J_ha_initial_hosts=instance1:5001,instance2:5001,instance3:5001 \
    --env=NEO4J_ACCEPT_LICENSE AGREEMENT=yes \
    neo4j:3.4-enterprise

docker run --name=instance2 --detach --publish=7475:7474 --publish=7688:7687 --net=cluster --hostname=instance2 \
    --volume=$HOME/neo4j/logs2:/logs \
    --env=NEO4J_dbms_mode=HA --env=NEO4J_ha_server_id=2 \
    --env=NEO4J_ha_host_coordination=instance2:5001 --env=NEO4J_ha_host_data=instance2:6001 \
    --env=NEO4J_ha_initial_hosts=instance1:5001,instance2:5001,instance3:5001 \
    --env=NEO4J_ACCEPT_LICENSE AGREEMENT=yes \
    neo4j:3.4-enterprise

docker run --name=instance3 --detach --publish=7476:7474 --publish=7689:7687 --net=cluster --hostname=instance3 \
    --volume=$HOME/neo4j/logs3:/logs \
    --env=NEO4J_dbms_mode=HA --env=NEO4J_ha_server_id=3 \
    --env=NEO4J_ha_host_coordination=instance3:5001 --env=NEO4J_ha_host_data=instance3:6001 \
    --env=NEO4J_ha_initial_hosts=instance1:5001,instance2:5001,instance3:5001 \
    --env=NEO4J_ACCEPT_LICENSE AGREEMENT=yes \
    neo4j:3.4-enterprise

```

See the [Set up an HA cluster](#) for more details of Neo4j Highly Available mode.

C.4. Status information

This section describes the available endpoints for Neo4j HA clusters.

C.4.1. Introduction

A common use case for Neo4j HA clusters is to direct all write requests to the master while using slaves for read operations, distributing the read load across the cluster and gaining failover capabilities for the deployment. The most common way to achieve this is to place a load balancer in front of the HA cluster. For a description of HAProxy, see [HAProxy for load balancing](#). As you can see in that section, it makes use of an HTTP endpoint to discover which instance is the master, and then directs write load to it.

C.4.2. The endpoints

Each HA instance comes with three endpoints regarding its status:

- */db/manage/server/ha/master*
- */db/manage/server/ha/slave*
- */db/manage/server/ha/available*

The */master* and */slave* endpoints can be used to direct write and non-write traffic to specific instances. This is the optimal way to take advantage of Neo4j's scaling characteristics. The */available* endpoint exists for the general case of directing arbitrary request types to instances that are available for transaction processing.

To use the endpoints, perform an HTTP `GET` operation. The table below outlines the possible responses for each endpoint:

Table 352. HA HTTP endpoint responses

Endpoint	Instance State	Returned Code	Body text
/db/manage/server/ha/master	Master	200 OK	true
	Slave	404 Not Found	false
	Unknown	404 Not Found	UNKNOWN
/db/manage/server/ha/slave	Master	404 Not Found	false
	Slave	200 OK	true
	Unknown	404 Not Found	UNKNOWN
/db/manage/server/ha/available	Master	200 OK	master
	Slave	200 OK	slave
	Unknown	404 Not Found	UNKNOWN

C.4.3. Examples

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 *master* endpoint on a running master with verbose output

```
#> curl -v localhost:7474/db/manage/server/ha/master
* About to connect() to localhost port 7474 (#0)
*   Trying ::1...
* connected
* Connected to localhost (::1) port 7474 (#0)
> GET /db/manage/server/ha/master 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(6.1.25)
<
* Connection #0 to host localhost left intact
true* Closing connection #0
```

- Requesting *slave* endpoint on a running master without verbose output:

```
#> curl localhost:7474/db/manage/server/ha/slave
false
```

- Finally, requesting the *master* endpoint on a slave with verbose output

