The Neo4j Operations Manual v4.2

Table of Contents

This manual covers the following areas:

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

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

Who should read this?

This manual is written for:

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

Chapter 1. Introduction

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

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

1.1. Neo4j editions

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

Community Edition

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

Enterprise Edition

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

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

Table 1. Community Edition vs Enterprise Edition key features

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

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

1.1.1. Performance and scalability

Table 2. Performance and scalability features

Feature	Community Edition	Enterprise Edition
Causal Clustering for global scale applications	✘	✔
Enterprise lock manager accesses all cores on server	✘	✔
Intra-cluster encryption	✘	✔
Offline backups	✔	✔
Online backups	✘	✔
Encrypted backups	✘	✔
Rolling upgrades	✘	✔
Automatic cache warming	✘	✔
Routing and load balancing with Neo4j Drivers	✘	✔
Advanced monitoring	✘	✔

Feature	Community Edition	Enterprise Edition
Graph size limitations	34 billion nodes, 34 billion relationships, and 68 billion properties	No limit
Bulk import tool	✔	✔
Bulk import tool, resumable	✖	✔

1.2. Versioning

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

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

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

[1] Must be downloaded and installed separately.

Chapter 2. Installation

The topics described are:

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

Installation-free options

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

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

2.1. System requirements

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

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

2.1.1. Supported platforms

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

2.1.2. Hardware requirements

In terms of minimum hardware requirements, follow these guidelines:

Table 3. Hardware requirement guidelines.

CPU	Performance is generally memory or I/O bound for large graphs, and compute bound for graphs that fit in memory.
Memory	More memory allows for larger graphs, but it needs to be configured properly to avoid disruptive garbage collection operations.

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

For personal use and software development:

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

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

For cloud environments:

Table 5. Hardware requirement guidelines for cloud environments.

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

For server-based, on-premise environments:

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

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

2.1.3. Software requirements

For personal use and software development:

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

Operating System	Supported JDK
MacOS 10.14+	ZuluJDK 11
Ubuntu Desktop 16.04+	OpenJDK 11, OracleJDK 11, and ZuluJDK 11
Debian 10+	OpenJDK 11, OracleJDK 11, and ZuluJDK 11
SuSE 15+	Oracle JDK 11
Windows 10	OracleJDK 11 and ZuluJDK 11

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

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

Operating System	Supported JDK
Ubuntu Server 16.04+	OpenJDK 11, OracleJDK 11, and ZuluJDK 11
Red Hat Enterprise Linux Server 7.9+	Red Hat OpenJDK 11, Oracle JDK 11, and ZuluJDK 11
CentOS Server 7	OpenJDK 11
Amazon Linux AMI 2018.03+	Amazon Corretto 11, OpenJDK 11, and OracleJDK 11
Windows Server 2016+	OracleJDK 11 and ZuluJDK 11

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

2.1.4. Filesystem

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

2.1.5. Java

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

Table 9. Neo4j version and JVM requirements.

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

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

2.2. Neo4j Browser

Neo4j Browser is a tool for developers to interact with the graph. It is the default interface for both Enterprise and Community Editions of the Neo4j database.

Neo4j Browser is bundled with Neo4j database, including both Neo4j Server and Neo4j Desktop (see Neo4j Desktop for more information).

The following web browsers are supported:

- Chrome (Latest version)
- Firefox (Latest version)
- Edge (Latest version)

Internet Explorer web browser is *not* supported.

2.3. Neo4j Desktop

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

To install Neo4j Desktop, go to Neo4j Download Center and follow the instructions.

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

Neo4j Desktop is not suited for production environments.

2.4. Linux installation

This section describes the following:

- Install Neo4j on Debian and Debian-based distributions
  - Installation
  - File locations
  - Operation
- Deploy Neo4j using the Neo4j RPM package
  - Install on Red Hat, CentOS, Fedora or Amazon Linux
    - Standard installation
    - Non-interactive installation of Neo4j Enterprise Edition
  - Install on SUSE
  - Offline installation
- Install Neo4j on Linux from a tarball

2.4.1. Debian-based distributions (.deb)

Installation

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

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

Java Prerequisites (Oracle Java and Ubuntu 16.04+ only)

Neo4j 4.2 requires the Java 11 runtime. Java 11 is not included in Ubuntu 16.04 LTS and will have to be set up manually prior to installing or upgrading to Neo4j 4.2, as described below. Ubuntu 18.04 onwards already has the Openjdk Java 11 package available through `apt`.

Oracle Java and Debian

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

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

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

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

Java 11 on Ubuntu 16.04

Add the official OpenJDK package repository to `apt`:

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

You are now ready to install Neo4j, which will install Java 11 automatically if it is not already installed. See Dealing with multiple installed Java versions to make sure you can start Neo4j after install.

Dealing with multiple installed Java versions

It is important that you configure your default Java version to point to Java 11, or Neo4j 4.2.19 will be unable to start. Do so with the `update-java-alternatives` command.

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

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

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

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

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

Add the repository

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

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

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

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

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

  ```
  wget -O - https://debian.neo4j.com/neotechnology.gpg.key | sudo apt-key add -
  echo 'deb https://debian.neo4j.com stable 4.2' | sudo tee -a /etc/apt/sources.list.d/neo4j.list
  sudo apt-get update
  ```

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

  ```
  apt list -a neo4j
  ```

In Ubuntu server installations you will also need to make sure that the `universe` repository is enabled. If the `universe` repository is not present, the Neo4j installation will fail with the error `Depends: daemon but it is not installable`.

This can be fixed by running the command:

```
sudo add-apt-repository universe
```

Install Neo4j

To install Neo4j Community Edition:

```
sudo apt-get install neo4j=1:4.2.19
```

To install Neo4j Enterprise Edition:

```
sudo apt-get install neo4j-enterprise=1:4.2.19
```

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

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

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

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

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

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

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

Non-interactive installation of Neo4j Enterprise Edition

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

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

Offline installation

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

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

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

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

     ```
     curl -O https://dist.neo4j.org/deb/neo4j-enterprise_4.2.19_all.deb
     ```

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

     ```
     dpkg --contents neo4j_4.2.19_all.deb
     ```

   - Neo4j Community Edition:

     ```
     curl -O https://dist.neo4j.org/deb/neo4j_4.2.19_all.deb
     ```

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

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

File locations

File locations for all Neo4j packages are documented here.

Operation

Most Neo4j configuration goes into *neo4j.conf*.

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

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

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

Starting the service automatically on system start

On Debian-based distributions, Neo4j is enabled to start automatically on system boot by default.

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

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

For more information, see Set an initial password.

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

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

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

Standard installation

1. Add the repository.

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

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

2. Ensure the correct Java version.

Neo4j 4.2 requires the Java 11 runtime. Most of our supported RPM Linux distributions have Java 11 available by default. There is some minor setup required for Amazon Linux, and for compatibility with Oracle Java 11:

- Java 11 on Amazon Linux:

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

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

  You are now ready to install Neo4j 4.2.19, which will install Java 11 automatically if it is not already installed.

- Oracle Java 11:

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

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

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

  b. Install the adapter:

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

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

You are now ready to install Neo4j 4.2.19.

3. Install Neo4j.

   - To install Neo4j Community Edition as `root`:

   ```
   yum install neo4j-4.2.19
   ```

   - To install Neo4j Enterprise Edition as `root`:

   ```
   yum install neo4j-enterprise-4.2.19
   ```

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

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

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

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

Non-interactive installation of Neo4j Enterprise Edition

When installing Neo4j Enterprise Edition, you will be required to accept the license agreement before installation is allowed to complete. This is an interactive prompt. If you require non-interactive installation of Neo4j Enterprise Edition, you can indicate that you have read and accepted the license agreement by setting the environment variable `NEO4J_ACCEPT_LICENSE_AGREEMENT` to `yes`:

```
NEO4J_ACCEPT_LICENSE_AGREEMENT=yes yum install neo4j-enterprise-4.2.19
```

Install on SUSE

For SUSE-based distributions the steps are as follows:

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

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

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

```
zypper install neo4j-4.2.19
```

   - To install Neo4j Enterprise Edition as `root`:

```
zypper install neo4j-enterprise-4.2.19
```

Offline installation

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

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

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

Downloading the RPM installers

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

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

    ```
    curl -O https://dist.neo4j.org/rpm/neo4j-enterprise-4.2.19-1.noarch.rpm
    ```

    - Neo4j Community Edition:

    ```
    curl -O https://dist.neo4j.org/rpm/neo4j-4.2.19-1.noarch.rpm
    ```

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

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

- To install Neo4j Enterprise Edition as `root`:

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

Performing an offline installation

Offline upgrade from 4.0.0 or later

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

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

Offline upgrade from 3.5 or earlier

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

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

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

Starting the service automatically on system start

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

```
systemctl enable neo4j
```

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

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

For more information, see Set an initial password.

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

2.4.3. Linux executable (.tar)

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

Install Neo4j from a tarball

1. If it is not already installed, get OpenJDK 11 or Oracle Java 11.
2. Download the latest Neo4j tarball from Neo4j Download Center and unpack it:

   ```
   tar zxf neo4j-enterprise-4.2.19-unix.tar.gz
   ```

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

   ```
   mv neo4j-enterprise-4.2.19 /opt/
   ln -s /opt/neo4j-enterprise-4.2.19 /opt/neo4j
   ```

4. Create a `neo4j` user and group:

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

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

     ```
     chown -R neo4j:adm /opt/neo4j-enterprise-4.2.19
     ```

- RedHat

```
chown -R neo4j /opt/neo4j-enterprise-4.2.19
```

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

Configure Neo4j to start automatically on system boot

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

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

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

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

[Install]
WantedBy=multi-user.target
//Reload systemctl to pick up the new service file
systemctl daemon-reload
```

2. Configure Neo4j to start at boot time:

```
systemctl enable neo4j
```

3. Start Neo4j:

```
systemctl start neo4j
```

4. Check the status of the newly created service:

```
systemctl status neo4j
```

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

```
reboot
```

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

Setting the number of open files

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

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

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

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

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

Running Neo4j as a service

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

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

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

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

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

1. Open the *user.conf* file with root privileges in a text editor, for example, Vim.

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

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

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

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

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

4. Define the following values:

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

5. Reload the `systemd` settings.

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

6. Reboot your machine.

2.4.4. Neo4j system service

Setting the number of open files.

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

Configuration

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

Controlling the service

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

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

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

```
systemctl edit neo4j
```

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

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

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

```
systemctl cat neo4j
```

Remember to restart neo4j if you change any settings.

```
systemctl restart neo4j
```

Log

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

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

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

2.5. macOS installation

2.5.1. Unix console application

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

   Select the appropriate tar.gz distribution for your platform.
3. Make sure to download Neo4j from Neo4j Download Center and always 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.
4. Extract the contents of the archive, using `tar -xf <filename>`

   Refer to the top-level extracted directory as: NEO4J_HOME
5. Change directory to: `$NEO4J_HOME`

   Run `./bin/neo4j console`
6. Stop the server by typing `Ctrl-C` in the console.

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

2.5.2. macOS service

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

2.5.3. macOS file descriptor limits

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

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

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

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

2.6. Windows installation

2.6.1. Windows console application

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

   Select the appropriate ZIP distribution.
3. Make sure to download Neo4j from Neo4j Download Center and always 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.
4. Right-click the downloaded file, click Extract All.
5. Change directory to the top-level extracted directory.

   Run `bin\neo4j console`
6. Stop the server by typing `Ctrl-C` in the console.

2.6.2. 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`.

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

Java options

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

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

Example 1. Update service example

1. Install service

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

2. Change memory configuration

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

3. Update service

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

4. Restart service

```
bin\neo4j restart
```

2.6.3. Windows PowerShell module

The Neo4j PowerShell module allows administrators to:

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

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

System requirements

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

Managing Neo4j on Windows

On Windows, it is sometimes necessary to *Unblock* a downloaded ZIP file before you can import its contents as a module. If you right-click on the ZIP file and choose "Properties" you will get a dialog which includes an "Unblock" button, which will enable you to import the module.

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

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

For more information, see About execution policies.

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

How do I import the module?

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

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

This will add the module to the current session.

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

```
Invoke-Neo4j console
```

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

How do I get help about the module?

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

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

The output should be similar to the following:

```
CommandType     Name                                Version    Source
-----------     ----                                -------    ------
Function        Invoke-Neo4j                        4.2.19      Neo4j-Management
Function        Invoke-Neo4jAdmin                   4.2.19      Neo4j-Management
Function        Invoke-Neo4jBackup                  4.2.19      Neo4j-Management
Function        Invoke-Neo4jImport                  4.2.19      Neo4j-Management
Function        Invoke-Neo4jShell                   4.2.19      Neo4j-Management
```

The module also supports the standard PowerShell help commands.

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

Run the following to see examples of help commands:

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

Example usage

- List of available commands:

  ```
  Invoke-Neo4j
  ```

- Current status of the Neo4j service:

  ```
  Invoke-Neo4j status
  ```

- Install the service with verbose output:

  ```
  Invoke-Neo4j install-service -Verbose
  ```

- Available commands for administrative tasks:

  ```
  Invoke-Neo4jAdmin
  ```

Common PowerShell parameters

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

Chapter 3. Cloud deployments

The topics covered are:

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

Other cloud deployment options

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

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

3.1. Neo4j cloud VMs

3.1.1. Basics and file Locations

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

3.1.2. VM configuration

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

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

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

3.1.3. Configuration via VM tags

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

tags, the configuration is not immediately applied to the Neo4j system service running inside of the VM. To affect these changes, please restart the system service.

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

3.1.4. Interacting with the Neo4j Service

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

```
systemctl status neo4j
```

3.2. Neo4j on Amazon EC2

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

To automate Neo4j deployment on AWS, see Neo4j deployment automation on AWS.

The following links provide more information for running Neo4j on AWS:

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

3.2.1. Neo4j deployment automation on AWS

Prerequisites

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

CloudFormation

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

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

The Neo4j CloudFormation templates have the following properties:

- Deploying one or more EC2 VMs in a specified region.

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

Creating a CloudFormation stack

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

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

Deploying Neo4j Enterprise Standalone

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

```
#!/bin/bash
VERSION=4.2.19
export SINGLE_TEMPLATE=http://neo4j-cloudformation.s3.amazonaws.com/neo4j-enterprise-standalone-stack-
$VERSION.json
export STACKNAME=neo4j-enterprise-$(echo $VERSION | sed s/[^A-Za-z0-9]/-/g)
export INSTANCE=r4.large
export REGION=us-east-1
export SSHKEY=my-ssh-keyname
aws cloudformation create-stack \
   --stack-name $STACKNAME \
   --region $REGION \
   --template-url $SINGLE_TEMPLATE \
   --parameters ParameterKey=InstanceType,ParameterValue=$INSTANCE \
     ParameterKey=NetworkWhitelist,ParameterValue=0.0.0.0/0 \
     ParameterKey=Password,ParameterValue=s00pers3cret \
     ParameterKey=SSHKeyName,ParameterValue=$SSHKEY \
     ParameterKey=VolumeSizeGB,ParameterValue=37 \
     ParameterKey=VolumeType,ParameterValue=gp2 \
     --capabilities CAPABILITY_NAMED_IAM
```

Deploying Neo4j Enterprise Causal Cluster

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

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

```
#!/bin/bash
VERSION=4.2.19
export CLUSTER_TEMPLATE=http://neo4j-cloudformation.s3.amazonaws.com/neo4j-enterprise-stack-$VERSION.json
export STACKNAME=neo4j-enterprise-$(echo $VERSION | sed s/[^A-Za-z0-9]/-/g)
export INSTANCE=r4.large
export REGION=us-east-1
export SSHKEY=my-ssh-keyname
aws cloudformation create-stack \
   --stack-name $STACKNAME \
   --region $REGION \
   --template-url $CLUSTER_TEMPLATE \
   --parameters ParameterKey=ClusterNodes,ParameterValue=3 \
     ParameterKey=InstanceType,ParameterValue=$INSTANCE \
     ParameterKey=NetworkWhitelist,ParameterValue=0.0.0.0/0 \
     ParameterKey=Password,ParameterValue=s00pers3cret \
     ParameterKey=SSHKeyName,ParameterValue=$SSHKEY \
     ParameterKey=VolumeSizeGB,ParameterValue=37 \
     ParameterKey=VolumeType,ParameterValue=gp2 \
     --capabilities CAPABILITY_NAMED_IAM
```

Deploying Neo4j Community Standalone

To deploy Neo4j Community Standalone, use the Community template.

```
#!/bin/bash
VERSION=4.2.19
export COMMUNITY_TEMPLATE=http://neo4j-cloudformation.s3.amazonaws.com/neo4j-community-standalone-stack-
$VERSION.json
export STACKNAME=neo4j-comm-$(echo $VERSION | sed s/[^A-Za-z0-9]/-/g)
export INSTANCE=r4.large
export REGION=us-east-1
export SSHKEY=my-ssh-keyname
aws cloudformation create-stack \
   --stack-name $STACKNAME \
   --region $REGION \
   --template-url $COMMUNITY_TEMPLATE \
   --parameters ParameterKey=InstanceType,ParameterValue=$INSTANCE \
     ParameterKey=NetworkWhitelist,ParameterValue=0.0.0.0/0 \
     ParameterKey=Password,ParameterValue=s00pers3cret \
     ParameterKey=SSHKeyName,ParameterValue=$SSHKEY \
     ParameterKey=VolumeSizeGB,ParameterValue=37 \
     ParameterKey=VolumeType,ParameterValue=gp2 \
     --capabilities CAPABILITY_NAMED_IAM
```

Checking to see if your instance is up

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

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

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

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

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

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

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

Cleaning up and removing your stack

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

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

3.3. Neo4j on Google Cloud Platform

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

- Single instances (VM-based) — Launching a single instance from an image.
- Causal Clusters (VM-Based) — Deploying Neo4j on GCP.
- Neo4j deployments automation on GCP – Automating Neo4j deployments on GCP.

3.3.1. Single instances (VM-based)

Prerequisites

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

Create a firewall rule to access your instance

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

```
gcloud compute firewall-rules create allow-neo4j-bolt-http-https \ ①
  --allow tcp:7473,tcp:7474,tcp:7687 \ ②
  --source-ranges 0.0.0.0/0 \ ③
  --target-tags neo4j ④
```

① Create a firewall rule with the name `allow-neo4j-bolt-http-https`.

② Allow traffic on ports:

- `7473` (HTTPS, for Neo4j Browser and HTTP API).
- `7474` (HTTP, for Neo4j Browser and HTTP API).
- `7687` (Bolt Protocol).

③ The ranges, provided with the `--source-ranges` argument, allow the entire Internet to contact your new instance.

④ The `--target-tags` argument specifies that this rule applies only to VMs tagged with `neo4j`.
When you launch your instance, you have to apply that tag to it.

Create a Google compute instance from the Neo4j public image

1. List all available Neo4j public images.

   The images are published in a GCP project called `launcher-public`, so by listing images in that project, you can see what is available.

   `launcher-public` *images*

   ```
   gcloud compute images list --project launcher-public
   ```

   `launcher-public` *images — filtered on Neo4j 4.X versions*

   ```
   gcloud compute images list --project launcher-public | grep --extended-regexp "neo4j-
   (community|enterprise)-1-4-.*"
   ```

   For example, the image `neo4j-enterprise-1-4-2-2-apoc` includes Neo4j Enterprise 4.2.2 with the APOC plugin.

2. Create a new instance.

   You create and launch an instance by using the following `gcloud` commands:

   ```
   gcloud config set project <project-id> ①
   gcloud compute instances create my-neo4j-instance --image-project launcher-public \ ②
     --image <neo4j-image-name> \ ③
     --tags neo4j ④
   ```

   ① Set your project configuration to ensure you know where you are launching your instance.

   ② Launch an image found in the provided public project `launcher-public`.

   ③ Replace `<neo4j-image-name>` with the image name you want to launch.

   ④ The `--tags` argument allows you to configure the correct network permissions.
   By default, Google blocks all external access to the network services unless you open them.

3. Note the `EXTERNAL_IP`.

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

   Example output

   ```
   Created [https://www.googleapis.com/compute/v1/projects/testbed-187316/zones/us-east1-b/instances/my-
   neo4j-instance].
   NAME              ZONE             MACHINE_TYPE   PREEMPTIBLE  INTERNAL_IP  EXTERNAL_IP     STATUS
   my-neo4j-instance  europe-north1-a  n1-standard-1               192.0.2.0    203.0.113.0     RUNNING
   ```

   Note the IP address[2] in the `EXTERNAL_IP` column, this is for the Neo4j server.

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

Access your new instance

Navigate to `http://[EXTERNAL_IP]:7474/browser` or `https://[EXTERNAL_IP]:7473/browser`, log in with the default username `neo4j` and password `neo4j`, and change the password, when prompted.

Neo4j 3.X versions include a self-signed certificate for TLS. Because you do not have a hostname or a valid SSL certificate configured by default, your browser will warn you that the certificate is not trusted.

Neo4j 4.X versions do not include any certificate for TLS. You can configure the certificate later.

Access your instance via SSH

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

ssh

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

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

systemctl

```
sudo systemctl status neo4j
```

```
● neo4j.service - Neo4j Graph Database
   Loaded: loaded (/etc/systemd/system/neo4j.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2021-01-01 13:01:02 UTC; 40min ago
 Main PID: 937 (java)
    Tasks: 62 (limit: 4401)
   CGroup: /system.slice/neo4j.service
           └─937 /usr/bin/java -cp
/var/lib/neo4j/plugins:/etc/neo4j:/usr/share/neo4j/lib/*:/var/lib/neo4j/plugins/* -XX:+UseG1GC -XX:
-OmitStackTraceInFastThrow
```

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

Delete your instance

You can run the following command to delete your instance:

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

3.3.2. Causal Clusters (VM-based)

Neo4j Enterprise is registered in GCP Marketplace.

Prerequisites

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

Deploy Neo4j via the GCP Marketplace

Deploy Neo4j Enterprise from the Google Cloud Launcher console following the interactive prompts.

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

Start using Neo4j Browser

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

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

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

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

Access your instance via SSH

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

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

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

Your cluster default configuration

The following notes are provided on your default cluster configuration.

- Ports `7687` (bolt) and `7473` (HTTPS access) are the only ports exposed to the entire internet. Consider narrowing the access to these ports to only your needed networks. External unencrypted HTTP access is disabled by default.

- Ports `5000`, `6000`, and `7000` are enabled only for internal network access (`10.0.0.8`), between the cluster nodes.
- Because cloud VMs can start and stop with different IP addresses, the configuration of these VMs is driven by a file in */etc/neo4j/neo4j.template*. Configuration changes must be made to the template, not to the */etc/neo4j/neo4j.conf* file, which is overwritten with the template substitutions at every startup. The template allows you to configure aspects of the cluster with the VMs metadata. The template's behavior and layout match the usual *neo4j.conf* file.

What's next

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

Terminating the deployment

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

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

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

Prerequisites

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

Google Cloud Deployment Manager

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

Creating a Deployment Manager stack

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

Each script contains the following configurations:

- The URL of the Neo4j stack template that tells GCP what to deploy.
- Various parameters that control how much hardware you want to use.

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

Deploying Neo4j Enterprise Edition with a Causal Cluster

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

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

```
#!/bin/bash
export NAME=neo4j-cluster
PROJECT=my-gcp-project-ID
MACHINE=n1-standard-2
DISK_TYPE=pd-ssd
DISK_SIZE=64
ZONE=us-east1-b
CORES=3
READ_REPLICAS=0
NEO4J_VERSION=4.2.19
TEMPLATE_URL=https://storage.googleapis.com/neo4j-deploy/$NEO4J_VERSION/causal-cluster/neo4j-causal-
cluster.jinja
OUTPUT=$(gcloud deployment-manager deployments create $NAME \
   --project $PROJECT \
   --template "$TEMPLATE_URL" \
   --properties "zone:'$ZONE',clusterNodes:'$CORES',readReplicas:'$READ_REPLICAS
',bootDiskSizeGb:$DISK_SIZE,bootDiskType:'$DISK_TYPE',machineType:'$MACHINE'")
echo $OUTPUT
PASSWORD=$(echo $OUTPUT | perl -ne 'm/password\s+([^\s]+)/; print $1;')
IP=$(echo $OUTPUT | perl -ne 'm/vm1URL\s+https:\/\/([^\s]+):/; print $1; ')
echo NEO4J_URI=bolt+routing://$IP
echo NEO4J_PASSWORD=$PASSWORD
echo STACK_NAME=$NAME
```

After you configure the parameters of what you are deploying, you call to `gcloud deployment-manager deployments create` to do the work. The variable `OUTPUT` contains all the information about your deployment. Then, you use `perl` to pull out the password and IP address of your new deployment, because it will have a strong randomly assigned password.

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

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

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

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

Deploying Neo4j Enterprise (or Community) Edition in standalone mode

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

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

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

```
#!/bin/bash
export PROJECT=my-gcp-project-id
export MACHINE=n1-standard-2
export DISK_TYPE=pd-ssd
export DISK_SIZE=64GB
export ZONE=us-east1-b
export NEO4J_VERSION=4.2.19
export PASSWORD=$(head -n 20 /dev/urandom | md5)
export STACK_NAME=neo4j-standalone
export IMAGE=neo4j-enterprise-1-3-5-3-apoc
# Setup firewalling.
echo "Creating firewall rules"
gcloud compute firewall-rules create "$STACK_NAME" \
    --allow tcp:7473,tcp:7687 \
    --source-ranges 0.0.0.0/0 \
    --target-tags neo4j \
    --project $PROJECT
if [ $? -ne 0 ] ; then
   echo "Firewall creation failed.  Bailing out"
   exit 1
fi
echo "Creating instance"
OUTPUT=$(gcloud compute instances create $STACK_NAME \
   --project $PROJECT \
   --image $IMAGE \
   --tags neo4j \
   --machine-type $MACHINE \
   --boot-disk-size $DISK_SIZE \
   --boot-disk-type $DISK_TYPE \
   --image-project launcher-public)
echo $OUTPUT
# Pull out the IP addresses, and toss out the private internal one (10.*)
IP=$(echo $OUTPUT | grep -oE '((1?[0-9][0-9]?|2[0-4][0-9]|25[0-5])\.){3}(1?[0-9][0-9]?|2[0-4][0-9]|25[0-
5])' | grep --invert-match "^10\.")
echo "Discovered new machine IP at $IP"
tries=0
while true ; do
   OUTPUT=$(echo "CALL dbms.changePassword('$PASSWORD');" | cypher-shell -a $IP -u neo4j -p "neo4j" 2>&1)
   EC=$?
   echo $OUTPUT

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

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

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

3.4. Neo4j on Microsoft Azure

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

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

3.4.1. Single instances (VM-based)

Prerequisites

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

Deploy Neo4j via the Azure Marketplace

Deploy Neo4j Enterprise VM from the Azure Marketplace following the interactive prompts.

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

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

Access your new instance

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

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

Access your instance via SSH

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

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

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

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

Deleting the instance

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

3.4.2. Causal Clusters (VM-based)

Prerequisites

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

Deploy Neo4j from the Azure Marketplace

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

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

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

Start using Neo4j Browser

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

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

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

Access your instance via SSH

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

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

Your cluster default configuration

The following notes are provided on your default cluster configuration.

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

What's next

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

Terminating the deployment

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

3.4.3. Neo4j deployments automation on Azure

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

Prerequisites

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

Azure Resource Manager

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

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

Creating an ARM deployment job

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

Each script contains the following configurations:

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

Deploying Neo4j Enterprise Causal Cluster

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

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

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

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

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

Deploying Neo4j Enterprise Standalone

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

Neo4j provides the VM through an Azure marketplace offer. To refer to the right VM image, you need to know the publisher (that's Neo4j), the "offer" (Neo4j version series), and SKU (the particular Neo4j version). Because you are not using ARM for this deployment, the script polls and waits until the VM service comes up, and then changes the Neo4j default password. At the top, you can choose a different password for the `neo4j` user as for a system administrator. Make sure to customize the `SUBSCRIPTION` variable to make this work.

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

```
# Change password
echo "Checking if Neo4j is up and changing password...."
while true; do
   if curl -s -I http://$IP_ADDRESS:7474 | grep "200 OK"; then
     echo "Neo4j is up; changing default password" 2>&1
     curl -v -H "Content-Type: application/json" \
       -XPOST -d '{"password":"'$NEO4J_PASSWORD'"}' \
       -u neo4j:neo4j \
       http://$IP_ADDRESS:7474/user/neo4j/password 2>&1
     echo "Password reset, signaling success" 2>&1
     break
   fi
   echo "Waiting for neo4j to come up" 2>&1
   sleep 1
done
echo NEO4J_URI=$NEO4J_URI
exit 0
```

Cleaning up and removing your deployment

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

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

[2] https://tools.ietf.org/html/rfc5737

Chapter 4. Docker

This chapter describes the following:

- Introduction — Introduction to running Neo4j in a Docker container.
- Configuration — How to configure Neo4j to run in a Docker container.
- Clustering — How to set up Causal Clustering when using Docker.
- Docker specific operations - Descriptions of various operations that are specific to using Docker.
- Security - Information about using encryption with the Docker image.
- Docker maintenance operations How to maintain Neo4j when running in a Docker container.
- Docker specific configuration settings - A conversion table for the Neo4j configuration settings to Docker format.

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

4.1. Introduction

Docker can be downloaded for MacOS, Windows, and Linux operating systems from https://www.docker.com/get-started. There is an official Neo4j image on DockerHub that provides a standard, ready-to-run package of Neo4j. From the DockerHub repo, it is possible to run Community Edition or Enterprise Edition with a variety of Neo4j versions.

4.1.1. Neo4j editions

Tags are available for both Community Edition and Enterprise Edition. Version-specific Enterprise Edition tags have an `-enterprise` suffix, for example: `neo4j:4.2.19-enterprise`. Community Edition tags have no suffix, for example `neo4j:4.2.19`. The latest Neo4j Enterprise Edition release is available as `neo4j:enterprise`.

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

Neo4j Enterprise Edition license

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

>
>
> Email inquiries can be directed to: licensing@neo4j.com
>
> More information is also available at: https://neo4j.com/licensing/

To accept the license agreement, set the environment variable `NEO4J_ACCEPT_LICENSE_AGREEMENT=yes`:

```
NEO4J_ACCEPT_LICENSE_AGREEMENT=yes
```

4.1.2. Using the Neo4j Docker image

A Neo4j container can be started using the following command:

```
docker run \
    --restart always \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j/data:/data \
    neo4j:4.2.19
```

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

Table 10. Options for `docker run`

Option	Description	Example
`--name`	Name your container to avoid generic ID	`docker run --name myneo4j neo4j`
`-p`	Specify which container port to expose	`docker run -p7687:7687 neo4j`
`-d`	Detach container to run in background	`docker run -d neo4j`
`-v`	Bind mount a volume	`docker run -v $HOME/neo4j/data:/data neo4j`
`--env`	Set config as environment variables for Neo4j database	`docker run --env NEO4J_AUTH=neo4j/test`
`--restart`	Control whether Neo4j containers start automatically when they exit, or when Docker restarts.	`docker run --restart always`
`--help`	Output full list of `docker run` options	`docker run --help`

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

If you no longer want to have the container auto-start on machine boot, you can disable this setting using the flag `no`:

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

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

4.1.3. Offline installation of Neo4j Docker image

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

This is an example command to save the `neo4j:4.2.19` image to a `.tar` file:

```
docker save -o neo4j-4.2.19.tar neo4j:4.2.19
```

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

```
docker load --input neo4j-4.2.19.tar
```

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

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

4.1.4. Using `NEO4J_AUTH` to set an initial password

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

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

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

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

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

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

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

4.1.5. Running Neo4j as a non-root user

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

```
docker run \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j/data:/data \
    --volume=$HOME/neo4j/logs:/logs \
    --user="$(id -u):$(id -g)" \
    neo4j:4.2.19
```

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

4.2. Configuration

The default configuration provided by this image is intended for learning about Neo4j, but must be modified to make it suitable for production use. In particular, the default memory assignments to Neo4j are very limited (`NEO4J_dbms_memory_pagecache_size=512M` and `NEO4J_dbms_memory_heap_max__size=512M`), to allow multiple containers to be run on the same server. You can read more about configuring Neo4j in the Docker specific configuration settings.

There are three ways to modify the configuration:

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

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

4.2.1. Environment variables

Pass environment variables to the container when you run it.

```
docker run \
    --detach \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j/data:/data \
    --volume=$HOME/neo4j/logs:/logs \
    --env NEO4J_dbms_memory_pagecache_size=4G \
    neo4j:4.2.19
```

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

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

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

```
--env NEO4J_dbms_tx__log_rotation_size
```

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

```
--env NEO4J_dbms_jvm_additional="-Dcom.sun.management.jmxremote.authenticate=true
-Dcom.sun.management.jmxremote.ssl=false -Dcom.sun.management.jmxremote.password.file=
$HOME/conf/jmx.password -Dcom.sun.management.jmxremote.access.file=$HOME/conf/jmx.access
-Dcom.sun.management.jmxremote.port=3637"
```

4.2.2. Mounting the /conf volume

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

```
docker run \
    --detach \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j/data:/data \
    --volume=$HOME/neo4j/logs:/logs \
    --volume=$HOME/neo4j/conf:/conf \
    neo4j:4.2.19
```

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

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

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

```
docker run --rm \
    --volume=$HOME/neo4j/conf:/conf \
    neo4j:4.2.19 dump-config
```

4.2.3. Customize a Neo4j Docker image

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

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

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

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

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

/example/Dockerfile

FROM neo4j:4.2.19-enterprise
COPY extension_script.sh /extension_script.sh
ENV EXTENSION_SCRIPT=/extension_script.sh

/example/extension_script.sh

echo "extension logic"

# Build the custom image:

docker build --file /example/Dockerfile --tag neo4j:4.2.19-enterprise-custom-container-1 /example

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

docker run --interactive --tty --name custom-container-1 -p7687:7687 -p7474:7474 -p7473:7473 --env
NEO4J_AUTH=neo4j/password --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes neo4j:4.2.19-enterprise-custom-
container-1
```

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

4.3. Clustering

4.3.1. Deploy a Causal Cluster with Docker Compose

You can deploy a Causal Cluster using Docker Compose. Docker Compose is a management tool for Docker containers. You use a YAML file to define the infrastructure of all your Causal Cluster members in one file. Then, by running the single command `docker-compose up`, you create and start all the members without the need to invoke each of them individually. For more information about Docker Compose, see the Docker Compose official documentation.

Prerequisites

- Verify that you have installed Docker Compose. For more information, see the Install Docker Compose official documentation.

Procedure

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

Example 2. Example docker-compose.yml file

```
version: '3.8'

x-shared:
  &common
  NEO4J_AUTH: neo4j/foobar  ①
  NEO4J_ACCEPT_LICENSE_AGREEMENT: "yes"
  NEO4J_causal__clustering_initial__discovery__members: core1:5000,core2:5000,core3:5000 ②
  NEO4J_dbms_memory_pagecache_size: "100M" ③
  NEO4J_dbms_memory_heap_initial__size: "100M" ④

x-shared-core:
  &common-core
  <<: *common
  NEO4J_dbms_mode: CORE
  NEO4J_causal__clustering_minimum__core__cluster__size__at__formation: 3

networks: ⑤
  lan:

services:

  core1:
    image: neo4j:4.2-enterprise
    networks:
      - lan ⑥
    ports: ⑦
      - "7474:7474"
      - "7687:7687"
    environment:
      <<: *common-core
      NEO4J_causal__clustering_discovery__advertised__address: core1:5000 ⑧
      NEO4J_causal__clustering_transaction__advertised__address: core1:6000 ⑨
      NEO4J_causal__clustering_raft__advertised__address: core1:7000 ⑩

  core2:
    image: neo4j:4.2-enterprise
    networks:
      - lan
    ports:
      - "7475:7474"
      - "7688:7687"
    environment:
      <<:  *common-core
      NEO4J_causal__clustering_discovery__advertised__address: core2:5000
      NEO4J_causal__clustering_transaction__advertised__address: core2:6000
      NEO4J_causal__clustering_raft__advertised__address: core2:7000

  core3:
    image: neo4j:4.2-enterprise
    networks:
      - lan
    ports:
      - "7476:7474"
      - "7689:7687"
    environment:
      <<:  *common-core
      NEO4J_causal__clustering_discovery__advertised__address: core3:5000
      NEO4J_causal__clustering_transaction__advertised__address: core3:6000
      NEO4J_causal__clustering_raft__advertised__address: core3:7000

  readreplica1:
    image: neo4j:4.2-enterprise
    networks:
      - lan
    ports:
      - "7477:7474"
      - "7690:7687"
    environment:
      <<:  *common
      NEO4J_dbms_mode: READ_REPLICA
      NEO4J_causal__clustering_discovery__advertised__address: readreplica1:5000
      NEO4J_causal__clustering_transaction__advertised__address: readreplica1:6000
      NEO4J_causal__clustering_raft__advertised__address: readreplica1:7000
```

① Initial password for the container.

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

② The values of `initial_discovery_members` match the advertised addresses and ports of the `NEO4J_causalClustering_discoveryAdvertisedAddress` setting.

③ Setting that specifies how much memory Neo4j is allowed to use for the page cache.

④ Setting that specifies the initial JVM heap size.

For further information, Memory configuration.

⑤ Custom top-level network.

For more information on how and why to use custom networks, see Docker official documentation.

⑥ Service-level network, which specifies the networks, from the list of the top-level networks (in this case only `lan`), that the server will connect to.

⑦ The ports that will be accessible from outside the container - HTTP (7474) and Bolt (7687).

For more information on the Neo4j ports, see Ports.

⑧ Address (the public hostname/IP address of the machine) and port setting that specifies where this instance advertises for discovery protocol messages from other members of the cluster.

⑨ Address (the public hostname/IP address of the machine) and port setting that specifies where this instance advertises for requests for transactions in the transaction-shipping catchup protocol.

⑩ Address (the public hostname/IP address of the machine) and port setting that specifies where this instance advertises for Raft messages within the Core cluster.

2. Deploy your Causal Cluster by running `docker-compose up` from your project folder.
3. Open core1 at http://core1-public-address:7474.
4. Authenticate with the default `neo4j/your_password` credentials.
5. Check the status of the cluster by running the following in Neo4j Browser:

```
:sysinfo
```

4.3.2. Deploy a Causal Cluster using environment variables

You can set up containers in a cluster to talk to each other using environment variables. Each container must have a network route to each of the others, and the `NEO4J_causal__clustering_expected__core__cluster__size` and `NEO4J_causal__clustering_initial__discovery__members` environment variables must be set for Cores. Read Replicas only need to define `NEO4J_causal__clustering_initial__discovery__members`.

Causal Cluster environment variables Enterprise edition

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

- `NEO4J_dbms_mode`: the database mode, defaults to `SINGLE`, set to `CORE` or `READ_REPLICA` for Causal Clustering.
- `NEO4J_causal__clustering_expected__core__cluster__size`: the initial cluster size (number of Core instances) at startup.
- `NEO4J_causal__clustering_initial__discovery__members`: the network addresses of an initial set of Core cluster members.
- `NEO4J_causal__clustering_discovery__advertised__address`: hostname/IP address and port to advertise for member discovery management communication.
- `NEO4J_causal__clustering_transaction__advertised__address`: hostname/IP address and port to advertise for transaction handling.
- `NEO4J_causal__clustering_raft__advertised__address`: hostname/IP address and port to advertise for cluster communication.

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

Set up a Causal Cluster on a single Docker host

Within a single Docker host, you can use the default ports for HTTP, HTTPS, and Bolt. For each container, these ports are mapped to a different set of ports on the Docker host.

Example of a `docker run` command for deploying a cluster with 3 COREs

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

docker run --name=core1 --detach --network=cluster \
    --publish=7474:7474 --publish=7473:7473 --publish=7687:7687 \
    --hostname=core1 \
    --env NEO4J_dbms_mode=CORE \
    --env NEO4J_causal__clustering_expected__core__cluster__size=3 \
    --env NEO4J_causal__clustering_initial__discovery__members=core1:5000,core2:5000,core3:5000 \
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
    --env NEO4J_dbms_connector_bolt_advertised__address=localhost:7687 \
    --env NEO4J_dbms_connector_http_advertised__address=localhost:7474 \
    neo4j:4.2.19-enterprise

docker run --name=core2 --detach --network=cluster \
    --publish=8474:7474 --publish=8473:7473 --publish=8687:7687 \
    --hostname=core2 \
    --env NEO4J_dbms_mode=CORE \
    --env NEO4J_causal__clustering_expected__core__cluster__size=3 \
    --env NEO4J_causal__clustering_initial__discovery__members=core1:5000,core2:5000,core3:5000 \
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
    --env NEO4J_dbms_connector_bolt_advertised__address=localhost:8687 \
    --env NEO4J_dbms_connector_http_advertised__address=localhost:8474 \
    neo4j:4.2.19-enterprise

docker run --name=core3 --detach --network=cluster \
    --publish=9474:7474 --publish=9473:7473 --publish=9687:7687 \
    --hostname=core3 \
    --env NEO4J_dbms_mode=CORE \
    --env NEO4J_causal__clustering_expected__core__cluster__size=3 \
    --env NEO4J_causal__clustering_initial__discovery__members=core1:5000,core2:5000,core3:5000 \
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
    --env NEO4J_dbms_connector_bolt_advertised__address=localhost:9687 \
    --env NEO4J_dbms_connector_http_advertised__address=localhost:9474 \
    neo4j:4.2.19-enterprise
```

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

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

```
docker run --name=read-replica1 --detach --network=cluster \
         --publish=10474:7474 --publish=10473:7473 --publish=10687:7687 \
         --hostname=read-replica1 \
         --env NEO4J_dbms_mode=READ_REPLICA \
         --env NEO4J_causal__clustering_initial__discovery__members=core1:5000,core2:5000,core3:5000 \
         --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
         --env NEO4J_dbms_connector_bolt_advertised__address=localhost:10687 \
         --env NEO4J_dbms_connector_http_advertised__address=localhost:10474 \
         neo4j:4.2.19-enterprise
```

Set up a Causal Cluster on multiple Docker hosts

To get the Causal Cluster high-availability characteristics, however, it is more sensible to put the cluster nodes on different physical machines.

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

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

```
docker run --name=neo4j-core --detach \
         --network=host \
         --publish=7474:7474 --publish=7687:7687 \
         --publish=5000:5000 --publish=6000:6000 --publish=7000:7000 \
         --hostname=public-address \
         --env NEO4J_dbms_mode=CORE \
         --env NEO4J_causal__clustering_expected__core__cluster__size=3 \
         --env NEO4J_causal__clustering_initial__discovery__members=core1-public-address:5000,core2-
public-address:5000,core3-public-address:5000 \
         --env NEO4J_causal__clustering_discovery__advertised__address=public-address:5000 \
         --env NEO4J_causal__clustering_transaction__advertised__address=public-address:6000 \
         --env NEO4J_causal__clustering_raft__advertised__address=public-address:7000 \
         --env NEO4J_dbms_connectors_default__advertised__address=public-address \
         --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
         --env NEO4J_dbms_connector_bolt_advertised__address=public-address:7687 \
         --env NEO4J_dbms_connector_http_advertised__address=public-address:7474 \
         neo4j:4.2.19-enterprise
```

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

Please note that if you are starting a Read Replica as above, you must publish the discovery port. For example, `--publish=5000:5000`.

In versions prior to Neo4j 4.0, this was only necessary with Core servers.

4.4. Docker specific operations

4.4.1. Use Neo4j Admin

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

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

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

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

4.4.2. Use Neo4j Import

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

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

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

Prerequisites

- Verify that you have created the folders that you want to mount as volumes to the Neo4j docker container.
- Verify that the CSV files that you want to load into Neo4j are formatted as per CSV header format.

- Verify that you have added the CSV files to the folder that will be mounted to */import* in your container.

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

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

```
docker run --interactive --tty --rm \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j/data:/data \
    --volume=$HOME/neo4j/import:/import \
    --user="$(id -u):$(id -g)" \
    neo4j:4.2.19 \
neo4j-admin import --nodes=Movies=/import/movies_header.csv,/import/movies.csv \
--nodes=Actors=/import/actors_header.csv,/import/actors.csv \
--relationships=ACTED_IN=/import/roles_header.csv,/import/roles.csv
```

4.4.3. Use Neo4j Admin for memory recommendations

The neo4j-admin memrec command with the argument `--docker` outputs environmental variables that can be passed to a Neo4j docker container. The recommended use is to save the generated environment variables to a file and pass the file to a docker container using the `--env-file` docker option. The following example shows how `neo4j-admin memrec --docker` provides a memory recommendation in a docker-friendly format.