```
#> curl -v localhost:7475/db/manage/server/ha/master
* About to connect() to localhost port 7475 (#0)
* Trying ::1...
* connected
* Connected to localhost (::1) port 7475 (#0)
> GET /db/manage/server/ha/master 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:7475
> Accept: */*
>
< HTTP/1.1 404 Not Found
< Content-Type: text/plain
< Access-Control-Allow-Origin: *
< Transfer-Encoding: chunked
< Server: Jetty(6.1.25)
<
* Connection #0 to host localhost left intact
false* Closing connection #0
```

Unknown status



The **UNKNOWN** status exists to describe when a Neo4j instance is neither master nor slave. For example, the instance could be transitioning between states (master to slave in a recovery scenario or slave being promoted to master in the event of failure), or the instance could be an arbiter instance. If the **UNKNOWN** status is returned, the client should not treat the instance as a master or a slave and should instead pick another instance in the cluster to use, wait for the instance to transit from the **UNKNOWN** state, or undertake restorative action via systems admin.

If [authentication and authorization](#) is enabled, the HA status endpoints will also require authentication credentials. For some load balancers and proxy servers, providing this with the request is not an option. For those situations, consider disabling authentication of the HA status endpoints by setting `dbms.security.ha_status_auth_enabled=false` in the `neo4j.conf` configuration file.

C.5. Upgrade

This section describes how to upgrade a Neo4j HA cluster.

Upgrading a Neo4j HA cluster requires following a specific process in order to ensure that the cluster remains consistent, and that all cluster instances are able to join and participate in the cluster following their upgrade.

Pre-upgrade steps

- Read [Upgrade planning](#) thoroughly and perform all the steps listed there.
- Verify that you have a [full backup](#) that is stored in a safe location.

Downtime

Neo4j 3.5.0 does not support rolling upgrades for HA clusters, so the upgrade process can involve significant downtime for the entire cluster. A test upgrade on a production-like equipment provides information on the duration of the downtime.

Rollback

Neo4j does not support downgrades. If the upgrade is not successful, you have to do a full rollback, including restoring a pre-upgrade backup.

Upgrade steps

1. Shut down the cluster
 - a. Shut down the slave instances, one by one.

- b. Shut down the master last.
2. Upgrade the master
 - a. Install Neo4j 3.5.0 on the master, keeping the database directory (in a default configuration: `data/databases/graph.db`) untouched.
 - b. Set `dbms.mode=SINGLE` in `neo4j.conf` to disable HA in the configuration.
 - c. Follow the [single instance upgrade](#) instructions to upgrade Neo4j.
 - d. When upgrade has completed, shut down Neo4j again.
 - e. Set `dbms.mode=HA` in `neo4j.conf` to re-enable HA in the configuration.
 - f. Make a full backup of the Neo4j database.

Please note that backups taken before the upgrade are no longer valid for update via the incremental online backup. Therefore, it is important to perform a *full backup*, using an empty target directory, at this point.

3. Upgrade the slaves

On each slave:

- a. Remove the database directory (in a default configuration: `data/databases/graph.db`).
- b. Install Neo4j 3.5.0.
- c. Review the settings in the configuration files in the previous installation, and transfer any custom settings to the 3.5.0 installation.
- d. If applicable, copy the security configuration from the master, since this is not propagated automatically.



Alternatively, at this point you can manually copy the database directory from the master to the slaves. Doing so will avoid the need to sync from the master when starting. This can save considerable time when upgrading large databases.

4. Restart the cluster

- a. Start the master instance.
- b. Start the slaves, one by one.

Once a slave has joined the cluster, it will sync the database from the master instance.

C.6. Backup and restore

The steps required to backup and restore a Neo4j HA cluster.

Review [Backup planning](#) for a discussion about backup strategies.

The applicable configuration options for backups in a Neo4j HA cluster are the same as for a standalone database. See [Configuration parameters](#).

To back up an HA cluster, follow the instructions in [Perform a backup](#).

To restore from backup in an HA cluster environment, follow these steps:

1. Shut down all database instances in the cluster.
2. Restore the backup on each instance, using `neo4j-admin restore`.
3. If you are restoring onto new hardware, please review the HA settings in `neo4j.conf`.

In particular, check the `ha.initial_hosts` setting to ensure that the servers listed reflect the servers on which the restores are being made.

4. Start the database instances, beginning with the master.

C.7. Metrics

This section lists metrics specific to HA clusters.

For information on monitoring, see [Monitoring](#).

The following metrics are available:

Table 353. Network metrics

Name	Description
<code>neo4j.network.slave_network_tx_writes</code>	The amount of bytes transmitted on the network containing the transaction data from a slave to the master in order to be committed
<code>neo4j.network.master_network_store_writes</code>	The amount of bytes transmitted on the network while copying stores from a machines to another
<code>neo4j.network.master_network_tx_writes</code>	The amount of bytes transmitted on the network containing the transaction data from a master to the slaves in order to propagate committed transactions

Table 354. Cluster metrics

Name	Description
<code>neo4j.cluster.slave_pull_updates</code>	The total number of update pulls executed by this instance
<code>neo4j.cluster.slave_pull_update_up_to_tx</code>	The highest transaction id that has been pulled in the last pull updates by this instance
<code>neo4j.cluster.is_master</code>	Whether or not this instance is the master in the cluster
<code>neo4j.cluster.is_available</code>	Whether or not this instance is available in the cluster

C.8. Tutorials

This section includes tutorials on how to configure an HA cluster.

The following tutorials are available:

- [Set up an HA cluster](#)
- [Set up a local HA cluster](#)

C.8.1. Set up an HA cluster

This tutorial provides step-by-step instructions for setting up a basic cluster of Neo4j running on three separate machines.

Download and configure

1. Download Neo4j Enterprise Edition from [the Neo4j download site](https://neo4j.com/download/other-releases/#releases) (<https://neo4j.com/download/other-releases/#releases>), and unpack on three separate machines.
2. Configure the HA related settings for each installation as outlined below. Note that all three installations have the same configuration except for the `ha.server_id` property.

Example 119. Configuration of neo4j.conf for each of the three HA servers

Neo4j instance #1 on the server named neo4j-01.local

conf/neo4j.conf