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

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

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

4.4.4. Use Cypher Shell

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

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

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

Retrieve data from a database in a Neo4j Docker container

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

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

```
docker run --interactive --tty --name <containerID/name> \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j/data:/data \
    --user="$(id -u):$(id -g)" \
    neo4j:4.2.19
```

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

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

3. Retrieve some data.

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

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

Pass a Cypher script file to a Neo4j Docker container

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

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

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

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

example.cypher script

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

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

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

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

docker run --rm \
--volume /path/to/local/examples:/examples \
--publish=7474:7474 \
--publish=7687:7687 \
--env NEO4J_AUTH=neo4j/<password> \
neo4j:4.2.19

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

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

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

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

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

docker run --rm \
--volume /path/to/local/examples:/examples \
--publish=7474:7474 \
--publish=7687:7687 \
--env NEO4J_AUTH=neo4j/<password> \
neo4j:4.2.19

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

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

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

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

Invoke `curl` with Cypher Shell

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

Invoke `cat` with Cypher Shell

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

Example output

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

These commands take the contents of the script file and pass it into the Docker container using Cypher Shell. Then, they run a Cypher example, `LOAD CSV` dataset, which might be hosted somewhere on a server

(with `curl`), create indexes, constraints, or do other administrative operations.

4.4.5. Install user-defined procedures

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

```
docker run --publish=7474:7474 --publish=7687:7687 --volume=$HOME/neo4j/plugins:/plugins neo4j:4.2.19
```

4.4.6. Configure Neo4j Labs plugins

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

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

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

The `NEO4JLABS_PLUGINS` environment variable can be used to specify the plugins to install using this method. This should be set to a JSON-formatted list of supported plugins.

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

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

and run the following command:

```
docker run -it --rm \
  --publish=7474:7474 --publish=7687:7687 \
  --user="$(id -u):$(id -g)" \
  -e NEO4J_AUTH=none \
  --env NEO4JLABS_PLUGINS='["apoc"]' \
  neo4j:4.2.19
```

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

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

Table 11. Supported Neo4j Labs plugins

Name	Key	Further information
APOC	`apoc`	https://neo4j.com/labs/apoc/
Bloom Enterprise edition [3]	`bloom`	Neo4j Bloom

Name	Key	Further information
Graph Data Science	`graph-data-science`	Graph Data Science
Neo Semantics	`n10s`	https://neo4j.com/labs/nsmtx-rdf/
Streams	`streams`	Neo4j Streaming Data Integrations User Guide
Graph-algorithms Deprecated	`graph-algorithms`	Graph Algorithms

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

4.5. Security

4.5.1. SSL Encryption

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

Set up your certificate folders

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

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

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

2. Create a local folder to store your certificates.

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

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

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

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

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

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

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

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

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

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

Configure SSL via *neo4j.conf*

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

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


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

For more information on configuring SSL policies, see Configuration.

For more information on configuring connectors, see Configuration options.

Example 4. A `docker run` command that launches a container with SSL policy enabled via neo4j.conf.

```
docker run \
    --publish=7473:7473 \ ①
    --publish=7687:7687 \
    --user="$(id -u):$(id -g)" \ ②
    --volume=$HOME/neo4j/certificates:/ssl \ ③
    --volume=$HOME/neo4j/conf:/conf \ ④
    neo4j:4.2.19
```

① The port to access the HTTPS endpoint.

② Docker will be started as the current user (assuming the current user has read-access to the certificates).

③ The volume that contains the SSL policies that you want to set up Neo4j to use.

④ The volume that contains the *neo4j.conf* file. In this example, the *neo4j.conf* is in the `$HOME/neo4j/conf` folder of the host.

Configure SSL via Docker environment variables

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

Example 5. A `docker run` command that launches a container with SSL policy enabled via Docker environment variables.

```
docker run \
    --publish=7473:7473 \ ①
    --publish=7687:7687 \
    --user="$(id -u):$(id -g)" \ ②
    --volume=$HOME/neo4j/certificates:/ssl \ ③
    --env NEO4J_dbms_connector_https_enabled=true \ ④
    --env NEO4J_dbms_ssl_policy_https_enabled=true \ ⑤
    --env NEO4J_dbms_ssl_policy_https_base__directory=/ssl/https \ ⑥
    neo4j:4.2.19
```

① The port to access the HTTPS endpoint.

② Docker will be started as the current user (assuming the current user has read-access to the certificates).

③ The volume that contains the SSL policies that you want to set up Neo4j to use.

④ The HTTPS connector is disabled by default. Therefore, you must set `dbms.connector.https.enabled` to `true`, to be able Neo4j to listen for incoming connections on the HTTPS port. However, for the Bolt SSL policy, you do not have to pass this parameter as the Bolt connector is enabled by default.

⑤ The SSL policy that you want to set up for Neo4j.

⑥ The base directory under which SSL certificates and keys are searched for. Note that the value is the docker volume folder */ssl/https* and not the */certificate/https* folder of the host.

4.6. Docker maintenance operations

4.6.1. Dump and load a Neo4j database (offline)

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

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

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

```
docker run --interactive --tty --rm \
   --publish=7474:7474 --publish=7687:7687 \
   --volume=$HOME/neo4j/data:/data \  ①
   --volume=$HOME/neo4j/backups:/backups \  ②
   --user="$(id -u):$(id -g)" \
   neo4j:4.2.19 \
neo4j-admin dump --database=neo4j --to=/backups/<dump-name>.dump
```

① The volume that contains the database that you want to dump.

② The volume that will be used for the dumped database.

Example 7. Invoke `neo4j-admin load` to load your data into the new database.

```
docker run --interactive --tty --rm \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j/data:/data \ ①
    --volume=$HOME/neo4j/backups:/backups \ ②
    --user="$(id -u):$(id -g)" \
    neo4j:4.2.19 \
neo4j-admin load --from=/backups/<dump-name>.dump --database=neo4j --force
```

① The volume that contains the database, into which you want to load the dumped data.

② The volume that stores the database dump.

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

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

4.6.2. Back up and restore a Neo4j database (online) Enterprise edition

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

Back up a database Enterprise edition

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

Example 8. A `docker run` command that mounts the host backup folder to a Neo4j container.

```
docker run --name <container name> \
    --detach \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j-enterprise/data:/data \ ①
    --volume=$HOME/neo4j-enterprise/backups:/backups \ ②
    --user="$(id -u):$(id -g)" \
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \ ③
    --env NEO4J_dbms_backup_enabled=true \ ④
    neo4j:4.2.19-enterprise
```

① The volume that contains the database that you want to back up.

② The volume that will be used for the database backup.

③ The environment variable that states that you have accepted the Neo4j Enterprise Edition license agreement.

④ The environment variable that enables online backups.

Example 9. Invoke `neo4j-admin backup` to back up an online database.

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

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

Restore a database Enterprise edition

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

Example 10. A `docker run` command that creates a container to be used for restoring a database backup.

```
docker run --name <container name> \
    --detach \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j-enterprise/data:/data \ ①
    --volume=$HOME/neo4j-enterprise/backups:/backups \ ②
    --user="$(id -u):$(id -g)" \
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \ ③
    neo4j:4.2.19-enterprise
```

① The volume that contains all your databases.

② The volume that contains the database backup.

③ The environment variable that states that you have accepted the Neo4j Enterprise Edition license agreement.

Example 11. Invoke `cypher-shell` to stop the database that you want to use for the backup restore.

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

Example 12. Invoke `neo4j-admin restore` to restore a database backup.

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

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

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

4.6.3. Upgrade Neo4j on Docker

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

```
docker run \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j/data:/data \ ①
    --env dbms_allow__upgrade=true \ ②
    neo4j:4.2.19 \③
```

① The volume that contains the database that you want to upgrade.

② The environment variable that enables the upgrade.

③ The new version of the Neo4j Docker image to which you want to upgrade your database.

The upgrade to a later patch release of Neo4j 4.2 is straightforward — stop the container and then restart it using the later Neo4j docker image. For more details on upgrading, see Upgrade and Migration Guide → Upgrade to a newer PATCH release.

4.6.4. Monitor Neo4j

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

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

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

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

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

4.7. Docker specific configuration settings

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

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

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

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

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

Neo4j format	Docker format
`browser.allow_outgoing_connections`	`NEO4J_browser_allow__outgoing__connections`
`browser.credential_timeout`	`NEO4J_browser_credential__timeout`
`browser.post_connect_cmd`	`NEO4J_browser_post__connect__cmd`
`browser.remote_content_hostname_whitelist`	`NEO4J_browser_remote__content__hostname__whitelist`
`browser.retain_connection_credentials`	`NEO4J_browser_retain__connection__credentials`
`causal_clustering.catch_up_client_inactivity_timeout`	`NEO4J_causal__clustering_catch__up__client__inactivity__timeout`
`causal_clustering.catchup_batch_size`	`NEO4J_causal__clustering_catchup__batch__size`

Neo4j format	Docker format
causal_clustering.cluster_allow_reads_on_followers	NEO4J_causal__clustering_cluster__allow__reads__on__followers
causal_clustering.cluster_binding_timeout	NEO4J_causal__clustering_cluster__binding__timeout
causal_clustering.cluster_topology_refresh	NEO4J_causal__clustering_cluster__topology__refresh
causal_clustering.command_applier_parallelism	NEO4J_causal__clustering_command__applier__parallelism
causal_clustering.connect-randomly-to-server-group	NEO4J_causal__clustering_connect-randomly-to-server-group
causal_clustering.discovery_advertised_address	NEO4J_causal__clustering_discovery__advertised__address
causal_clustering.discovery_listen_address	NEO4J_causal__clustering_discovery__listen__address
causal_clustering.discovery_type	NEO4J_causal__clustering_discovery__type
causal_clustering.election_failure_detection_window	NEO4J_causal__clustering_election__failure__detection__window
causal_clustering.enable_pre_voting	NEO4J_causal__clustering_enable__pre__voting
causal_clustering.global_session_tracker_state_size	NEO4J_causal__clustering_global__session__tracker__state__size
causal_clustering.handshake_timeout	NEO4J_causal__clustering_handshake__timeout
causal_clustering.in_flight_cache.max_entries	NEO4J_causal__clustering_in__flight__cache_max__entries
causal_clustering.in_flight_cache.type	NEO4J_causal__clustering_in_flight_cache_type
causal_clustering.initial_discovery_members	NEO4J_causal__clustering_initial__discovery__members
causal_clustering.join_catch_up_max_lag	NEO4J_causal__clustering_join__catch__up__max__lag
causal_clustering.join_catch_up_timeout	NEO4J_causal__clustering_join__catch__up__timeout
causal_clustering.kubernetes.address	NEO4J_causal__clustering_kubernetes_address
causal_clustering.kubernetes.ca_crt	NEO4J_causal__clustering_kubernetes_ca__crt
causal_clustering.kubernetes.label_selector	NEO4J_causal__clustering_kubernetes_label__selector
causal_clustering.kubernetes.namespace	NEO4J_causal__clustering_kubernetes_namespace
causal_clustering.kubernetes.service_port_name	NEO4J_causal__clustering_kubernetes_service_port_name
causal_clustering.kubernetes.token	NEO4J_causal__clustering_kubernetes_token
causal_clustering.last_applied_state_size	NEO4J_causal__clustering_last__applied__state__size
causal_clustering.leader_election_timeout	NEO4J_causal__clustering_leader__election__timeout
causal_clustering.leader_failure_detection_window	NEO4J_causal__clustering_leader__failure__detection__window
causal_clustering.leadership_balancing	NEO4J_causal__clustering_leadership__balancing
causal_clustering.load_balancing.plugin	NEO4J_causal__clustering_load__balancing_plugin
causal_clustering.load_balancing.shuffle	NEO4J_causal__clustering_load__balancing_shuffle
causal_clustering.log_shipping_max_lag	NEO4J_causal__clustering_log__shipping__max__lag
causal_clustering.log_shipping_retry_timeout	NEO4J_causal__clustering_log__shipping__retry__timeout
causal_clustering.middleware.logging.level	NEO4J_causal__clustering_middleware_logging_level
causal_clustering.minimum_core_cluster_size_at_formation	NEO4J_causal__clustering_minimum__core__cluster__size__at__formation

Neo4j format	Docker format
causal_clustering.minimum_core_cluster_size_at_runtime	NEO4J_causal__clustering_minimum__core__cluster__size__at__runtime
causal_clustering.multi_dc_license	NEO4J_causal__clustering_multi__dc__license
causal_clustering.protocol_implementations.catchup	NEO4J_causal__clustering_protocol__implementations_catchup
causal_clustering.protocol_implementations.compression	NEO4J_causal__clustering_protocol__implementations_compression
causal_clustering.protocol_implementations.raft	NEO4J_causal__clustering_protocol__implementations_raft
causal_clustering.pull_interval	NEO4J_causal__clustering_pull__interval
causal_clustering.raft_advertised_address	NEO4J_causal__clustering_raft__advertised__address
causal_clustering.raft_handler_parallelism	NEO4J_causal__clustering_raft__handler__parallelism
causal_clustering.raft_in_queue_max_bytes	NEO4J_causal__clustering_raft__in__queue__max__bytes
causal_clustering.raft_listen_address	NEO4J_causal__clustering_raft__listen__address
causal_clustering.raft_log_implementation	NEO4J_causal__clustering_raft__log__implementation
causal_clustering.raft_log_prune_strategy	NEO4J_causal__clustering_raft__log__prune__strategy
causal_clustering.raft_log_pruning_frequency	NEO4J_causal__clustering_raft__log__pruning__frequency
causal_clustering.raft_log_reader_pool_size	NEO4J_causal__clustering_raft__log__reader__pool__size
causal_clustering.raft_log_rotation_size	NEO4J_causal__clustering_raft__log__rotation__size
causal_clustering.raft_membership_state_size	NEO4J_causal__clustering_raft__membership__state__size
causal_clustering.raft_term_state_size	NEO4J_causal__clustering_raft__term__state__size
causal_clustering.raft_vote_state_size	NEO4J_causal__clustering_raft__vote__state__size
causal_clustering.refuse_to_be_leader	NEO4J_causal__clustering_refuse__to__be__leader
causal_clustering.replicated_lease_state_size	NEO4J_causal__clustering_replicated__lease__state__size
causal_clustering.replication_leader_await_timeout	NEO4J_causal__clustering_replication__leader__await__timeout
causal_clustering.replication_retry_timeout_base	NEO4J_causal__clustering_replication__retry__timeout__base
causal_clustering.replication_retry_timeout_limit	NEO4J_causal__clustering_replication__retry__timeout__limit
causal_clustering.server_groups	NEO4J_causal__clustering_server__groups
causal_clustering.state_machine_apply_max_batch_size	NEO4J_causal__clustering_state__machine__apply__max__batch__size
causal_clustering.state_machine_flush_window_size	NEO4J_causal__clustering_state__machine__flush__window__size
causal_clustering.status_throughput_window	NEO4J_causal__clustering_status__throughput__window
causal_clustering.store_copy_chunk_size	NEO4J_causal__clustering_store__copy__chunk__size
causal_clustering.store_copy_max_retry_time_per_request	NEO4J_causal__clustering_store__copy__max__retry__time__per__request
causal_clustering.transaction_advertised_address	NEO4J_causal__clustering_transaction__advertised__address

Neo4j format	Docker format
causal_clustering.transaction_listen_address	NEO4J_causal__clustering_transaction__listen__address
causal_clustering.unknown_address_logging_throttle	NEO4J_causal__clustering_unknown__address__logging__throttle
causal_clustering.upstream_selection_strategy	NEO4J_causal__clustering_upstream__selection__strategy
causal_clustering.user_defined_upstream_strategy	NEO4J_causal__clustering_user__defined__upstream__strategy
cypher.default_language_version	NEO4J_cypher_default__language__version
cypher.forbid_exhaustive_shortestpath	NEO4J_cypher_forbid__exhaustive__shortestpath
cypher.forbid_shortestpath_common_nodes	NEO4J_cypher_forbid__shortestpath__common__nodes
cypher.hints_error	NEO4J_cypher_hints__error
cypher.lenient_create_relationship	NEO4J_cypher_lenient__create__relationship
cypher.min_replan_interval	NEO4J_cypher_min__replan__interval
cypher.planner	NEO4J_cypher_planner
cypher.statistics_divergence_threshold	NEO4J_cypher_statistics__divergence__threshold
db.temporal.timezone	NEO4J_db_temporal_timezone
dbms.allow_single_automatic_upgrade	NEO4J_dbms_allow__single__automatic__upgrade
dbms.allow_upgrade	NEO4J_dbms_allow__upgrade
dbms.backup.enabled	NEO4J_dbms_backup_enabled
dbms.backup.listen_address	NEO4J_dbms_backup_listen__address
dbms.checkpoint	NEO4J_dbms_checkpoint
dbms.checkpoint.interval.time	NEO4J_dbms_checkpoint_interval_time
dbms.checkpoint.interval.tx	NEO4J_dbms_checkpoint_interval_tx
dbms.checkpoint.iops.limit	NEO4J_dbms_checkpoint_iops_limit
dbms.config.strict_validation	NEO4J_dbms_config_strict__validation
dbms.connector.bolt.advertised_address	NEO4J_dbms_connector_bolt_advertised__address
dbms.connector.bolt.enabled	NEO4J_dbms_connector_bolt_enabled
dbms.connector.bolt.listen_address	NEO4J_dbms_connector_bolt_listen__address
dbms.connector.bolt.thread_pool_keep_alive	NEO4J_dbms_connector_bolt_thread__pool__keep__alive
dbms.connector.bolt.thread_pool_max_size	NEO4J_dbms_connector_bolt_thread__pool__max__size
dbms.connector.bolt.thread_pool_min_size	NEO4J_dbms_connector_bolt_thread__pool__min__size
dbms.connector.bolt.tls_level	NEO4J_dbms_connector_bolt_tls__level
dbms.connector.bolt.unsupported_thread_pool_shutdown_wait_time	NEO4J_dbms_connector_bolt_unsupported__thread__pool__shutdown__wait__time
dbms.connector.http.advertised_address	NEO4J_dbms_connector_http_advertised__address
dbms.connector.http.enabled	NEO4J_dbms_connector_http_enabled
dbms.connector.http.listen_address	NEO4J_dbms_connector_http_listen__address
dbms.connector.https.advertised_address`	NEO4J_dbms_connector_https_advertised__address
dbms.connector.https.enabled	NEO4J_dbms_connector_https_enabled

Neo4j format	Docker format
dbms.connector.https.listen_address	NEO4J_dbms_connector_https_listen__address
dbms.db.timezone	NEO4J_dbms_db_timezone
dbms.default_advertised_address	NEO4J_dbms_default__advertised__address
dbms.default_database	NEO4J_dbms_default__database
dbms.default_listen_address	NEO4J_dbms_default__listen__address
dbms.directories.data	NEO4J_dbms_directories_data
dbms.directories.dumps.root	NEO4J_dbms_directories_dumps_root
dbms.directories.import	NEO4J_dbms_directories_import
dbms.directories.lib	NEO4J_dbms_directories_lib
dbms.directories.logs	NEO4J_dbms_directories_logs
dbms.directories.metrics	NEO4J_dbms_directories_metrics
dbms.directories.neo4j_home	NEO4J_dbms_directories_neo4j__home
dbms.directories.plugins	NEO4J_dbms_directories_plugins
dbms.directories.run	NEO4J_dbms_directories_run
dbms.directories.transaction.logs.root	NEO4J_dbms_directories_transaction_logs_root
dbms.dynamic.setting.whitelist	NEO4J_dbms_dynamic_setting_whitelist
dbms.filewatcher.enabled	NEO4J_dbms_filewatcher_enabled
dbms.import.csv.buffer_size	NEO4J_dbms_import_csv_buffer__size
dbms.import.csv.legacy_quote_escaping	NEO4J_dbms_import_csv_legacy__quote__escaping
dbms.index.default_schema_provider	NEO4J_dbms_index_default__schema__provider
dbms.index.fulltext.default_analyzer	NEO4J_dbms_index_fulltext_default__analyzer
dbms.index.fulltext.eventually_consistent	NEO4J_dbms_index_fulltext_eventually__consistent
dbms.index.fulltext.eventually_consistent_index_update_queue_max_length	NEO4J_dbms_index_fulltext_eventually__consistent__index__update__queue__max__length
dbms.index_sampling.background_enabled	NEO4J_dbms_index_sampling_background__enabled
dbms.index_sampling.sample_size_limit	NEO4J_dbms_index_sampling_sample__size__limit
dbms.index_sampling.update_percentage	NEO4J_dbms_index_sampling_update__percentage
dbms.index_searcher_cache_size	NEO4J_dbms_index_searcher__cache__size
dbms.jvm.additional	NEO4J_dbms_jvm_additional
dbms.lock.acquisition.timeout	NEO4J_dbms_lock_acquisition_timeout
dbms.logs.debug.level	NEO4J_dbms_logs_debug_level
dbms.logs.debug.path	NEO4J_dbms_logs_debug_path
dbms.logs.debug.rotation.delay	NEO4J_dbms_logs_debug_rotation_delay
dbms.logs.debug.rotation.keep_number	NEO4J_dbms_logs_debug_rotation_keep__number
dbms.logs.debug.rotation.size	NEO4J_dbms_logs_debug_rotation_size
dbms.logs.gc.enabled	NEO4J_dbms_logs_gc_enabled
dbms.logs.gc.options	NEO4J_dbms_logs_gc_options