```
# Unique server id for this Neo4j instance
# can not be negative id and must be unique
ha.server_id=1

# List of other known instances in this cluster
ha.initial_hosts=neo4j-01.local:5001,neo4j-02.local:5001,neo4j-03.local:5001
# Alternatively, use IP addresses:
#ha.initial_hosts=192.168.0.20:5001,192.168.0.21:5001,192.168.0.22:5001

# HA - High Availability
# SINGLE - Single mode, default.
dbms.mode=HA

# HTTP Connector
dbms.connector.http.enabled=true
dbms.connector.http.listen_address=:7474
```

Neo4j instance #2 on the server named neo4j-02.local

conf/neo4j.conf

```
# Unique server id for this Neo4j instance
# can not be negative id and must be unique
ha.server_id=2

# List of other known instances in this cluster
ha.initial_hosts=neo4j-01.local:5001,neo4j-02.local:5001,neo4j-03.local:5001
# Alternatively, use IP addresses:
#ha.initial_hosts=192.168.0.20:5001,192.168.0.21:5001,192.168.0.22:5001

# HA - High Availability
# SINGLE - Single mode, default.
dbms.mode=HA

# HTTP Connector
dbms.connector.http.enabled=true
dbms.connector.http.listen_address=:7474
```

Neo4j instance #3 on the server named neo4j-03.local

conf/neo4j.conf

```
# Unique server id for this Neo4j instance
# can not be negative id and must be unique
ha.server_id=3

# List of other known instances in this cluster
ha.initial_hosts=neo4j-01.local:5001,neo4j-02.local:5001,neo4j-03.local:5001
# Alternatively, use IP addresses:
#ha.initial_hosts=192.168.0.20:5001,192.168.0.21:5001,192.168.0.22:5001

# HA - High Availability
# SINGLE - Single mode, default.
dbms.mode=HA

# HTTP Connector
dbms.connector.http.enabled=true
dbms.connector.http.listen_address=:7474
```

Start the Neo4j Servers

Start the Neo4j servers as usual. Note that the startup order does not matter.

Example 120. Start the three HA servers

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

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

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

Startup Time



When running in HA mode, the startup script returns immediately instead of waiting for the server to become available. The database will be unavailable until all members listed in `ha.initial_hosts` are online and communicating with each other. In the example above this happens when you have started all three instances. To keep track of the startup state you can follow the messages in `neo4j.log` — the path is printed before the startup script returns.

Now, you should be able to access the three servers and check their HA status. Open the locations below in a web browser and issue the following command in the editor after having set a password for the database: `:play sysinfo`

- `http://neo4j-01.local:7474/`
- `http://neo4j-02.local:7474/`
- `http://neo4j-03.local:7474/`



You can replace database #3 with an 'arbiter' instance, see [Arbiter instances](#).

That is it! You now have a Neo4j HA cluster of three instances running. You can start by making a change on any instance and those changes will be propagated between them. For more cluster related configuration options take a look at [Setup and configuration](#).

C.8.2. Set up a local HA cluster

This tutorial walks through the basics of setting up a Neo4j HA cluster on one machine. The result is a local cluster of three instances: one master and two slaves.

If you want to start a cluster similar to the one described above, but for development and testing purposes, it is convenient to run all Neo4j instances on the same machine. This is easy to achieve, although it requires some additional configuration as the defaults will conflict with each other. Furthermore, the default `dbms.memory.pagecache.size` assumes that Neo4j has the machine to itself. If we in this example assume that the machine has 4 gigabytes of memory, and that each JVM consumes 500 megabytes of memory, then we can allocate 500 megabytes of memory to the page cache of each server.

Download and configure

1. Download Neo4j Enterprise Edition from [the Neo4j download site](#) (<https://neo4j.com/download/other-releases/#releases>), and unpack into three separate directories on your test machine.
2. Configure the HA related settings for each installation as outlined below.

Neo4j instance #1 — ~/neo4j-01

conf/neo4j.conf