Neo4j format	Docker format
dbms.logs.gc.rotation.keep_number	NEO4J_dbms_logs_gc_rotation_keep__number
dbms.logs.gc.rotation.size	NEO4J_dbms_logs_gc_rotation_size
dbms.logs.http.enabled	NEO4J_dbms_logs_http_enabled
dbms.logs.http.path	NEO4J_dbms_logs_http_path
dbms.logs.http.rotation.keep_number	NEO4J_dbms_logs_http_rotation_keep__number
dbms.logs.http.rotation.size	NEO4J_dbms_logs_http_rotation_size
dbms.logs.query.allocation_logging_enabled	NEO4J_dbms_logs_query_allocation__logging__enabled
dbms.logs.query.early_raw_logging_enabled	NEO4J_dbms_logs_query_early__raw__logging__enabled
dbms.logs.query.enabled	NEO4J_dbms_logs_query_enabled
dbms.logs.query.page_logging_enabled	NEO4J_dbms_logs_query_page__logging__enabled
dbms.logs.query.parameter_full_entities	NEO4J_dbms_logs_query_parameter__full__entities
dbms.logs.query.parameter_logging_enabled	NEO4J_dbms_logs_query_parameter__logging__enabled
dbms.logs.query.path	NEO4J_dbms_logs_query_path
dbms.logs.query.rotation.keep_number	NEO4J_dbms_logs_query_rotation_keep__number
dbms.logs.query.rotation.size	NEO4J_dbms_logs_query_rotation_size
dbms.logs.query.runtime_logging_enabled	NEO4J_dbms_logs_query_runtime__logging__enabled
dbms.logs.query.threshold	NEO4J_dbms_logs_query_threshold
dbms.logs.query.time_logging_enabled	NEO4J_dbms_logs_query_time__logging__enabled
dbms.logs.security.level	NEO4J_dbms_logs_security_level
dbms.logs.security.path	NEO4J_dbms_logs_security_path
dbms.logs.security.rotation.delay	NEO4J_dbms_logs_security_rotation_delay
dbms.logs.security.rotation.keep_number	NEO4J_dbms_logs_security_rotation_keep__number
dbms.logs.security.rotation.size	NEO4J_dbms_logs_security_rotation_size
dbms.logs.user.path	NEO4J_dbms_logs_user_path
dbms.logs.user.rotation.delay	NEO4J_dbms_logs_user_rotation_delay
dbms.logs.user.rotation.keep_number	NEO4J_dbms_logs_user_rotation_keep__number
dbms.logs.user.rotation.size	NEO4J_dbms_logs_user_rotation_size
dbms.logs.user.stdout_enabled	NEO4J_dbms_logs_user_stdout__enabled
dbms.max_databases	NEO4J_dbms_max__databases
dbms.memory.heap.initial_size	NEO4J_dbms_memory_heap_initial__size
dbms.memory.heap.max_size	NEO4J_dbms_memory_heap_max__size
dbms.memory.off_heap.block_cache_size	NEO4J_dbms_memory_off__heap_block__cache__size
dbms.memory.off_heap.max_cacheable_block_size	NEO4J_dbms_memory_off__heap_max__cacheable__block__size
dbms.memory.off_heap.max_size	NEO4J_dbms_memory_off__heap_max__size
dbms.memory.pagecache.direction	NEO4J_dbms_memory_pagecache_direction
dbms.memory.pagecache.scan.prefetchers	NEO4J_dbms_memory_pagecache_scan_prefetchers

Neo4j format	Docker format
dbms.memory.pagecache.size	NEO4J_dbms_memory_pagecache_size
dbms.memory.pagecache.swapper	NEO4J_dbms_memory_pagecache_swapper
dbms.memory.pagecache.warmup.enable	NEO4J_dbms_memory_pagecache_warmup_enable
dbms.memory.pagecache.warmup.preload	NEO4J_dbms_memory_pagecache_warmup_preload
dbms.memory.pagecache.warmup.preload.whitelist	NEO4J_dbms_memory_pagecache_warmup_preload_whitelist
dbms.memory.pagecache.warmup.profile.interval	NEO4J_dbms_memory_pagecache_warmup_profile_interval
dbms.memory.tracking.enable	NEO4J_dbms_memory_tracking_enable
dbms.memory.transaction.datababase_max_size	NEO4J_dbms_memory_transaction_datababase__max__size
dbms.memory.transaction.global_max_size	NEO4J_dbms_memory_transaction_global__max__size
dbms.memory.transaction.max_size	NEO4J_dbms_memory_transaction_max__size
dbms.mode	NEO4J_dbms_mode
dbms.netty.ssl.provider	NEO4J_dbms_netty_ssl_provider
dbms.query_cache_size	NEO4J_dbms_query__cache__size
dbms.read_only	NEO4J_dbms_read__only
dbms.reconciler.max_backoff	NEO4J_dbms_reconciler_max__backoff
dbms.reconciler.max_parallelism	NEO4J_dbms_reconciler_max__parallelism
dbms.reconciler.may_retry	NEO4J_dbms_reconciler_may__retry
dbms.reconciler.min_backoff	NEO4J_dbms_reconciler_min__backoff
dbms.record_format	NEO4J_dbms_record_format
dbms.recovery.fail_on_missing_files	NEO4J_dbms_recovery_fail__on__missing__files
dbms.relationship_grouping_threshold	NEO4J_dbms_relationship__grouping__threshold
dbms.rest.transaction.idle_timeout	NEO4J_dbms_rest_transaction_idle__timeout
dbms.routing.advertised_address	NEO4J_dbms_routing_advertised__address
dbms.routing.driver.api	NEO4J_dbms_routing_driver_api
dbms.routing.driver.connection.connect_timeout	NEO4J_dbms_routing_driver_connection_connect__timeout
dbms.routing.driver.connection.max_lifetime	NEO4J_dbms_routing_driver_connection_max__lifetime
dbms.routing.driver.connection.pool.acquisition_timeout	NEO4J_dbms_routing_driver_connection_pool_acquisition__timeout
dbms.routing.driver.connection.pool.idle_test	NEO4J_dbms_routing_driver_connection_pool_idle__test
dbms.routing.driver.connection.pool.max_size	NEO4J_dbms_routing_driver_connection_pool_max__size
dbms.routing.driver.logging.level	NEO4J_dbms_routing_driver_logging_level
dbms.routing.enabled	NEO4J_dbms_routing_enabled
dbms.routing.listen_address	NEO4J_dbms_routing_listen__address
dbms.routing_ttl	NEO4J_dbms_routing__ttl
dbms.security.allow_csv_import_from_file_urls	NEO4J_dbms_security_allow__csv__import__from__file__urls
dbms.security.auth_cache_max_capacity	NEO4J_dbms_security_auth__cache__max__capacity
dbms.security.auth_cache_ttl	NEO4J_dbms_security_auth__cache__ttl

Neo4j format	Docker format
dbms.security.auth_cache_use_ttl	NEO4J_dbms_security_auth__cache__use__ttl
dbms.security.auth_enabled	NEO4J_dbms_security_auth__enabled
dbms.security.auth_lock_time	NEO4J_dbms_security_auth__lock__time
dbms.security.auth_max_failed_attempts	NEO4J_dbms_security_auth__max__failed__attempts
dbms.security.authentication_providers	NEO4J_dbms_security_authentication__providers
dbms.security.authorization_providers	NEO4J_dbms_security_authorization__providers
dbms.security.causal_clustering_status_auth_enabled	NEO4J_dbms_security_causal__clustering__status__auth__enabled
dbms.security.http_access_control_allow_origin	NEO4J_dbms_security_http__access__control__allow_origin
dbms.security.http_auth_whitelist	NEO4J_dbms_security_http__auth__whitelist
dbms.security.http_strict_transport_security	NEO4J_dbms_security_http__strict__transport__security
dbms.security.ldap.authentication.cache_enabled	NEO4J_dbms_security_ldap_authentication_cache__enabled
dbms.security.ldap.authentication.mechanism	NEO4J_dbms_security_ldap_authentication_mechanism
dbms.security.ldap.authentication.use_samaccountname	NEO4J_dbms_security_ldap_authentication_use__samaccountname
dbms.security.ldap.authentication.user_dn_template	NEO4J_dbms_security_ldap_authentication_user__dn__template
dbms.security.ldap.authorization.group_membership_attributes	NEO4J_dbms_security_ldap_authorization_group__membership__attributes
dbms.security.ldap.authorization.group_to_role_mapping	NEO4J_dbms_security_ldap_authorization_group__to__role__mapping
dbms.security.ldap.authorization.system_password	NEO4J_dbms_security_ldap_authorization_system__password
dbms.security.ldap.authorization.system_username	NEO4J_dbms_security_ldap_authorization_system__username
dbms.security.ldap.authorization.use_system_account	NEO4J_dbms_security_ldap_authorization_use__system__account
dbms.security.ldap.authorization.user_search_base	NEO4J_dbms_security_ldap_authorization_user__search__base
dbms.security.ldap.authorization.user_search_filter	NEO4J_dbms_security_ldap_authorization_user__search__filter
dbms.security.ldap.connection_timeout	NEO4J_dbms_security__ldap_connection__timeout
dbms.security.ldap.host	NEO4J_dbms_security__ldap__host
dbms.security.ldap.read_timeout	NEO4J_dbms_security__ldap_read__timeout
dbms.security.ldap.referral	NEO4J_dbms_security__ldap_referral
dbms.security.ldap.use_starttls	NEO4J_dbms_security__ldap_use__starttls
dbms.security.log_successful_authentication	NEO4J_dbms_security_log__successful__authentication
dbms.security.procedures.default_allowed	NEO4J_dbms_security_procedures_default__allowed
dbms.security.procedures.roles	NEO4J_dbms_security_procedures_roles
dbms.security.procedures.unrestricted	NEO4J_dbms_security_procedures_unrestricted
dbms.security.procedures.whitelist	NEO4J_dbms_security_procedures_whitelist

Neo4j format	Docker format
dbms.shutdown_transaction_end_timeout	NEO4J_dbms_shutdown__transaction__end__timeout
dbms.threads.worker_count	NEO4J_dbms_threads_worker__count
dbms.track_query_allocation	NEO4J_dbms_track__query__allocation
dbms.track_query_cpu_time	NEO4J_dbms_track__query__cpu__time
dbms.transaction.bookmark_ready_timeout	NEO4J_dbms_transaction_bookmark__ready__timeout
dbms.transaction.concurrent.maximum	NEO4J_dbms_transaction_concurrent_maximum
dbms.transaction.monitor.check.interval	NEO4J_dbms_transaction_monitor_check_interval
dbms.transaction.sampling.percentage	NEO4J_dbms_transaction_sampling_percentage
dbms.transaction.timeout	NEO4J_dbms_transaction_timeout
dbms.transaction.tracing.level	NEO4J_dbms_transaction_tracing_level
dbms.tx_log.preallocate	NEO4J_dbms_tx__log_preallocate
dbms.tx_log.rotation.retention_policy	NEO4J_dbms_tx__log_rotation_retention__policy
dbms.tx_log.rotation.size	NEO4J_dbms_tx__log_rotation_size
dbms.tx_state.memory_allocation	NEO4J_dbms_tx__state_memory__allocation
dbms.unmanaged_extension_classes	NEO4J_dbms_unmanaged__extension__classes
dbms.upgrade_max_processors	NEO4J_dbms_upgrade__max__processors
dbms.windows_service_name	NEO4J_dbms_windows__service__name
fabric.database.name	NEO4J_fabric_database_name
fabric.driver.api	NEO4J_fabric_driver_api
fabric.driver.connection.connect_timeout	NEO4J_fabric_driver_connection_connect__timeout
fabric.driver.connection.max_lifetime	NEO4J_fabric_driver_connection_max__lifetime
fabric.driver.connection.pool.acquisition_timeout	NEO4J_fabric_driver_connection_pool_acquisition__timeout
fabric.driver.connection.pool.idle_test	NEO4J_fabric_driver_connection_pool_idle__test
fabric.driver.connection.pool.max_size	NEO4J_fabric_driver_connection_pool_max__size
fabric.driver.logging.level	NEO4J_fabric_driver_logging_level
fabric.routing.servers	NEO4J_fabric_routing_servers
fabric.routing.ttl	NEO4J_fabric_routing_ttl
fabric.stream.buffer.low_watermark	NEO4J_fabric_stream_buffer_low__watermark
fabric.stream.buffer.size	NEO4J_fabric_stream_buffer_size
fabric.stream.concurrency	NEO4J_fabric_stream_concurrency
fabric.graph.<id>.uri	NEO4J_fabric_graph_<id>_uri
fabric.graph.<id>.database	NEO4J_fabric_graph_<id>_database
fabric.graph.<id>.name	NEO4J_fabric_graph_<id>_name
metrics.bolt.messages.enabled	NEO4J_metrics_bolt_messages_enabled
metrics.csv.enabled	NEO4J_metrics_csv_enabled
metrics.csv.interval	NEO4J_metrics_csv_interval

Neo4j format	Docker format
metrics.csv.rotation.keep_number	NEO4J_metrics_csv_rotation_keep__number
metrics.csv.rotation.size	NEO4J_metrics_csv_rotation_size
metrics.cypher.replanning.enabled	NEO4J_metrics_cypher_replanning_enabled
metrics.enabled	NEO4J_metrics_enabled
metrics.graphite.enabled	NEO4J_metrics_graphite_enabled
metrics.graphite.interval	NEO4J_metrics_graphite_interval
metrics.graphite.server	NEO4J_metrics_graphite_server
metrics.jmx.enabled	NEO4J_metrics_jmx_enabled
metrics.jvm.buffers.enabled	NEO4J_metrics_jvm_buffers_enabled
metrics.jvm.file.descriptors.enabled	NEO4J_metrics_jvm_file_descriptors_enabled
metrics.jvm.gc.enabled	NEO4J_metrics_jvm_gc_enabled
metrics.jvm.heap.enabled	NEO4J_metrics_jvm_heap_enabled
metrics.jvm.memory.enabled	NEO4J_metrics_jvm_memory_enabled
metrics.jvm.pause_time.enabled	NEO4J_metrics_jvm_pause__time_enabled
metrics.jvm.threads.enabled	NEO4J_metrics_jvm_threads_enabled
metrics.neo4j.causal_clustering.enabled	NEO4J_metrics_neo4j_causal__clustering_enabled
metrics.neo4j.checkpointing.enabled	NEO4J_metrics_neo4j_checkpointing_enabled
metrics.neo4j.counts.enabled	NEO4J_metrics_neo4j_counts_enabled
metrics.neo4j.data.counts.enabled	NEO4J_metrics_neo4j_data_counts_enabled
metrics.neo4j.database_operation_count.enabled	NEO4J_metrics_neo4j_database__operation__count_enabled
metrics.neo4j.logs.enabled	NEO4J_metrics_neo4j_logs_enabled
metrics.neo4j.pagecache.enabled	NEO4J_metrics_neo4j_pagecache_enabled
metrics.neo4j.pools.enabled	NEO4J_metrics_neo4j_pools_enabled
metrics.neo4j.server.enabled	NEO4J_metrics_neo4j_server_enabled
metrics.neo4j.size.enabled	NEO4J_metrics_neo4j_size_enabled
metrics.neo4j.tx.enabled	NEO4J_metrics_neo4j_tx_enabled
metrics.prefix	NEO4J_metrics_prefix
metrics.prometheus.enabled	NEO4J_metrics_prometheus_enabled
metrics.prometheus.endpoint	NEO4J_metrics_prometheus_endpoint

[3] The Bloom plugin requires a license and this needs to be provided as a shared volume. Please see Bloom User Guide → Installing Bloom in a Docker container.

Chapter 5. Configuration

The topics described are:

- The *neo4j.conf* file — An introduction to the primary configuration file in Neo4j.
- File locations — An overview of where files are stored in the different Neo4j distributions and the necessary file permissions for running Neo4j.
- Ports — An overview of the ports relevant to a Neo4j installation.
- Set initial password — How to set an initial password.
- Password and user recovery — How to recover after a lost admin password.
- Configure Neo4j connectors — How to configure Neo4j connectors.
- Configure dynamic settings — How to configure certain Neo4j parameters while Neo4j is running.
- Transaction logs — The transaction logs record all write operations in the database.

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

5.1. The neo4j.conf file

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

5.1.1. Introduction

The *neo4j.conf* file is the main source of configuration settings in Neo4j and includes the mappings of configuration setting keys to values. The location of the *neo4j.conf* file in the different configurations of Neo4j is listed in Default file locations.

Most of the configuration settings in the *neo4j.conf* file apply directly to Neo4j itself, but there are also other settings related to the Java Runtime (the JVM) on which Neo4j runs. For more information, see the JVM specific configuration settings below. Many of the configuration settings are also used by the `neo4j` launcher scripts.

5.1.2. Syntax

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

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

5.1.3. JVM-specific configuration settings

A Java virtual machine (JVM) is a virtual machine that enables a computer to run Java programs as well as programs written in other languages that are also compiled to Java bytecode. The Java heap is where the objects of a Java program live. Depending on the JVM implementation, the JVM heap size often determines how and for how long time the virtual machine performs garbage collection.

Table 12. JVM-specific settings

Setting	Description
`dbms.memory.heap.initial_size`	Sets the initial heap size for the JVM. By default, the JVM heap size is calculated based on the available system resources.
`dbms.memory.heap.max_size`	Sets the maximum size of the heap for the JVM. By default, the maximum JVM heap size is calculated based on the available system resources.
`dbms.jvm.additional`	Sets additional options for the JVM. The options are set as a string and can vary depending on JVM implementation.

If you want to have good control of the system behavior, it is recommended to set the heap size parameters to the same value to avoid unwanted full garbage collection pauses.

5.1.4. List currently active settings

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

Example 13. List currently active configuration settings

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

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

For information about dynamic settings, see Update dynamic settings.

5.2. Command expansion

Command expansion provides an additional capability to configure Neo4j by allowing you to specify scripts that set values sourced from external files. This is especially useful for:

- avoiding setting sensitive information, such as usernames, passwords, keys, etc., in the *neo4j.conf* file in plain text.
- handling the configuration settings of instances running in environments where the file system is not accessible.

5.2.1. How it works

The scripts are specified in the *neo4j.conf* file with a `$` prefix and the script to execute within brackets (), i.e., `dbms.setting=$(script_to_execute)`.
The configuration accepts any command that can be executed within a child process by the user who owns and executes the Neo4j server. This also means that, in the case of Neo4j set as a service, the commands are executed within the service.

A generic example would be:

```
neo4j.configuration.example=$(/bin/bash echo "expanded value")
```

By providing such a configuration in the *neo4j.conf* file upon server start with command expansion enabled, Neo4j evaluates the script and retrieves the value of the configuration settings prior to the instantiation of Neo4j. The values are then passed to the starting Neo4j instance and kept in memory, in the running instance.

You can also use the `curl` (https://curl.se/docs/manpage.html) command to fetch a token or value for a configuration setting. For example, you can apply an extra level of security by replacing any sensitive information in your *neo4j.conf* file with a secured reference to a provider of some sort.

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

Scripts and their syntax differ between operating systems.

5.2.2. Enabling

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

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

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

Neo4j performs the following basic security checks on the *neo4j.conf* file. If they fail, Neo4j does not evaluate the script commands in *neo4j.conf*, and the Neo4j process does not start.

On Unix (both Linux and Mac OS)

- The *neo4j.conf* file must, at least, be readable by its owner or by the user-group to which the owner belongs.
- The Neo4j process must run as a user who is either the owner of the *neo4j.conf* file or in the user-group which owns the *neo4j.conf* file.

The Linux permissions bitmask for the least restrictive permissions is `640`. More restrictive Linux permissions are also allowed. For example, the *neo4j.conf* file can have no group permissions and only be readable by its owner (`400` bitmask).

On Windows

- The *neo4j.conf* file must, at least, be readable by the user that the Neo4j process runs as.

5.2.3. Logging

The execution of scripts is logged in *neo4j.log*. For each setting that requires the execution of an external command, Neo4j adds an entry into the log file that contains information, for example:

```
… Executing the external script to retrieve the value of <setting>...
```

5.2.4. Error Handling

The scripts' execution may generate two types of errors:

- Errors during the execution — These errors are reported in the *debug.log*, with a code returned from the external execution. In this case, the execution stops and the server does not start.
- Errors for incorrect values — The returned value is not the one expected for the setting. In this case, the server does not start.

For more information, see Exit codes.

5.3. File locations

5.3.1. Default file locations

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

Table 13. Default file locations

File type	Description	Linux / macOS / Docker	Windows	Debian / RPM	Neo4j Desktop [4]
Bin	The Neo4j running script and built-in tools, such as, cypher-shell and neo4j-admin.	*<neo4j-home>/bin*	*<neo4j-home>\bin*	*/usr/bin*	From the Open dropdown menu of your Neo4j instance, select *Terminal*, and navigate to *<installation-version>/bin*.
Configuration [5]	The Neo4j configuration settings and the JMX access credentials.	*<neo4j-home>/conf/neo4j.conf*	*<neo4j-home>\conf\neo4j.conf*	*/etc/neo4j/neo4j.conf*	From the Open dropdown menu of your Neo4j instance, select *Terminal*, and navigate to *<installation-version>/conf/neo4j.conf*.
Data [6]	All data-related content, such as databases, transactions, cluster-state (if applicable), dumps, and cypher script files (from the `neo4j-admin restore` command).	*<neo4j-home>/data*	*<neo4j-home>\data*	*/var/lib/neo4j/data*	From the Open dropdown menu of your Neo4j instance, select *Terminal*, and navigate to *<installation-version>/data*.
Import	All CSV files that the command `LOAD CSV` uses as sources to import data in Neo4j.	*<neo4j-home>/import*	*<neo4j-home>\import*	*/var/lib/neo4j/import*	From the Open dropdown menu of your Neo4j instance, select *Terminal*, and navigate to *<installation-version>/import*.
Labs [7]	Contains APOC Core.	*<neo4j-home>/labs*	*<neo4j-home>\labs*	*/var/lib/neo4j/labs*	From the Open dropdown menu of your Neo4j instance, select *Terminal*, and navigate to *<installation-version>/labs*.

File type	Description	Linux / macOS / Docker	Windows	Debian / RPM	Neo4j Desktop [4]
Lib	All Neo4j dependencies.	*<neo4j-home>/lib*	*<neo4j-home>\lib*	*/usr/share/neo4j/lib*	From the Open dropdown menu of your Neo4j instance, select *Terminal*, and navigate to *<installation-version>/lib*.
Logs	The Neo4j log files.	*<neo4j-home>/logs* [8]	*<neo4j-home>\logs*	*/var/log/neo4j/* [9]	From the Open dropdown menu of your Neo4j instance, select *Terminal*, and navigate to *<installation-version>/logs*.
Metrics	The Neo4j built-in metrics for monitoring the Neo4j DBMS and each individual database.	*<neo4j-home>/metrics*	*<neo4j-home>\metrics*	*/var/lib/neo4j/metrics*	From the Open dropdown menu of your Neo4j instance, select *Terminal*, and navigate to *<installation-version>/metrics*.
Plugins	Custom code that extends Neo4j, for example, user-defined procedures, functions, and security plugins.	*<neo4j-home>/plugins*	*<neo4j-home>\plugins*	*/var/lib/neo4j/plugins*	From the Open dropdown menu of your Neo4j instance, select *Terminal*, and navigate to *<installation-version>/plugins*.
Run	The processes IDs.	*<neo4j-home>/run*	*<neo4j-home>\run*	*/var/lib/neo4j/run*	From the Open dropdown menu of your Neo4j instance, select *Terminal*, and navigate to *<installation-version>/run*.