```
# Reduce the default page cache memory allocation
dbms.memory.pagecache.size=500m

# Port to listen to for incoming backup requests.
dbms.backup.address = 127.0.0.1:6366

# Unique server id for this Neo4j instance
# can not be negative id and must be unique
ha.server_id = 1

# List of other known instances in this cluster
ha.initial_hosts = 127.0.0.1:5001,127.0.0.1:5002,127.0.0.1:5003

# IP and port for this instance to bind to for communicating cluster information
# with the other neo4j instances in the cluster.
ha.host.coordination = 127.0.0.1:5001

# IP and port for this instance to bind to for communicating data with the
# other neo4j instances in the cluster.
ha.host.data = 127.0.0.1:6363

# HA - High Availability
# SINGLE - Single mode, default.
dbms.mode=HA

dbms.connector.http.type=HTTP
dbms.connector.http.enabled=true
dbms.connector.http.address=0.0.0.0:7474
```

Neo4j instance #2 — ~/neo4j-02

conf/neo4j.conf

```
# Reduce the default page cache memory allocation
dbms.memory.pagecache.size=500m

# Port to listen to for incoming backup requests.
dbms.backup.address = 127.0.0.1:6367

# Unique server id for this Neo4j instance
# can not be negative id and must be unique
ha.server_id = 2

# List of other known instances in this cluster
ha.initial_hosts = 127.0.0.1:5001,127.0.0.1:5002,127.0.0.1:5003

# IP and port for this instance to bind to for communicating cluster information
# with the other neo4j instances in the cluster.
ha.host.coordination = 127.0.0.1:5002

# IP and port for this instance to bind to for communicating data with the
# other neo4j instances in the cluster.
ha.host.data = 127.0.0.1:6364

# HA - High Availability
# SINGLE - Single mode, default.
dbms.mode=HA

dbms.connector.http.type=HTTP
dbms.connector.http.enabled=true
dbms.connector.http.address=0.0.0.0:7475
```

Neo4j instance #3 — ~/neo4j-03

conf/neo4j.conf

```
# Reduce the default page cache memory allocation
dbms.memory.pagecache.size=500m

# Port to listen to for incoming backup requests.
dbms.backup.address = 127.0.0.1:6368

# Unique server id for this Neo4j instance
# can not be negative id and must be unique
ha.server_id = 3

# List of other known instances in this cluster
ha.initial_hosts = 127.0.0.1:5001,127.0.0.1:5002,127.0.0.1:5003

# IP and port for this instance to bind to for communicating cluster information
# with the other neo4j instances in the cluster.
ha.host.coordination = 127.0.0.1:5003

# IP and port for this instance to bind to for communicating data with the
# other neo4j instances in the cluster.
ha.host.data = 127.0.0.1:6365

# HA - High Availability
# SINGLE - Single mode, default.
dbms.mode=HA

dbms.connector.http.type=HTTP
dbms.connector.http.enabled=true
dbms.connector.http.address=0.0.0.0:7476
```

Start the Neo4j Servers

Start the Neo4j servers as usual. Note that the startup order does not matter.

```
localhost:~/neo4j-01$ ./bin/neo4j start
```

```
localhost:~/neo4j-02$ ./bin/neo4j start
```

```
localhost:~/neo4j-03$ ./bin/neo4j start
```

Now, you should be able to access the three servers and check their HA status. Open the locations below in a web browser and issue the following command in the editor after having set a password for the database: `:play sysinfo`

- <http://127.0.0.1:7474/>
- <http://127.0.0.1:7475/>
- <http://127.0.0.1:7476/>

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>.