5.3.2. Customize your file locations

The file locations can also be customized by using environment variables and options.

The locations of *<neo4j-home>* and *conf* can be configured using environment variables:

Table 14. Configuration of <neo4j-home> and conf

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

The rest of the locations can be configured by uncommenting the respective setting in the *conf/neo4j.conf* file and changing the default value.

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

5.3.3. File permissions

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

Read only

- *conf*
- *import*
- *bin*
- *lib*
- *plugins*
- *certificates*

Read and write

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

Execute

- all files in *bin*

5.4. Ports

An overview of the Neo4j-specific ports. Note that these ports are in addition to those necessary for ordinary network operation.

Specific recommendations on port openings cannot be made, as the firewall configuration must be

performed taking your particular conditions into consideration.

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

The listen address configuration settings will set the network interface and port to listen on. For example the IP-address `127.0.0.1` and port `7687` can be set with the value `127.0.0.1:7687`. The table below shows an overview of available Neo4j-specific ports and related configuration settings.

Table 15. Listen address configuration settings overview

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

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

The advertised address configuration settings are used for routing purposes. An advertised address is composed by a hostname/IP-address and port. For example the IP-address `127.0.0.1` and port `7687` can be set with the value `127.0.0.1:7687`. If a host name resolution service has been configured, the advertised address can use a hostname, for example `example.com:7687`. The table below shows an overview of available Neo4j-specific ports and related configuration settings.

Table 16. Advertised address configuration settings overview

Name	Default port	Related configuration setting
HTTP	7474	dbms.connector.http.advertised_address
HTTPS	7473	dbms.connector.https.advertised_address

Name	Default port	Related configuration setting
Bolt	7687	dbms.connector.bolt.advertised_address
Causal Cluster discovery management	5000	causal_clustering.discovery_advertised_address
Causal Cluster transaction	6000	causal_clustering.transaction_advertised_address
Causal Cluster RAFT	7000	causal_clustering.raft_advertised_address
Causal Cluster routing connector	7688	dbms.routing.advertised_address

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

5.4.1. Backup Enterprise edition

Default port: 6362

Table 17. Backup

Related configuration setting	Default value	Description
dbms.backup.listen_address	127.0.0.1:6362	Network interface and port for the backup server to listen on.
dbms.backup.enabled	true	Enable support for running online backups.

In production environments, external access to the backup port should be blocked by a firewall.

For more information, see Server configuration.

5.4.2. HTTP

Default port: 7474

Table 18. HTTP connector

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

- The HTTP connector is enabled by default.
- The network communication is unencrypted.
- Used by Neo4j Browser and the HTTP API.

For more information, see Configure connectors.

5.4.3. HTTPS

Default port: 7473

Table 19. HTTPS connector

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

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

For more information, see Configure connectors.

5.4.4. Bolt

Default port: 7687

Table 20. Bolt connector

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

- By default, the Bolt connector is **enabled**, but its encryption is **turned off**.
- Used by Cypher Shell, Neo4j Browser, and the official Neo4j drivers.

For more information, see Configure connectors.

5.4.5. Causal Cluster Enterprise edition

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

Table 21. Cluster listen address

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

Table 22. Cluster advertised address

Name	Default port	Default value	Related configuration setting
Discovery management	5000	:5000	causal_clustering.discovery_advertised_address
Transaction	6000	:6000	causal_clustering.transaction_advertised_address
RAFT	7000	:7000	causal_clustering.raft_advertised_address
Routing connector	7688	:7688	dbms.routing.advertised_address

The ports are likely be different in a production installation; therefore the potential opening of ports must be modified accordingly.

For more information, see:

- Deploy a cluster
- Settings reference

5.4.6. Graphite monitoring

Default port: 2003

Table 23. Graphite

Related configuration setting	Default value	Description
metrics.graphite.server	:2003	Hostname/IP-address and port of the Graphite server.
metrics.graphite.enabled	false	Enable exporting metrics to the Graphite server.

This is an outbound connection that enables a Neo4j instance to communicate with a Graphite server.

For further information, see Graphite and the Graphite official documentation.

5.4.7. Prometheus monitoring

Default port: 2004

Table 24. Prometheus

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

For more information, see Prometheus.

5.4.8. JMX monitoring

Default port: `3637`

Table 25. Java Management Extensions

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

For further information, see the official documentation on Monitoring and Management Using JMX.

5.4.9. Remote debugging

Default port: `5005`

Table 26. Remote debugging

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

For more information, see the Java Reference → Setup for remote debugging.

5.5. Set an initial password

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

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

Syntax:

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

Example 14. Use the `set-initial-password` command of neo4j-admin

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

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

Example 15. Use the `set-initial-password` command of neo4j-admin with the optional `--require-password-change` flag

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

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

5.6. Configure plugins

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

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

Bloom and GDS Enterprise require a license activation key to become available to the user within Neo4j. Reach out to your Neo4j account representative or request a representative to contact you.

The following plugins are supported:

Table 27. Supported Neo4j plugins

Name	Key	License required	Further information
APOC	`apoc`	✘	APOC user guide
Bloom	`bloom`	✔	Neo4j Bloom
Streams	`streams`	✘	Neo4j Streaming Data Integrations User Guide
Graph Data Science	`graph-data-science`	✘	Graph Data Science
Neo Semantics	`n10s`	✘	Neo Semantics

Name	Key	License required	Further information
Ops Manager	`ops-manager`	✖	Neo4j Ops Manager

For more information on using plugins in a different Neo4j setup, see Java-Reference → Setting up a plugin project.

5.6.1. Install and configure plugins

Here are the steps to enable the plugins:

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

```
# to enable GDS:

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


#  to enable Bloom:

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


#  to enable both GDS and Bloom:

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

3. Install the plugins.

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

All installed plugins will automatically be loaded every time Neo4j is started. Because of that, the number of plugins may impact the startup time. Install only the necessary plugins to avoid performance issues.

5.7. Dynamic settings

5.7.1. Introduction

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

Changes to the configuration at runtime are not persisted. To avoid losing changes when restarting Neo4j, make sure you update *neo4j.conf* as well.

In a clustered environment, `CALL dbms.setConfigValue` affects only the cluster member it is run against, and it is not propagated to other members. If you want to change the configuration settings on all cluster members, you have to run the procedure against each of them and update their *neo4j.conf* file.

5.7.2. Discover dynamic settings

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

Example 16. Discover dynamic settings

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

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

5.7.3. Update dynamic settings

An administrator is able to change some configuration settings at runtime, without restarting the service.

Syntax:

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

Returns:

Nothing on success.

Exceptions:

Unknown or invalid setting name.
The setting is not dynamic and can not be changed at runtime.
Invalid setting value.

The following example shows how to dynamically enable query logging.

Example 17. Set a config value

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

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

Example 18. Try to set invalid config value

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

```
Failed to invoke procedure `dbms.setConfigValue`: Caused by:
org.neo4j.graphdb.config.InvalidSettingException: Bad value 'yes' for setting
'dbms.logs.query.enabled': 'yes' not one of [OFF, INFO, VERBOSE]
```

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

Example 19. Reset a config value to default

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

5.7.4. Dynamic settings reference

causal_clustering.cluster_allow_reads_on_leader

Configure if the `dbms.routing.getRoutingTable()` procedure should include the leader as read endpoint or return only read replicas/followers.

causal_clustering.connect_randomly_to_server_group

Comma separated list of groups to be used by the connect-randomly-to-server-group selection strategy.

causal_clustering.server_groups

A list of group names for the server used when configuring load balancing and replication policies.

dbms.allow_single_automatic_upgrade

Whether to allow a system graph upgrade to happen automatically in single instance mode (dbms.mode=SINGLE).

dbms.allow_upgrade

Whether to allow a store upgrade in case the current version of the database starts against an older version of the store.

dbms.checkpoint.iops.limit

Limit the number of IOs the background checkpoint process will consume per second.

dbms.logs.debug.level

Debug log level threshold.

dbms.logs.query.allocation_logging_enabled

Log allocated bytes for the executed queries being logged.

dbms.logs.query.early_raw_logging_enabled

Log query text and parameters without obfuscating passwords.

dbms.logs.query.enabled

Log executed queries.

dbms.logs.query.page_logging_enabled

Log page hits and page faults for the executed queries being logged.

dbms.logs.query.parameter_full_entities

Log complete parameter entities including id, labels or relationship type, and properties.

dbms.logs.query.parameter_logging_enabled

Log parameters for the executed queries being logged.

dbms.logs.query.rotation.keep_number

Maximum number of history files for the query log.

dbms.logs.query.rotation.size

The file size in bytes at which the query log will auto-rotate.

dbms.logs.query.runtime_logging_enabled

Logs which runtime that was used to run the query.

dbms.logs.query.threshold

If the execution of query takes more time than this threshold, the query is logged once completed - provided query logging is set to INFO.

dbms.logs.query.time_logging_enabled

Log detailed time information for the executed queries being logged.

dbms.memory.pagecache.flush.buffer.enabled

Page cache can be configured to use a temporal buffer for flushing purposes.

dbms.memory.pagecache.flush.buffer.size_in_pages

Page cache can be configured to use a temporal buffer for flushing purposes.

dbms.memory.transaction.database_max_size

Limit the amount of memory that all transactions in one database can consume, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g').

dbms.memory.transaction.global_max_size

Limit the amount of memory that all of the running transactions can consume, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g').

dbms.memory.transaction.max_size

Limit the amount of memory that a single transaction can consume, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g').

dbms.track_query_allocation

Enables or disables tracking of how many bytes are allocated by the execution of a query.

dbms.track_query_cpu_time

Enables or disables tracking of how much time a query spends actively executing on the CPU.

dbms.transaction.concurrent.maximum

The maximum number of concurrently running transactions.

dbms.transaction.sampling.percentage

Transaction sampling percentage.

dbms.transaction.timeout

The maximum time interval of a transaction within which it should be completed.

dbms.transaction.tracing.level

Transaction creation tracing level.

dbms.tx_log.preallocate

Specify if Neo4j should try to preallocate logical log file in advance.

dbms.tx_log.rotation.retention_policy

Tell Neo4j how long logical transaction logs should be kept to backup the database.For example, "10 days" will prune logical logs that only contain transactions older than 10 days.Alternatively, "100k txs" will keep the 100k latest transactions from each database and prune any older transactions.

dbms.tx_log.rotation.size

Specifies at which file size the logical log will auto-rotate.

dbms.upgrade_max_processors

Max number of processors used when upgrading the store.

fabric.routing.servers

A comma-separated list of Fabric instances that form a routing group.

5.8. Transaction log

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

- The transaction log can be used to provide for incremental backups, as well as for cluster operations.
- For any given configuration, at least the latest non-empty transaction log will be kept.

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

The transaction log has nothing to do with log monitoring.

5.8.1. Transaction logging

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

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

An overview of configuration settings for transaction logging:

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

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

5.8.2. Log location

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

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

5.8.3. Log rotation

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

5.8.4. Log retention

Manually deleting transaction log files is not supported.

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

Other possible values are:

- `true` or `keep_all` — keep transaction logs indefinitely.

  This option is not recommended due to the effectively unbounded storage usage. Old transaction logs cannot be safely archived or removed by external jobs since safe log pruning requires knowledge about the most recent successful checkpoint.

- `false` or `keep_none` — keep only the most recent non-empty log.

  Log pruning is called only after checkpoint completion to ensure at least one checkpoint and points to a valid place in the transaction log data. In reality, this means that all transaction logs created between checkpoints will be kept for some time, and only after a checkpoint, the pruning strategy will remove them. For more details on how to speed up checkpointing, see Log pruning. To force a checkpoint, run the procedure `call db.checkpoint()`.

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

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

  Table 28. Types that can be used to control log retention

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

Type	Description	Example
days	Keep logs that contain any transaction committed within N days from the current time.	"50 days"

Example 20. Configure log retention policy

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

- Keep transaction logs indefinitely:

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

or

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

- Keep only the most recent non-empty log:

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

or

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

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

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

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

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

5.8.5. Log pruning

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

Two things are necessary for a file to be removed:

- The file must have been rotated.
- At least one checkpoint must have happened in a more recent log file.

Observing that you have more transaction log files than you expected is likely due to checkpoints either not happening frequently enough, or taking too long. This is a temporary condition and the gap between

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

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

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

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

[4] Applicable to all operating systems where Neo4j Desktop is supported.
[5] For details about neo4j.conf, see: The neo4j.conf file.
[6] The data directory is internal to Neo4j and its structure is subject to change between versions without notice.
[7] For more information, see APOC User Guide → Installation.
[8] To view *neo4j.log* in Docker, use `docker logs <containerID/name>`.
[9] To view the neo4j.log for Debian and RPM, use `journalctl --unit=neo4j`.

Chapter 6. Manage databases

This chapter describes the following:

6.1. Introduction

6.1.1. Concepts

With Neo4j 4.2 you can create and use more than one active database at the same time.

DBMS

Neo4j is a Database Management System, or *DBMS*, capable of managing multiple databases. The DBMS can manage a standalone server, or a group of servers in a Causal Cluster.

Instance

A Neo4j instance is a Java process that is running the Neo4j server code.

Transaction domain

A transaction domain is a collection of graphs that can be updated within the context of a single transaction.

Execution context

An execution context is a runtime environment for the execution of a request. In practical terms, a request may be a query, a transaction, or an internal function or procedure.

Database

A database is an administrative partition of a DBMS. In practical terms, it is a physical structure of files organized within a directory or folder, that has the same name of the database. In logical terms, a database is a container for one or more graphs.

A database defines a *transaction domain* and an *execution context*. This means that a transaction cannot span across multiple databases. Similarly, a procedure is called within a database, although its logic may access data that is stored in other databases.

A default installation of Neo4j 4.2 contains two databases:

- `system` - the system database, containing metadata on the DBMS and security configuration.
- `neo4j` - the default database, a single database for user data. This has a default name of `neo4j`. A different name can be configured before starting Neo4j for the first time.

Graph

This is a data model within a database. In Neo4j 4.0 there is only one graph within each database, and many administrative commands that refer to a specific graph do so using the database name.

In Neo4j Fabric, it is possible to refer to multiple graphs within the same transaction and Cypher query.

The following image illustrates a default installation, including the `system` database and a single database named `neo4j` for user data:

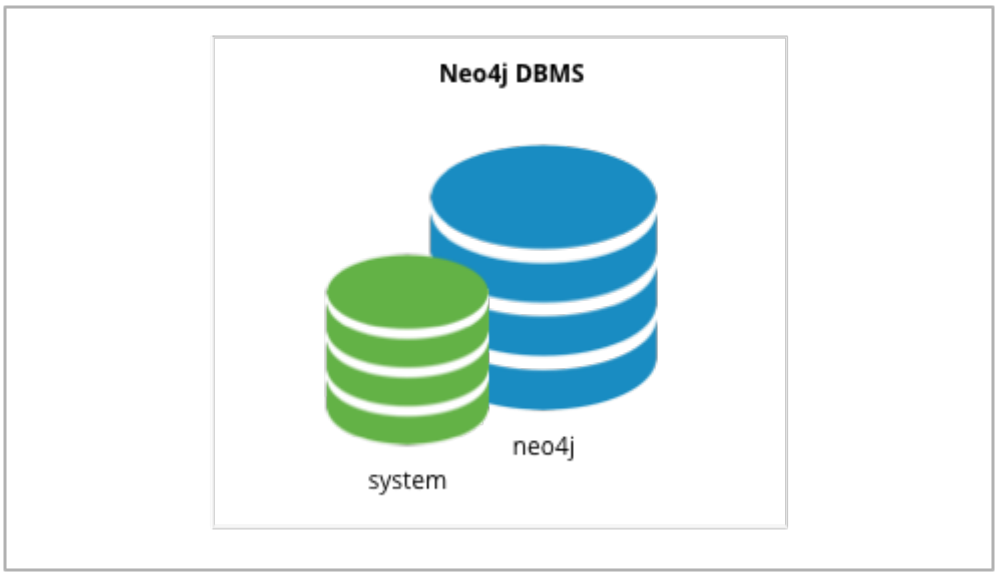


Figure 1. A default Neo4j installation.

Editions

The edition of Neo4j determines the number of possible databases:

- Installations of Community Edition can have exactly **one** user database.
- Installations of Enterprise Edition can have any number of user databases.

All installations include the `system` database.

6.1.2. The `system` database

All installations include a built-in database named `system`, which contains meta-data and security configuration.

The `system` database behaves differently than all other databases. In particular, when connected to this database you can only perform a specific set of administrative functions, as described in detail in Cypher Manual → Administration.

Most of the available administrative commands are restricted to users with specific administrative privileges. An example of configuring security privileges is described in Fine-grained access control. Security administration is described in detail in Cypher Manual → Security of administration .

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

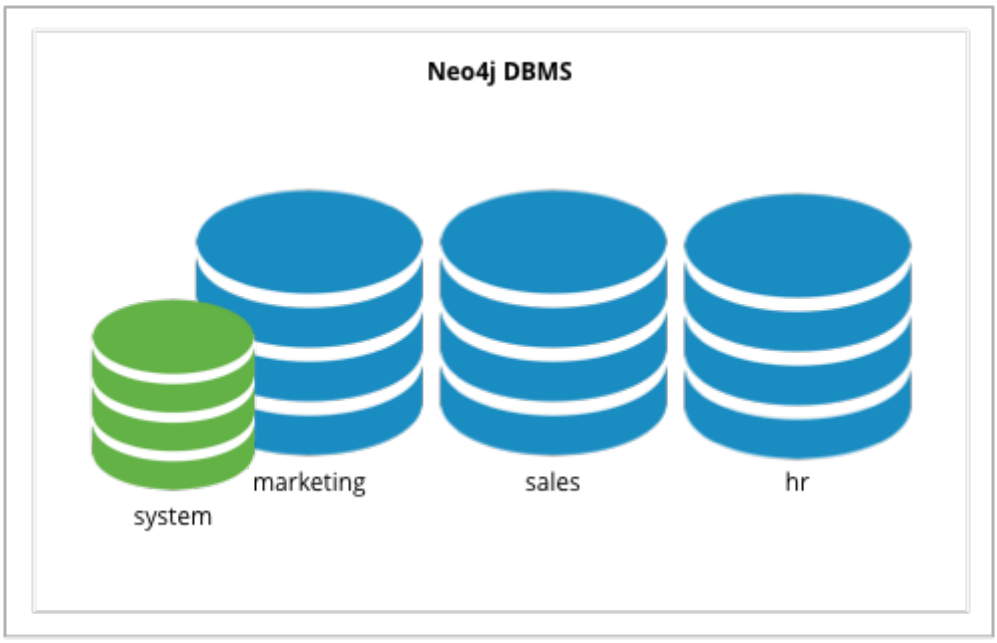


Figure 2. A multiple database Neo4j installation.

6.1.3. The default database

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

The default database is configurable. See configuration parameters for details.

The following image illustrates an installation of Neo4j containing the three databases for user data, named `marketing`, `sales` and `hr`, and the `system` database. The default database is `sales`:

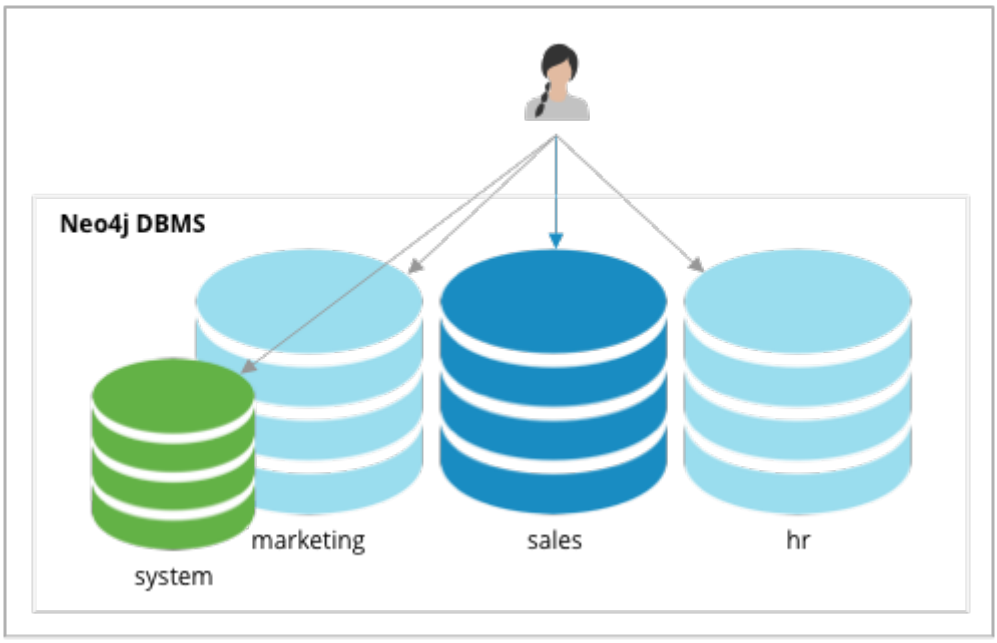


Figure 3. A multiple database Neo4j installation, with a default database.

6.2. Administration and configuration

6.2.1. Administrative commands

Administrative commands should not be used during a rolling upgrade. For more information, see Upgrade and Migration Guide → Upgrade a Causal Cluster.

For detailed information on Cypher administrative commands, see Cypher Manual → Administration.

Before using administrative commands, it is important to understand the difference between stopped databases, and dropped databases:

- Databases that are stopped with the `STOP` command are completely shutdown, and may be started again through the `START` command. In a Causal Cluster, as long as a database is in a shutdown state, it can not be considered available to other members of the cluster. It is not possible to do online backups against shutdown databases and they need to be taken into special consideration during disaster recovery, as they do not have a running Raft machine while shutdown.
- Dropped databases are completely removed and are not intended to be used again at all.

The following Cypher commands are used on the `system` database to manage multiple databases:

Command	Description
`CREATE DATABASE name`	Create and start a new database.
`DROP DATABASE name`	Drop (remove) an existing database.
`START DATABASE name`	Start a database that has been stopped.
`STOP DATABASE name`	Shut down a database.
`SHOW DATABASE name`	Show the status of a specific database.
`SHOW DATABASES`	Show the name and status of all the databases.
`SHOW DEFAULT DATABASE`	Show the name and status of the default database.

Naming rules for databases are as follows:

- Length must be between 3 and 63 characters.
- The first character of a name must be an ASCII alphabetic character.
- Subsequent characters must be ASCII alphabetic or numeric characters, dots or dashes; `[a..z][0..9].-`

- Names are case-insensitive and normalized to lowercase.
- Names that begin with an underscore and with the prefix `system` are reserved for internal use.

All of the above commands are executed as Cypher commands, and the database name is subject to the standard Cypher restrictions on valid identifiers. In particular, the `-` (dash) and `.` (dot) characters are not legal in Cypher variables, and therefore names with dashes must be enclosed within back-ticks. For example, `CREATE DATABASE \`main-db\``. Database names are the only identifier for which dots don't need to be escaped. For example, `main.db` is a valid database name.

For detailed information on Cypher administrative commands, see Cypher Manual → Administration.

For examples of using the Cypher administrative commands to manage multiple active databases, see Queries.

6.2.2. Configuration parameters

Configuration parameters are defined in the neo4j.conf file.

The following configuration parameters are applicable for managing databases:

Parameter name	Description	Default value
`dbms.default_database`	Name of the default database for the Neo4j instance. The database is created if it does not exist when the instance starts.	`neo4j`
`dbms.max_databases`	Maximum number of databases that can be used in a Neo4j single instance or Causal Cluster. The number includes all the online and offline databases. The value is an integer with a minimum value of 2. Note that once the limit has been reached, it is not possible to create any additional databases. Similarly, if the limit is changed to a number lower than the total number of existing databases, no additional databases can be created.	`100`

In a clustered setup, the value of `dbms.default_database` is only used to set the initial default database.

To change the default database at a later point, see Change the default database.

6.3. Queries

For detailed information on Cypher administrative commands, see Cypher Manual → Administration.

All commands and example queries in this section are run in the Neo4j Cypher Shell command-line interface (CLI).

Note that the `cypher-shell` queries are not case-sensitive, but must end with a semicolon.

6.3.1. Show the status of a specific database

Example 21. `SHOW DATABASE`

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

In standalone mode:

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

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

Or in a Causal Cluster:

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

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

6.3.2. Show the status of all databases

Example 22. `SHOW DATABASES`

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

In standalone mode:

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

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

Or in a Causal Cluster:

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

6 rows available after 5 ms, consumed after another 1 ms
```

Switching between `online` and `offline` states is achieved using the `START DATABASE` and `STOP DATABASE` commands.

6.3.3. Show the status of the default database

The config setting `dbms.default_database` defines which database is created and started by default when Neo4j starts. The default value of this setting is `neo4j`.

Example 23. `SHOW DEFAULT DATABASE`

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

In standalone mode:

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

1 row available after 57 ms, consumed after another 2 ms
```

Or in a Causal Cluster:

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

3 row available after 57 ms, consumed after another 2 ms
```

You can change the default database by using `dbms.default_database`, and restarting the server.

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

6.3.4. Create a database Enterprise edition

Example 24. `CREATE DATABASE`

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

```
0 rows available after 108 ms, consumed after another 0 ms
```

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

In standalone mode:

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

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

Or in a Causal Cluster:

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

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

6.3.5. Switch a database Enterprise edition

Example 25. `:use <database-name>`

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

6.3.6. Create or replace a database

Example 26. `CREATE OR REPLACE DATABASE`

```
neo4j@sales> match (n) return count(n) as countNode;
```

```
+-----------+
| countNode |
+-----------+
| 115       |
+-----------+

1 row available after 12 ms, consumed after another 0 ms
```

```
neo4j@system> CREATE OR REPLACE DATABASE sales;
```

```
0 rows available after 64 ms, consumed after another 0 ms
```

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

In standalone mode:

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

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

Or in a Causal Cluster:

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

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

```
neo4j@system> :use sales
neo4j@sales> match (n) return count(n) as countNode;
```

```
+-----------+
| countNode |
+-----------+
| 0         |
+-----------+

1 row available after 15 ms, consumed after another 1 ms
```

6.3.7. Stop a database

Example 27. `STOP DATABASE`

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

```
0 rows available after 18 ms, consumed after another 6 ms
```

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

In standalone mode:

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

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

Or in a Causal Cluster:

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

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

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

```
Unable to get a routing table for database 'sales' because this database is unavailable
neo4j@sales[UNAVAILABLE]>
```

6.3.8. Start a database

Example 28. START DATABASE

```
neo4j@sales[UNAVAILABLE]> :use system
neo4j@system> START DATABASE sales;
```

```
0 rows available after 21 ms, consumed after another 1 ms
```

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

In standalone mode:

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

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

Or in a Causal Cluster:

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

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

6.3.9. Drop or remove a database Enterprise edition

Example 29. `DROP DATABASE`

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

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

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

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

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

6.4. Error handling

When running the database management queries, such as `CREATE DATABASE`, it is possible to encounter errors.

6.4.1. Observing errors

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

Example 30. Fail to create a database

```
neo4j@system> CREATE DATABASE foo;
```

```
0 rows available after 108 ms, consumed after another 0 ms
```

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

In standalone mode:

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

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

In a Causal Cluster:

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

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

6.4.2. Database states

A database management operation may fail for a number of reasons. For example, if the file system instance has incorrect permissions, or Neo4j itself is misconfigured. As a result, the contents of the `error` column in the `SHOW DATABASE` query results may vary significantly.

However, databases may only be in one of a select number of states:

Current state	Description
`initial`	The database has not yet been created.

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

Most often, when a database management operation fails, Neo4j attempts to transition the database in question to the `offline` state. If the system is certain that no store files have yet been created, it transitions the database to `initial` instead. Similarly, if the system suspects that the store files underlying the database are invalid (incomplete, partially deleted, or corrupt), then it transitions the database to `dirty`.

Whilst `dropped` is a valid database state, it is only transiently observable, as database records are removed from `SHOW DATABASE` results once the `DROP` operation is complete.

6.4.3. Retrying failed operations

Database management operations may be safely retried in the event of failure. However, these retries are not guaranteed to succeed, and errors may persist through several attempts.

Example 31. Retry to start a database

```
neo4j@system> START DATABASE foo;
```

```
0 rows available after 108 ms, consumed after another 0 ms
```

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

```
+-----------------------------------------------------------------------------------------------------
---------+
| name   | address          | role         | requestedStatus | currentStatus | error                |
default |
+-----------------------------------------------------------------------------------------------------
---------+
| "foo"  | "localhost:7687" | "standalone" | "online"        | "offline"     | "Some error message" |
FALSE   |
+-----------------------------------------------------------------------------------------------------
---------+

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

After investigating and addressing the underlying issue, you can start the database again and verify that it is running properly:

```
neo4j@system> START DATABASE foo;
```

```
0 rows available after 108 ms, consumed after another 0 ms
```

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

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

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

If repeated retries of a command have no effect, or if a database is in a `dirty` state, you may drop and recreate the database, as detailed in Cypher manual → Administration.

When running `DROP DATABASE` as part of an error handling operation, you can also append `DUMP DATA` to the command. It produces a database dump that can be further examined and potentially repaired.

6.4.4. Using quarantine in a cluster for fixing errors

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

member due to a file system permissions issue with the volume where the database is located, or when a recently started database begins to log errors. The quarantine state renders the database inaccessible on that cluster member and prevents its state from being changed, for example, via the `START DATABASE` command. After lifting the quarantine, the cluster member tries to bring the database to the desired state.

It is recommended to run the quarantine procedure over the `bolt://` protocol rather than `neo4j://`, which may route requests to unexpected instances.

Syntax:

```
CALL dbms.cluster.quarantineDatabase(databaseName,setStatus,reason)
```

Arguments:

Name	Type	Description
`databaseName`	String	The name of the database that will be put into or removed from quarantine.
`setStatus`	Boolean	`true` for placing the database into quarantine; `false` for lifting the quarantine.
`reason`	String	(Optional) The reason for placing the database in quarantine.

Returns:

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

Quarantine a database

```
neo4j@system> CALL dbms.cluster.quarantineDatabase("foo",true);
```

```
+-----------------------------------------------------------------------------------+
| databaseName | quarantined | result                                               |
+-----------------------------------------------------------------------------------+
| "foo"        | TRUE        | "By neo4j at 2020-10-15T15:10:41.348Z: No reason given" |
+-----------------------------------------------------------------------------------+

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

Check if a database is quarantined

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

```
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------+
| name  | address          | role       | requestedStatus | currentStatus | error
| default |
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------+
| "foo" | "localhost:7688" | "unknown"  | "online"        | "quarantined" | "By neo4j at 2020-10-
15T15:10:41.348Z: No reason given" | FALSE   |
| "foo" | "localhost:7689" | "follower" | "online"        | "online"      | ""
| FALSE   |
| "foo" | "localhost:7687" | "leader"   | "online"        | "online"      | ""
| FALSE   |
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------+

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

6.5. Databases in a Causal Cluster

Multiple databases in a Causal Cluster are managed the same way as a single instance. Administrators can use the same Cypher commands described in Administrative commands to manage databases. This is based on two main principles:

- All databases are available on all members of a cluster - this applies to Core servers and Read Replicas.
- Administrative commands must be executed on the `system` database, on the Leader member of the cluster.

6.5.1. Change the default database

You can use the procedure `dbms.cluster.setDefaultDatabase("newDefaultDatabaseName")` to change the default database of a Causal cluster.

1. Ensure that the database to be set as default exists, otherwise create it using the command `CREATE DATABASE <database-name>`.
2. Show the name and status of the current default database by using the command `SHOW DEFAULT DATABASE`.
3. Stop the current default database using the command `STOP DATABASE <database-name>`.
4. On the Leader member of the cluster, run `CALL dbms.cluster.setDefaultDatabase("newDefaultDatabaseName")` against the `system` database to set the new default database.
5. On each cluster member:
    a. Open the *<neo4j-home>/conf/neo4j.conf* file and update the value of `dbms.default_database` to the new default database.
    b. Restart the instance.
6. Optionally, you can start the previous default database as non-default by using `START DATABASE <database-name>`.

6.5.2. Run Cypher administrative commands from Cypher Shell on a Causal Cluster

For the following examples consider a Causal Cluster environment formed by 5 members, 3 Core servers, and 2 Read Replicas:

Example 32. View the members of a Causal Cluster

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

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

5 rows available after 5 ms, consumed after another 0 ms
```

The leader is currently the instance exposing port `7681` for the `bolt` protocol, and `7471/7481` for the `http/https` protocol.

Administrators can connect and execute Cypher commands in the following ways:

Example 33. Using the `bolt://` scheme to connect to the Leader:

```
$ bin/cypher-shell -a bolt://localhost:7681 -d system -u neo4j -p neo4j1
```

```
Connected to Neo4j 4.0.0 at bolt://localhost:7681 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
```

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

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

2 rows available after 34 ms, consumed after another 0 ms
```

```
neo4j@system> CREATE DATABASE data001;
```

```
0 rows available after 378 ms, consumed after another 12 ms
Added 1 nodes, Set 4 properties, Added 1 labels
neo4j@system> SHOW DATABASES;
+-------------------------------+
| name      | status   | default |
+-------------------------------+
| "neo4j"   | "online" | TRUE    |
| "system"  | "online" | FALSE   |
| "data001" | "online" | FALSE   |
+-------------------------------+

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

Example 34. Using the neo4j:// scheme to connect to any Core member:

```
$ bin/cypher-shell -a neo4j://localhost:7683 -d system -u neo4j -p neo4j1
```

```
Connected to Neo4j 4.0.0 at neo4j://localhost:7683 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
```

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

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

3 rows available after 0 ms, consumed after another 0 ms
```

```
neo4j@system> CREATE DATABASE data002;
```

```
0 rows available after 8 ms, consumed after another 1 ms
Added 1 nodes, Set 4 properties, Added 1 labels
```

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

```
+--------------------------------+
| name      | status   | default |
+--------------------------------+
| "neo4j"   | "online" | TRUE    |
| "system"  | "online" | FALSE   |
| "data001" | "online" | FALSE   |
| "data002" | "online" | FALSE   |
+--------------------------------+

4 rows available after 33 ms, consumed after another 0 ms
```

The neo4j:// scheme is the equivalent to the bolt+routing: scheme available in earlier versions of Neo4j, but it can be used seamlessly with a standalone and clustered DBMS.

Chapter 7. Clustering

This chapter describes the following:

- Introduction — An overview of the Causal Clustering architecture.
- Deploy a cluster — The basics of configuring and deploying a new cluster.
- Seed a cluster — How to deploy a Causal Cluster with pre-existing data.
- Discovery — How members of a cluster discover each other.
- Intra-cluster encryption — How to secure the cluster communication.
- Internals — A few internals regarding the operation of the cluster.
- Settings reference — A summary of the most important Causal Cluster settings.

Further information:

- For instructions on setting up Causal Clustering when running Neo4j in a Docker container, see Causal Clustering on Docker.
- For an example of managing multiple databases in a Causal Cluster, see Multiple databases in a Causal Cluster.
- For instructions on how to upgrade your Neo4j Causal Cluster, see Upgrade and Migration Guide → Upgrade a Causal Cluster.
- For a summary of the facilities that are available for monitoring a Neo4j Causal Cluster, see Monitoring (and specifically, Monitoring a Causal Cluster).
- For a tutorial on setting up a test cluster locally on a single machine, see Set up a local Causal Cluster.
- For advanced concepts, including the implementation of the Raft Protocol, see Advanced Causal Clustering

7.1. Introduction

7.1.1. Overview

Neo4j's Causal Clustering provides three main features:

1. **Safety:** Core Servers provide a fault tolerant platform for transaction processing which will remain available while a simple majority of those Core Servers are functioning.
2. **Scale:** Read Replicas provide a massively scalable platform for graph queries that enables very large graph workloads to be executed in a widely distributed topology.
3. **Causal consistency:** when invoked, a client application is guaranteed to read at least its own writes.

Together, this allows the end-user system to be fully functional and both read and write to the database in the event of multiple hardware and network failures and makes reasoning about database interactions straightforward.

In the remainder of this section we will provide an overview of how causal clustering works in production,

including both operational and application aspects.

7.1.2. Operational view

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

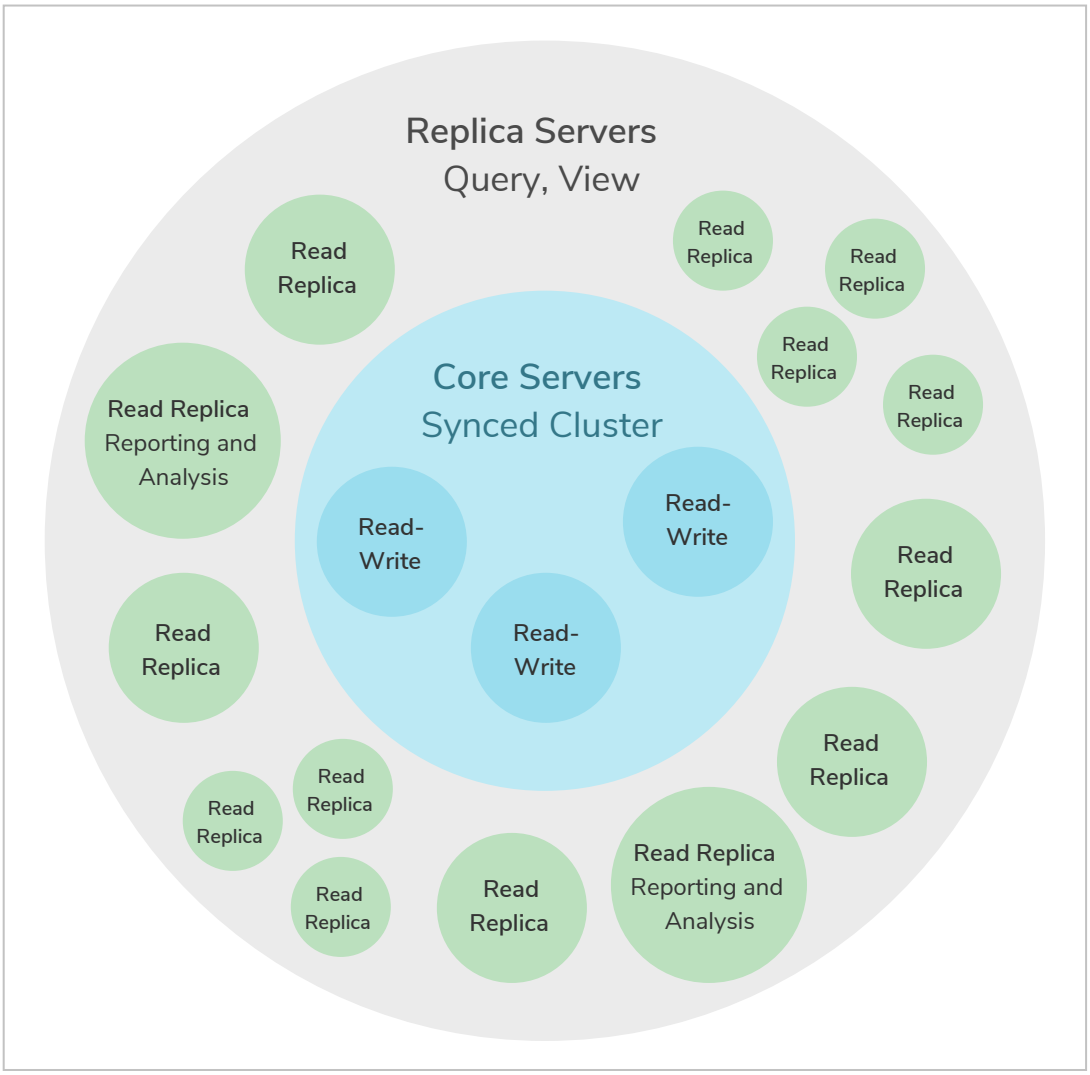


Figure 4. Causal Cluster Architecture

The two roles are foundational in any production deployment but are managed at different scales from one another and undertake different roles in managing the fault tolerance and scalability of the overall cluster.

Core Servers

The main responsibility of Core Servers is to safeguard data. Core Servers do so by replicating all transactions using the Raft protocol. Raft ensures that the data is safely durable before confirming transaction commit to the end user application. In practice this means once a majority of Core Servers in a cluster (N/2+1) have accepted the transaction, it is safe to acknowledge the commit to the end user application.

The safety requirement has an impact on write latency. Implicitly writes will be acknowledged by the fastest majority, but as the number of Core Servers in the cluster grows so do the size of the majority

needed to acknowledge a write.

In practice this means that there are relatively few machines in a typical Core Server cluster, enough to provide sufficient fault tolerance for the specific deployment. This is calculated with the formula `M = 2F + 1` where `M` is the number of Core Servers required to tolerate `F` faults. For example:

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

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

Read Replicas

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

Read Replicas are asynchronously replicated from Core Servers via transaction log shipping. They will periodically poll an upstream server for new transactions and have these shipped over. Many Read Replicas can be fed data from a relatively small number of Core Servers, allowing for a large fan out of the query workload for scale.

Read Replicas should typically be run in relatively large numbers and treated as disposable. Losing a Read Replica does not impact the cluster's availability, aside from the loss of its fraction of graph query throughput. It does not affect the fault tolerance capabilities of the cluster.

7.1.3. Causal consistency

While the operational mechanics of the cluster are interesting from an application point of view, it is also helpful to think about how applications will use the database to get their work done. In an application we typically want to read from the graph and write to the graph. Depending on the nature of the workload we usually want reads from the graph to take into account previous writes to ensure causal consistency.

Causal consistency is one of numerous consistency models used in distributed computing. It ensures that causally related operations are seen by every instance in the system in the same order. Consequently, client applications are guaranteed to read their own writes, regardless of which instance they communicate with. This simplifies interaction with large clusters, allowing clients to treat them as a single (logical) server.

Causal consistency makes it possible to write to Core Servers (where data is safe) and read those writes from a Read Replica (where graph operations are scaled out). For example, causal consistency guarantees that the write which created a user account will be present when that same user subsequently attempts to log in.

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

On executing a transaction, the client can ask for a bookmark which it then presents as a parameter to subsequent transactions. Using that bookmark the cluster can ensure that only servers which have processed the client's bookmarked transaction will run its next transaction. This provides a *causal chain* which ensures correct read-after-write semantics from the client's point of view.

Aside from the bookmark everything else is handled by the cluster. The database drivers work with the cluster topology manager to choose the most appropriate Core Servers and Read Replicas to provide high quality of service.

7.1.4. Summary

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

7.2. Deploy a cluster

7.2.1. Introduction

This section describes how to set up a new Causal Cluster consisting of three Core instances. It includes how to add more Core Servers as well as Read Replicas to a running cluster. Additionally, it also describes how to detach a Read Replica from a cluster and turn it into a standalone instance.

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

Refer to Set up a local Causal Cluster for a tutorial on how to set up a Causal Cluster on a local machine.

7.2.2. Configure a Core-only cluster

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

Table 29. Important settings for a new Causal Cluster

Option name	Description
`dbms.default_listen_address`	The address or network interface this machine uses to listen for incoming messages. Setting this value to `0.0.0.0` makes Neo4j bind to all available network interfaces.
`dbms.default_advertised_address`	The address that other machines are told to connect to. In the typical case, this should be set to the fully qualified domain name or the IP address of this server.
`dbms.mode`	The operating mode of a single server instance. For Causal Clustering, there are two possible modes: `CORE` or `READ_REPLICA`.
`causal_clustering.minimum_core_cluster_size_at_formation`	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.
`causal_clustering.minimum_core_cluster_size_at_runtime`	The minimum number of Core instances which will exist in the consensus group.
`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. In the default case, the initial discovery members are given as a comma-separated list of address/port pairs, and the default port for the discovery service is `:5000`. It is good practice to set this parameter to the same value on all Core Servers. The behavior of this setting can be modified by configuring the setting `causal_clustering.discovery_type`. This is described in detail in Discovery.

Listen configuration

Listening on 0.0.0.0 makes the ports publicly available. Make sure you understand the security implications and strongly consider setting up encryption.

The following example shows how to set up a simple cluster with three Core servers:

Example 35. Configure a Core-only cluster

In this example, three Core Servers named `core01.example.com`, `core02.example.com` and `core03.example.com` are configured. Neo4j Enterprise Edition is already installed on all three servers. They are configured by preparing *neo4j.conf* on each server. Note that they are all identical, except for the configuration of `dbms.default_advertised_address`:

neo4j.conf on core01.example.com:

```
dbms.default_listen_address=0.0.0.0
dbms.default_advertised_address=core01.example.com
dbms.mode=CORE
causal_clustering.initial_discovery_members=core01.example.com:5000,core02.example.com:5000,core03.example.com:5000
```

neo4j.conf on core02.example.com:

```
dbms.default_listen_address=0.0.0.0
dbms.default_advertised_address=core02.example.com
dbms.mode=CORE
causal_clustering.initial_discovery_members=core01.example.com:5000,core02.example.com:5000,core03.example.com:5000
```

neo4j.conf on core03.example.com:

```
dbms.default_listen_address=0.0.0.0
dbms.default_advertised_address=core03.example.com
dbms.mode=CORE
causal_clustering.initial_discovery_members=core01.example.com:5000,core02.example.com:5000,core03.example.com:5000
```

Now the Neo4j servers are ready to be started. The startup order does not matter.

After the cluster has started, it is possible to connect to any of the instances and run `CALL dbms.cluster.overview()` to check the status of the cluster. This shows information about each member of the cluster.

A Neo4j Causal Cluster of three instances is now running.

Startup time

The instance may appear unavailable while it is joining the cluster. If you want to follow along with the startup, you can follow the messages in *neo4j.log*.

7.2.3. Add a Core Server to an existing cluster

Core Servers are added to an existing cluster by starting a new Neo4j instance with the appropriate

configuration. The new server joins the existing cluster and become available once it has copied the data from its peers. It may take some time for the new instance to perform the copy if the existing cluster contains large amounts of data.

The setting `causal_clustering.initial_discovery_members` needs to be updated on all the servers in the cluster to include the new server.

Example 36. Add a Core Server to an existing cluster

In this example, a Core Server, `core04.example.com`, is added to the cluster created in Configure a Core-only cluster.

Configure the following entries in *neo4j.conf*:

neo4j.conf on core04.example.com:

```
dbms.default_listen_address=0.0.0.0
dbms.default_advertised_address=core04.example.com
dbms.mode=CORE
causal_clustering.minimum_core_cluster_size_at_formation=3
causal_clustering.minimum_core_cluster_size_at_runtime=3
causal_clustering.initial_discovery_members=core01.example.com:5000,core02.example.com:5000,core03.example.com:5000,core04.example.com:5000
```

Note that the configuration is very similar to that of the previous servers. In this example, the new server is not intended to be a permanent member of the cluster, thus it is not included in `causal_clustering.initial_discovery_members`.

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

7.2.4. Add a Read Replica to an existing cluster

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

Example 37. Add a Read Replica to an existing cluster

In this example, a Read Replica, `replica01.example.com`, is added to the cluster created in Configure a Core-only cluster.

Configure the following entries in *neo4j.conf*:

neo4j.conf on replica01.example.com:

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

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

7.2.5. Detach a Read Replica from an existing cluster

It is possible to turn a Read Replica into a standalone instance that thus contains a snapshot of the data in the cluster. This can, in theory, be done for a Core Server as well, but this is **not** recommended for performance and safety reasons.

Example 38. Detach a Read Replica and turn it into a stand alone instance

In this example, a Read Replica, `replica01.example.com`, is detached from a cluster. See Add a Read Replica to an existing cluster above on how to add a Read Replica to a cluster.

First, ensure that the Read Replica is up-to-date, then shut it down. Once the Read Replica is shut down, configure the following entry in *neo4j.conf*:

neo4j.conf on replica01.example.com:

```
dbms.mode=SINGLE
```

Start the instance again. It is now a standalone instance containing the same data as the cluster (at the time of shutting down the Read Replica).

There is always a chance that the Read Replica is behind the Core Servers at any time. If a transaction is being processed at the time of the shutdown of the Read Replica, this transaction is eventually reflected in the remaining Cluster, but not on the detached Read Replica. A way to ensure that a Read Replica contains a snapshot of a database in the cluster at a point in time, is to pause the read Replica before shutting it down. See `dbms.cluster.readReplicaToggle()` for more information.

7.3. Seed a cluster

7.3.1. Introduction

Regardless of whether you are just playing around with Neo4j or setting up a production environment, you likely have some existing data that you want to transfer into your newly created Causal cluster. Neo4j supports seeding a cluster from a database dump, a database backup, or from another data source (with the Import tool). For more information about the different backup options and how to use the Neo4j Import tool, see Backup and restore options and Neo4j Admin.

The databases that you want to seed and the Neo4j cluster must be of the same version.

7.3.2. Seed a cluster from a database dump (offline)

If you have an existing Neo4j database that you want to use for your new cluster, use `neo4j-admin dump` to create an offline backup. This could be an offline backup from a standalone Neo4j instance or a cluster member (e.g., an existing Read Replica).

This scenario is useful in disaster recovery where some servers have retained their data during a catastrophic event.

Moving files and directories manually in or out of a Neo4j installation is not recommended and considered unsupported.

1. Create a new Neo4j Core-only cluster following the instructions in Configure a Core-only cluster.
2. Delete the databases whose name conflicts with the ones in your seeds by using the Cypher command `DROP DATABASE <database-name` against the `system` database.
   The command is automatically routed to the leader and from there to the other cluster members.

Dropping a database also deletes the users and roles associated with it.

If you cannot delete the database because Neo4j is not running or because your seeds include the `system` database (which cannot be dropped), you must run `neo4j-admin unbind`.

1. Run `neo4j-admin unbind` to turn the cluster members into standalone instances by removing their cluster state. Otherwise, the store files you have (post restore) will be out of sync with the cluster state you have for that database, leading to logical corruption.
2. Remove the store and transaction log files for the database in question. The locations of these files may be configured.

3. Stop each cluster member.
4. Use `neo4j-admin load` to seed each of the Core members in the cluster.

   The examples assume that you are restoring one user database with the default name of `neo4j` and the `system` database, containing the replicated configuration state. Modify the command line arguments to match your exact setup.

```
neo4j-01$ ./bin/neo4j-admin load --from=/path/to/system.dump --database=system
neo4j-01$ ./bin/neo4j-admin load --from=/path/to/neo4j.dump --database=neo4j
```

```
neo4j-02$ ./bin/neo4j-admin load --from=/path/to/system.dump --database=system
neo4j-02$ ./bin/neo4j-admin load --from=/path/to/neo4j.dump --database=neo4j
```

```
neo4j-03$ ./bin/neo4j-admin load --from=/path/to/system.dump --database=system
neo4j-03$ ./bin/neo4j-admin load --from=/path/to/neo4j.dump --database=neo4j
```

5. Start each cluster member.

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

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

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

The cluster forms and the replicated Neo4j DBMS deployment comes online.

7.3.3. Seed a cluster from a database backup (online)

If you have a running Neo4j database that you want to seed in a running cluster, use `neo4j-admin backup` to create a database backup. This could be a backup from a standalone Neo4j instance or another cluster member (e.g., an existing Read Replica).

You transfer that database backup to each core member and then use the `CREATE DATABASE` Cypher command to seed the cluster. For more information on the `CREATE DATABASE` syntax and options, see Cypher Manual → Creating databases.

These scenarios are useful when you want to restore a database in a running cluster.

Moving files and directories manually in or out of a Neo4j installation is not recommended and considered unsupported.

This example uses a user database called `movies1`.

1. To be sure that the `movies1` database does not exist in the cluster, on one of the core members, use Cypher Shell and run `DROP DATABASE movies1`. Use the `system` database to connect. The command is automatically routed to the leader and from there to the other cluster members.

```
DROP DATABASE movies1;
```

Dropping a database also deletes the users and roles associated with it.

If you cannot delete the database because Neo4j is not running or because your seeds include the `system` database (which cannot be dropped), you must run `neo4j-admin unbind`.

1. Run `neo4j-admin unbind` to turn the cluster members into standalone instances by removing their cluster state. Otherwise, the store files you have (post restore) will be out of sync with the cluster state you have for that database, leading to logical corruption.
2. Remove the store and transaction log files for the database in question. The locations of these files may be configured.

2. Restore the database in each core member in the cluster.

```
neo4j@core1$ ./bin/neo4j-admin restore --from=/path/to/movies1-backup-dir --database=movies1
```

```
neo4j@core2$ ./bin/neo4j-admin restore --from=/path/to/movies1-backup-dir --database=movies1
```

```
neo4j@core3$ ./bin/neo4j-admin restore --from=/path/to/movies1-backup-dir --database=movies1
```

However, restoring a database does not automatically create it.

3. On one of the cluster members, run `CREATE DATABASE movies1` against the `system` database to create the `movies1` database. The command is automatically routed to the leader and from there to the other cluster members.

```
CREATE DATABASE movies1;
```

```
0 rows
ready to start consuming query after 701 ms, results consumed after another 0 ms
```

4. Verify that the `movies1` database is online on all members.

```
SHOW DATABASES;
```

```
+-------------------------------------------------------------------------------------------+
| name     | address      | role       | requestedStatus | currentStatus | error | default |
+-------------------------------------------------------------------------------------------+
| "neo4j"  | "core1:7687" | "leader"   | "online"        | "online"      | ""    | TRUE    |
| "neo4j"  | "core3:7687" | "follower" | "online"        | "online"      | ""    | TRUE    |
| "neo4j"  | "core2:7687" | "follower" | "online"        | "online"      | ""    | TRUE    |
| "movies1"| "core1:7687" | "leader"   | "online"        | "online"      | ""    | FALSE   |
| "movies1"| "core3:7687" | "follower" | "online"        | "online"      | ""    | FALSE   |
| "movies1"| "core2:7687" | "follower" | "online"        | "online"      | ""    | FALSE   |
| "system" | "core1:7687" | "follower" | "online"        | "online"      | ""    | FALSE   |
| "system" | "core3:7687" | "follower" | "online"        | "online"      | ""    | FALSE   |
| "system" | "core2:7687" | "leader"   | "online"        | "online"      | ""    | FALSE   |
+-------------------------------------------------------------------------------------------+

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

7.3.4. Seed a cluster using the import tool

To create a cluster based on imported data, it is recommended to first import the data into a standalone Neo4j DBMS and then use an offline backup to seed the cluster.

1. Import the data.
   a. Deploy a standalone Neo4j DBMS.
   b. Import the data using the import tool.
2. Use `neo4j-admin dump` to create an offline backup of the `neo4j` database.
3. Seed a new cluster using the instructions in Seed a cluster from a database dump (offline).

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

7.4. Discovery

7.4.1. Overview

In order to form or connect to a running cluster, a Core Server or a Read Replica needs to know the addresses of some of the Core Servers. This information is used to bind to the Core Servers in order to run the discovery protocol and get the full information about the cluster. The way in which this is best done depends on the configuration in each specific case.

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

- If Core members are replaced and the new members have different addresses, the list will become outdated. An outdated list can be avoided by ensuring that the new members can be reached via the same address as the old members, but this is not always practical.
- Under some circumstances the addresses are unknown when configuring the cluster. This can be the case, for example, when using container orchestration such as Kubernetes to deploy a Causal Cluster.

Additional mechanisms for using DNS are provided for the cases where it is not practical or possible to explicitly list the addresses of cluster members to discover.

The discovery configuration is just used for initial discovery and a running cluster will continuously exchange information about changes to the topology. The behavior of the initial discovery is determined by the parameters `causal_clustering.discovery_type` and `causal_clustering.initial_discovery_members`, and is described in the following sections.

Discovery using a list of server addresses

If the addresses of the other cluster members are known upfront, they can be listed explicitly. We use the default `causal_clustering.discovery_type=LIST` and hard code the addresses in the configuration of each machine. This alternative is illustrated by Configure a Core-only cluster.

Discovery using DNS with multiple records

When using initial discovery with DNS, a DNS record lookup is performed when an instance starts up. Once an instance has joined a cluster, further membership changes are communicated amongst Core members as part of the discovery service.

The following DNS-based mechanisms can be used to get the addresses of Core Cluster members for discovery:

`causal_clustering.discovery_type=DNS`

With this configuration, the initial discovery members will be resolved from *DNS A records* to find the IP addresses to contact. The value of `causal_clustering.initial_discovery_members` should be set to a single domain name and the port of the discovery service. For example: `causal_clustering.initial_discovery_members=cluster01.example.com:5000`. The domain name should return an A record for every Core member when a DNS lookup is performed. Each A record returned by DNS should contain the IP address of the Core Server. The configured Core Server will use all the IP addresses from the A records to join or form a cluster.

The discovery port must be the same on all Cores when using this configuration. If this is not possible,

consider using the discovery type `SRV` instead.

`causal_clustering.discovery_type=SRV`

With this configuration, the initial discovery members will be resolved from *DNS SRV* records to find the IP addresses/hostnames and discovery service ports to contact. The value of `causal_clustering.initial_discovery_members` should be set to a single domain name and the port set to `0`. For example: `causal_clustering.initial_discovery_members=cluster01.example.com:0`. The domain name should return a single SRV record when a DNS lookup is performed. The SRV record returned by DNS should contain the IP address or hostname, and the discovery port, for the Core Servers to be discovered. The configured Core Server will use all the addresses from the SRV record to join or form a cluster.

Discovery in Kubernetes

When running a Causal Cluster in Kubernetes, each Core Server should be reachable via a separate Kubernetes service. In that case the addresses of the Core Cluster members do not need to be configured in advance and can be obtained by neo4j at runtime using the Kubernetes List Service API, as described in the Kubernetes API documentation.

To use the Kubernetes API to find the Core Cluster members the following configurations must be set in the *neo4j.conf* file of each cluster member:

- `causal_clustering.discovery_type=K8S`.
- `causal_clustering.kubernetes.label_selector` - to the label selector for the Causal Cluster services. For more information, see the Kubernetes official documentation.
- `causal_clustering.kubernetes.service_port_name` - to the name of the service port used in the Kubernetes service definition for the Core's discovery port. For more information, see the Kubernetes official documentation

- The pod running Neo4j must use a service account that has the permission to list services. For further information, see the Kubernetes documentation on RBAC authorization or ABAC authorization.
- As with DNS-based methods, the Kubernetes record lookup is only performed at startup.
- 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-domain>`, e.g., `my-neo4j-1.default.svc.cluster.local`.
- When the `discovery_type` is set to `K8S`, `causal_clustering.initial_discovery_members` is not used, and any value assigned to it will be ignored.

There are some additional settings that can be used to further configure the Kubernetes-based discovery mechanism. However, you do not normally or very rarely need to set them:

- The Kubernetes cluster domain

- `config_causal_clustering.kubernetes.cluster_domain` - if unset, the cluster domain `cluster.local` is used.

- The Kubernetes api address
  - `config_causal_clustering.kubernetes.address` - if unset the address `kubernetes.default.svc:443` is used.

- And the following configurations, which take file locations as values. Neo4j reads these files at runtime and uses the values that they contain.

By default, Neo4j uses the files that Kubernetes provides automatically. They are located in the `/var/run/secrets/kubernetes.io/serviceaccount` directory.

  - `config_causal_clustering.kubernetes.namespace`
  - `config_causal_clustering.kubernetes.ca_crt`
  - `config_causal_clustering.kubernetes.token`

7.5. Intra-cluster encryption

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

7.5.1. Introduction

The security solution for cluster communication is based on standard SSL/TLS technology (referred to jointly as SSL). Encryption is in fact just one aspect of security, with the other cornerstones being authentication and integrity. A secure solution will be based on a key infrastructure which is deployed together with a requirement of authentication.

The SSL support in the platform is documented in detail in SSL framework. This section will cover the specifics as they relate to securing a cluster.

Under SSL, an endpoint can authenticate itself using certificates managed by a Public Key Infrastructure (PKI).

It should be noted that the deployment of a secure key management infrastructure is beyond the scope of this manual, and should be entrusted to experienced security professionals. The example deployment illustrated below is for reference purposes only.

7.5.2. Example deployment

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

- Generate and install cryptographic objects
- Configure Causal Clustering with the SSL policy
- Validate the secure operation of the cluster

Generate and install cryptographic objects

The generation of cryptographic objects is for the most part outside the scope of this manual. It will generally require having a PKI with a Certificate Authority (CA) within the organization and they should be able to advise here. Please note that the information in this manual relating to the PKI is mainly for illustrative purposes.

When the certificates and private keys have been obtained they can be installed on each of the servers. Each server will have a certificate of its own, signed by a CA, and the corresponding private key. The certificate of the CA is installed into the `trusted` directory, and any certificate signed by the CA will thus be trusted. This means that the server now has the capability of establishing trust with other servers.

Please be sure to exercise caution when using CA certificates in the `trusted` directory, as any certificates signed by that CA will then be trusted to join the cluster. For this reason, never use a public CA to sign certificates for your cluster. Instead, use an intermediate certificate or a CA certificate which originates from and is controlled by your organization.

In this example we will deploy a mutual authentication setup, which means that both ends of a channel have to authenticate. To enable mutual authentication the SSL policy must have `client_auth` set to `REQUIRE` (which is the default). Servers are by default required to authenticate themselves, so there is no corresponding server setting.

If the certificate for a particular server is compromised it is possible to revoke it by installing a Certificate Revocation List (CRL) in the `revoked` directory. It is also possible to redeploy using a new CA. For contingency purposes, it is advised that you have a separate intermediate CA specifically for the cluster which can be substituted in its entirety should it ever become necessary. This approach would be much easier than having to handle revocations and ensuring their propagation.

Example 39. Generate and install cryptographic objects

In this example we assume that the private key and certificate file are named *private.key* and *public.crt*, respectively. If you want to use different names you may override the policy configuration for the key and certificate names/locations. We want to use the default configuration for this server so we create the appropriate directory structure and install the certificate:

```
$neo4j-home> mkdir certificates/cluster
$neo4j-home> mkdir certificates/cluster/trusted
$neo4j-home> mkdir certificates/cluster/revoked

$neo4j-home> cp $some-dir/private.key certificates/cluster
$neo4j-home> cp $some-dir/public.crt certificates/cluster
```

Configure the cluster SSL policy

By default, cluster communication is unencrypted. To configure a Causal Cluster to encrypt its intra-cluster communication, set `dbms.ssl.policy.cluster.enabled` to `true`.

An SSL policy utilizes the installed cryptographic objects and additionally allows parameters to be

configured. We will use the following parameters in our configuration:

Table 30. Example settings

Setting suffix	Value	Comment
`client_auth`	`REQUIRE`	Setting this to `REQUIRE` effectively enables mutual authentication for servers.
`ciphers`	`TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384`	We can enforce a particular single strong cipher and remove any doubt about which cipher gets negotiated and chosen. The cipher chosen above offers Perfect Forward Secrecy (PFS) which is generally desirable. It also uses Advanced Encryption Standard (AES) for symmetric encryption which has great support for acceleration in hardware and thus allows performance to generally be negligibly affected.
`tls_versions`	`TLSv1.2`	Since we control the entire cluster we can enforce the latest TLS standard without any concern for backwards compatibility. It has no known security vulnerabilities and uses the most modern algorithms for key exchanges, etc.

In the following example we will create and configure an SSL policy that we will use in our cluster.

Example 40. Configure the cluster SSL policy

In this example we assume that the directory structure has been created, and certificate files have been installed, as per the previous example.

We add the following content to our *neo4j.conf* file:

```
dbms.ssl.policy.cluster.enabled=true
dbms.ssl.policy.cluster.tls_versions=TLSv1.2
dbms.ssl.policy.cluster.ciphers=TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
dbms.ssl.policy.cluster.client_auth=REQUIRE
```

Any user data communicated between instances will now be secured. Please note that an instance which is not correctly setup would not be able to communicate with the others.

Note that the policy must be configured on every server with the same settings. The actual cryptographic objects installed will be mostly different since they do not share the same private keys and corresponding certificates. The trusted CA certificate will be shared however.

Validate the secure operation of the cluster

To make sure that everything is secured as intended it makes sense to validate using external tooling such as, for example, the open source assessment tools `nmap` or `OpenSSL`.

Example 41. Validate the secure operation of the cluster

In this example we will use the `nmap` tool to validate the secure operation of our cluster. A simple test to perform is a cipher enumeration using the following command:

```
nmap --script ssl-enum-ciphers -p <port> <hostname>
```

The hostname and port have to be adjusted according to our configuration. This can prove that TLS is in fact enabled and that the only the intended cipher suites are enabled. All servers and all applicable ports should be tested.

For testing purposes we could also attempt to utilize a separate testing instance of Neo4j which, for example, has an untrusted certificate in place. The expected result of this test is that the test server is not able to participate in replication of user data. The debug logs will generally indicate an issue by printing an SSL or certificate-related exception.

7.6. Internals of clustering

7.6.1. Elections and leadership

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

If a follower has not heard from the leader for a while, then it can initiate an election and attempt to become the new leader. The follower makes itself a *Candidate* and asks other Cores to vote for it. If it can get a majority of the votes, then it assumes the leader role. Cores will not vote for a candidate which is less up-to-date than itself. There can only be one leader at any time per database, and that leader is guaranteed to have the most up-to-date log.

It is expected for elections to occur during the normal running of a cluster and they do not pose an issue in and of itself. If you are experiencing frequent re-elections and they are disturbing the operation of the cluster then you should try to figure out what is causing them. Some common causes are environmental issues (e.g. a flaky networking) and work overload conditions (e.g. more concurrent queries and transactions than the hardware can handle).

7.6.2. Leadership balancing

Write transactions will always be routed to the leader for the respective database. As a result, unevenly distributed leaderships may cause write queries to be disproportionately directed to a subset of instances. By default, Neo4j avoids this by automatically transferring database leaderships so that they are evenly distributed throughout the cluster.

7.6.3. Multi-database and the reconciler

Databases operate as independent entities in a Neo4j DBMS, both in standalone and in a cluster. Since a cluster consists of multiple independent server instances, the effects of administrative operations like creating a new database happen asynchronously and independently for each server. However, the immediate effect of an administrative operation is to safely commit the desired state in the system database.

The desired state committed in the system database gets replicated and is picked up by an internal component called the reconciler. It runs on every instance and takes the appropriate actions required locally on that instance for reaching the desired state; creating, starting, stopping and dropping databases.

Every database runs in an independent Raft group and since there are two databases in a fresh cluster, `system` and `neo4j`, this means that it also has two Raft groups. Every Raft group also has an independent leader and thus a particular Core server could be the leader for one database and a follower for another.

7.6.4. Server-side routing

Server-side routing is a complement to the client-side routing, performed by a Neo4j Driver.

In a Causal Cluster deployment of Neo4j, Cypher queries may be directed to a cluster member that is unable to run the given query. With server-side routing enabled, such queries will be rerouted internally to a cluster member that is expected to be able to run it. This situation can occur for write-transaction queries, when they address a database for which the receiving cluster member is not the leader.

The cluster role for core cluster members is per database. Thus, if a write-transaction query is sent to a cluster member which is not the leader for the specified database (specified either via the Bolt Protocol or by the Cypher syntax; `USE` clause), server-side routing will be performed, if properly configured.

Server-side routing is enabled by the DBMS, by setting `dbms.routing.enabled=true` for each cluster member. The listen address (`dbms.routing.listen_address`) and advertised address (`dbms.routing.advertised_address`) also need to be configured for server-side routing communication.

Client connections need to state that server-side routing should be used and this is only available for Neo4j Drivers that use the Bolt Protocol.

In order to determine which cluster member to query, Neo4j Drivers will query the cluster for routing information, using `dbms.cluster.routing.getRoutingTable()`, when needed. When you use the `neo4j://` URI scheme, then the Neo4j Drivers will do this using client-side routing.

However, when using the `neo4j://` URI scheme, the Neo4j Drivers will also state that server-side routing should be used.

With the exception of the Python Driver, if you use the `bolt://` URI scheme, the Neo4j Drivers will **not** state that server-side routing should be used.

The specification for setting the flag, that states whether server-side routing should be used, was introduced in Bolt Protocol 4.1 - server-side routing.

When using the `neo4j://` URI scheme, a Neo4j Driver will:

1. Query the cluster for routing information.
2. Use the routing table (client-side routing) to send a query.
3. State that server-side routing is allowed to be used.

Provided that server-side routing has been configured for the cluster, when the Neo4j Driver sends a write-transaction query to a non-leader cluster member, then that cluster member will try to route it correctly to the cluster member that is the leader.

The table below show the criteria, when server-side routing will be performed:

Table 31. Server-side routing criteria

CLIENT - Neo4j Driver (Bolt Protocol)				SERVER - Neo4j Cluster member		
URI scheme	Client-side routing	Request server-side routing	Transaction type	Role (per database)	Server-side routing enabled	Routes the query
`neo4j://`	yes	yes	write	follower	yes	Yes
`neo4j://`	yes	yes	read	follower	yes	no
`neo4j://`	yes	yes	write	read replica	yes	Yes
`neo4j://`	yes	yes	read	read replica	yes	no
`neo4j://`	yes	yes	write	leader	yes	no
`neo4j://`	yes	yes	read	leader	yes	no
`bolt://`	no	no	write	follower	yes	no
`bolt://`	no	no	read	follower	yes	no
`bolt://`	no	no	write	read replica	yes	no

CLIENT - Neo4j Driver (Bolt Protocol)				SERVER - Neo4j Cluster member		
`bolt://`	no	no	read	read replica	yes	no
`bolt://`	no	no	write	leader	yes	no
`bolt://`	no	no	read	leader	yes	no
`bolt://` (Python Driver)	no	yes	write	follower	yes	Yes
`bolt://` (Python Driver)	no	yes	read	follower	yes	no
`bolt://` (Python Driver)	no	yes	write	read replica	yes	Yes
`bolt://` (Python Driver)	no	yes	read	read replica	yes	no
`bolt://` (Python Driver)	no	yes	write	leader	yes	no
`bolt://` (Python Driver)	no	yes	read	leader	yes	no

Server-side routing connector configuration

Rerouted queries are communicated over the Bolt Protocol using a designated communication channel. The receiving end of of the communication is configured using the following settings:

- `dbms.routing.enabled`
- `dbms.routing.listen_address`
- `dbms.routing.advertised_address`

Server-side routing driver configuration

Server-side routing uses the Neo4j Java driver to connect to other cluster members. This driver is configured with settings of the format:

- `dbms.routing.driver.*`

The configuration options described in *Configuration* in the Neo4j Driver manuals have an equivalent in the server-side routing configuration.

Server-side routing encryption

Encryption of server-side routing communication is configured by the cluster SSL policy. For more information, see Cluster Encryption.

7.6.5. Store copy

Store copies are initiated when an instance does not have an up-to-date copy of the database. For example, this will be the case when a new instance is joining a cluster (without a seed). It can also happen as a consequence of falling behind the rest of the cluster, for reasons such as connectivity issues or having been shutdown. Upon re-establishing connection with the cluster, an instance will recognize that it is too far behind and fetch a new copy from the rest of the cluster.

A store copy is a major operation which may disrupt the availability of instances in the cluster. Store copies should not be a frequent occurrence in a well-functioning cluster, but rather be an exceptional operation that happens due to specific causes, e.g. network outages or planned maintenance outages. If store copies happen during regular operation, then the configuration of the cluster, or the workload directed at it, might have to be reviewed so that all instances can keep up, and that there is enough of a buffer of Raft logs and transaction logs to handle smaller transient issues.

The protocol used for store copies is robust and configurable. The network requests will be directed at an upstream member according to configuration and they will be retried despite transient failures. The maximum amount of time to retry every request can be configured with `causal_clustering.store_copy_max_retry_time_per_request`. If a request fails and the maximum retry time has elapsed then it will stop retrying and the store copy will fail.

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

The `causal_clustering.catch_up_client_inactivity_timeout` configuration is for all requests from the catchup client, including the pulling of transactions.

The default upstream strategy differs for Cores and Read Replicas. Cores will always send the initial request to the leader to get the most up-to-date information about the store. The strategy for the file and index requests for Cores is to vary every other request to a random Read Replica and every other to a random Core member.

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

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

Using the Replica instance in case of failure

In case of failure (e.g. a partial failure of a cluster due to the loss of an instance, but not of the majority), you may transform a Read Replica instance into a Core instance as a way to restore the cluster's core availability. However, keep in mind that this is not advised as it could cause data loss and complications in the Raft group.

To avoid that, the *read_replica* instance must not be initialized as a **single** instance, nor be introduced in a different or new cluster. This action would cause an override of the raft state, thus preventing the replica from successfully joining the targeted cluster.

After performing that change, follow these instructions to unbind the Replica instance and update the discovery configurations amongst cluster members:

1. Ensure that the converted *read_replica* currently belongs to the same cluster that it will be re-introduced back to, as a core. This can be done by performing `CALL dbms.cluster.overview()` and verifying the instance's address and cluster mode.
2. Stop and unbind the *read_replica* instance.

3. Update the cluster mode configuration in *neo4j.conf*, from `dbms.mode=READ_REPLICA` to `dbms.mode=CORE`.
4. Stop Neo4j on the removed core instances that are not intended to serve as core members.
5. Unbind those instances from the cluster by performing `neo4j-admin unbind` while they are stopped. This action will prevent such instances from subsequently attempting to rejoin the running cluster.

At this point, the previous *read_replica* (now *core*) instance may be introduced into the running cluster. To persist this change in the cluster's architecture, the following configuration updates are advised:

- On the previous *read_replica* (now *core*) instance, set `causal_clustering.discovery_advertised_address` and `causal_clustering.discovery_listen_address` as appropriate.
- Update the `causal_clustering.initial_discovery_members` configuration with the currently valid list of discovery addresses for each member of the cluster. This should replace the addresses of any removed *core(s)* with the discovery addresses of the previous *read_replica* (now *core*) instance.

In cases where `causal_clustering.discovery_type` is other than `LIST`, make sure to update the corresponding address resolution addresses records. For example, DNS A records for discovery types DNS and SRV, and any Kubernetes service address alternate to reflect the inclusion of the *read_replica* discovery address.

7.6.6. On-disk state

The on-disk state of cluster instances is different to that of standalone instances. The biggest difference being the existence of additional cluster state. Most of the files there are relatively small, but the Raft logs can become quite large depending on the configuration and workload.

It is important to understand that once a database has been extracted from a cluster and used in a standalone deployment, it must not be put back into an operational cluster. This is because the cluster and the standalone deployment now have separate databases, with different and irreconcilable writes applied to them.

If you try to reinsert a modified database back into the cluster, then the logs and stores will mismatch. Operators should not try to merge standalone databases into the cluster in the optimistic hope that their data will become replicated. That will not happen and will likely lead to unpredictable cluster behavior.

7.7. Settings reference

Parameter	Explanation
`dbms.mode`	This setting configures the operating mode of the database. For Causal Clustering, there are two possible modes: `CORE` or `READ_REPLICA`. **Example:** `dbms.mode=READ_REPLICA` will define this server as a Read Replica.

Parameter	Explanation
`dbms.read_only`	This setting is not supported.
`causal_clustering.minimum_core_cluster_size_at_formation`	Minimum number of Core machines required to form a cluster. **Example:** `causal_clustering.minimum_core_cluster_size_at_formation=3` will specify that the cluster will form when at least three Core members have discovered each other.
`causal_clustering.minimum_core_cluster_size_at_runtime`	The minimum size of the dynamically adjusted voting set (which only Core members may be a part of). Adjustments to the voting set happen automatically as the availability of Core members changes, due to explicit operations such as starting or stopping a member, or unintended issues such as network partitions. Please note that this dynamic scaling of the voting set is generally desirable, as under some circumstances it can increase the number of instance failures which may be tolerated. A majority of the voting set must be available before members are voted in or out. **Example:** `causal_clustering.minimum_core_cluster_size_at_runtime=3` will specify that the cluster should not try to dynamically adjust below three Core members in the voting set.

Parameter	Explanation
causal_clustering.discovery_type	This setting specifies the strategy that the instance will use to determine the addresses for other instances in the cluster to contact for *bootstrapping*. Possible values are: LIST, DNS, SRV, and K8S. LIST Treat causal_clustering.initial_discovery_members as a list of addresses of Core Servers to contact for discovery. DNS Treat causal_clustering.initial_discovery_members as a domain name to resolve via DNS. Expect DNS resolution to provide A records with hostnames or IP addresses of Cores to contact for discovery, on the port specified by causal_clustering.initial_discovery_members. SRV Treat causal_clustering.initial_discovery_members as a domain name to resolve via DNS. Expect DNS resolution to provide SRV records with hostnames or IP addresses, and ports, of Cores to contact for discovery. K8S Access the Kubernetes list service API to derive addresses of Cores to contact for discovery. Requires causal_clustering.kubernetes.label_selector to be a Kubernetes label selector for Kubernetes services running a Core each and causal_clustering.kubernetes.service_port_name to be a service port name identifying the discovery port of Core services. The value of causal_clustering.initial_discovery_members is ignored for this option. The value of this setting determines how causal_clustering.initial_discovery_members is interpreted. Detailed information about discovery and discovery configuration options is given in Discovery using DNS with multiple records. **Example:** causal_clustering.discovery_type=DNS combined with causal_clustering.initial_discovery_members=cluster01.example.com:5000 will fetch all DNS A records for *cluster01.example.com* and attempt to reach Neo4j instances listening on port 5000 for each A record's IP address.

Parameter	Explanation
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. In the default case, the initial discovery members are given as a comma-separated list of address/port pairs, and the default port for the discovery service is :5000. It is good practice to set this parameter to the same value on all Core Servers. The behavior of this setting can be modified by configuring the setting causal_clustering.discovery_type. This is described in detail in Discovery using DNS with multiple records. **Example:** causal_clustering.discovery_type=LIST combined with core01.example.com:5000,core02.example.com:5000,core03.example.com:5000 will attempt to reach Neo4j instances listening on core01.example.com, core01.example.com and core01.example.com; all on port 5000.
causal_clustering.discovery_advertised_address	The address/port setting that specifies where the instance advertises that it will listen for discovery protocol messages from other members of the cluster. If this instance is included in the initial_discovery_members of other cluster members, the value there must **exactly** match this advertised address. **Example:** causal_clustering.discovery_advertised_address=192.168.33.21:5001 indicates that other cluster members can communicate with this instance using the discovery protocol at host 192.168.33.20 and port 5001.
causal_clustering.raft_advertised_address	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:** causal_clustering.raft_advertised_address=192.168.33.20:7000 will listen for cluster communication in the network interface bound to 192.168.33.20 on port 7000.
causal_clustering.transaction_advertised_address	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:** causal_clustering.transaction_advertised_address=192.168.33.20:6001 will listen for transactions from cluster members on the network interface bound to 192.168.33.20 on port 6001.

Parameter	Explanation
causal_clustering.discovery_listen_address	The address/port setting that specifies which network interface and port the Neo4j instance will bind to for the cluster discovery protocol. **Example:** causal_clustering.discovery_listen_address=0.0.0.0:5001 will listen for cluster membership communication on any network interface at port 5001.
causal_clustering.raft_listen_address	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 causal_clustering.raft_advertised_address. **Example:** causal_clustering.raft_listen_address=0.0.0.0:7000 will listen for cluster communication on any network interface at port 7000.
causal_clustering.transaction_listen_address	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 causal_clustering.transaction_advertised_address. **Example:** causal_clustering.transaction_listen_address=0.0.0.0:6001 will listen for cluster communication on any network interface at port 6001.
causal_clustering.store_copy_max_retry_time_per_request	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:** causal_clustering.store_copy_max_retry_time_per_request=60min

7.7.1. Multi-data center settings

Parameter	Explanation
causal_clustering.multi_dc_license	Enables multi-data center features. Requires appropriate licensing. **Example:** causal_clustering.multi_dc_license=true will enable the multi-data center features.
causal_clustering.server_groups	A list of group names for the server used when configuring load balancing and replication policies. **Example:** causal_clustering.server_groups=us,us-east will add the current instance to the groups us and us-east.

Parameter	Explanation
causal_clustering.leadership_priority_group.<database>	The group of servers which should be preferred when selecting leaders for the specified database. If the instance currently acting as leader for this database is not a member of the configured server group, then the cluster will attempt to transfer leadership to an instance which *is* a member. It is not guaranteed that leadership will always be held by a server in the desired group. For example, if no member of the desired group is available or has up-to-date store contents. The cluster will seek to preserve availability, over respecting the leadership_priority_group setting. To set a default leadership_priority_group for all databases that do not have an explicitly set leadership_priority_group, the <database> can be omitted. See causal_clustering.leadership_priority_group. **Example:** causal_clustering.leadership_priority_group.foo=us will ensure that if the leader for foo is not held by a server configured with causal_clustering.server_groups=us, the cluster will attempt to transfer leadership to a server which is.
causal_clustering.upstream_selection_strategy	An ordered list in descending preference of the strategy which Read Replicas use to choose upstream database server from which to pull transactional updates. **Example:** causal_clustering.upstream_selection_strategy=connect-randomly-within-server-group,typically-connect-to-random-read-replica will configure the behavior so that the Read Replica will first try to connect to any other instance in the group(s) specified in causal_clustering.server_groups. Should we fail to find any live instances in those groups, then we will connect to a random Read Replica. A value of user-defined will enable custom strategy definitions using the setting causal_clustering.user_defined_upstream_strategy.
causal_clustering.user_defined_upstream_strategy	Defines the configuration of upstream dependencies. Can only be used if causal_clustering.upstream_selection_strategy is set to user-defined. **Example:** causal_clustering.user_defined_upstream_strategy=groups(north2); groups(north); halt() will look for servers in the north2. If none are available it will look in the north server group. Finally, if we cannot resolve any servers in any of the previous groups, then rule chain will be stopped via halt().
causal_clustering.load_balancing.plugin	The load balancing plugin to use. One pre-defined plugin named server_policies is available by default. **Example:** causal_clustering.load_balancing.plugin=server_policies will enable custom policy definitions.
causal_clustering.load_balancing.config.server_policies.<policy-name>	Defines a custom policy under the name <policy-name>. Note that load balancing policies are cluster-global configurations and should be defined the exact same way on all core machines. Example: causal_clustering.load_balancing.config.server_policies.north1_only=groups(north1)→min(2); halt(); defines a load balancing policy named north1_only. Queries are sent only to servers in the north1 server group, provided there are two of them available. If there are less than two servers in north1, the chain is halted. By default, the load balancer sends read requests only to replicas/followers, which means these two servers must be of that kind. To allow reads on the leader, set to causal_clustering.cluster_allow_reads_on_leader to true.

Chapter 8. Fabric

This chapter describes the following:

8.1. Introduction

8.1.1. Overview

Fabric, introduced in Neo4j 4.0, is a way to store and retrieve data in multiple databases, whether they are on the same Neo4j DBMS or in multiple DBMSs, using a single Cypher query. Fabric achieves a number of desirable objectives:

- a unified view of local and distributed data, accessible via a single client connection and user session
- increased scalability for read/write operations, data volume and concurrency
- predictable response time for queries executed during normal operations, a failover or other infrastructure changes
- High Availability and No Single Point of Failure for large data volume.

In practical terms, Fabric provides the infrastructure and tooling for:

- **Data Federation**: the ability to access data available in distributed sources in the form of **disjointed graphs**.
- **Data Sharding**: the ability to access data available in distributed sources in the form of a **common graph partitioned on multiple databases**.

With Fabric, a Cypher query can store and retrieve data in multiple federated and sharded graphs.

8.1.2. Fabric concepts

The fabric database

A Fabric setup includes a *Fabric virtual database*, which acts as the entry point to a federated or sharded graph infrastructure. This database is the execution context in which multi-graph queries can be executed. Drivers and client applications access and use the Fabric execution context by naming it as the selected database for a session. For more information, see *Databases and execution context* in the Neo4j Driver manuals.

The Fabric virtual database (execution context) differs from normal databases in that it cannot store any

data, and only relays data stored elsewhere. The Fabric virtual database can be configured on a standalone Neo4j DBMS only, i.e. on a Neo4j DBMS where the configuration setting `dbms.mode` must be set to `SINGLE`.

The Neo4j Admin commands cannot be applied to the Fabric virtual database. They must be run directly on the databases that are part of the Fabric setup.

Fabric graphs

In a Fabric virtual database, data is organized in the form of graphs. Graphs are seen by client applications as local logical structures, where physically data is stored in one or more databases. Databases accessed as Fabric graphs can be local, i.e in the same Neo4j DBMS, or they can be located in external Neo4j DBMSes. The databases are also accessible by client applications from regular local connections in their respective Neo4j DBMSs.

8.1.3. Deployment examples

Fabric constitutes an extremely versatile environment that provides scalability and availability with no single point of failure in various topologies. Users and developers may use applications that can work on a standalone DBMS as well on a very complex and largely distributed infrastructure without the need to apply any change to the queries accessing the Fabric graphs.

Development deployment

In its simplest deployment, Fabric can be used on a single instance, where Fabric graphs are associated to local databases. This approach is commonly used by software developers to create applications that will be deployed on multiple Neo4j DBMSs, or by power users who intend to execute Cypher queries against local disjoint graphs.

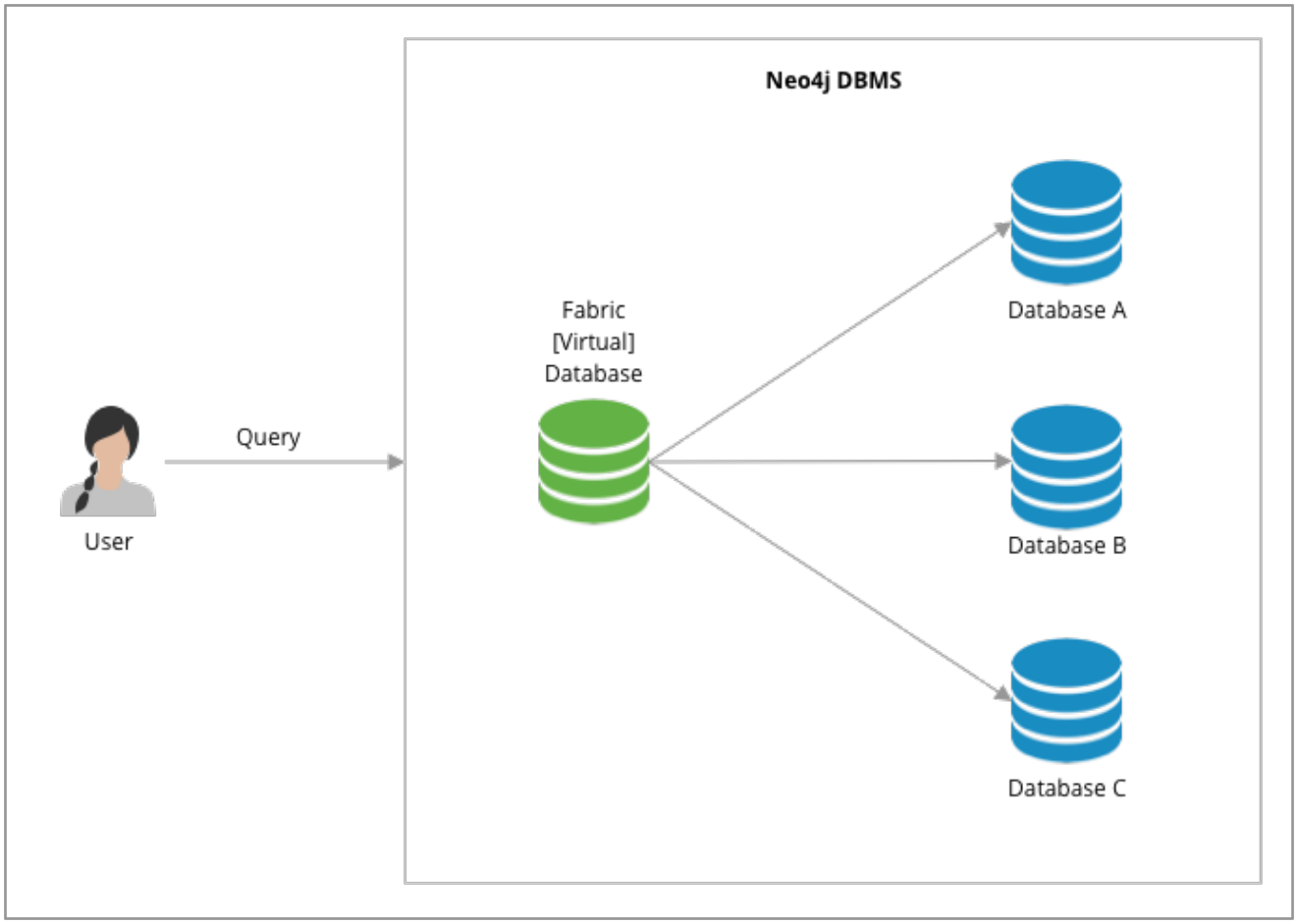


Figure 6. Fabric deployment in a single instance

Cluster deployment with no single point of failure

In this deployment Fabric guarantees access to disjoint graphs in high availability with no single point of failure. Availability if reached by creating redundant entry points for the Fabric Database (i.e. two standalone Neo4j DBMSs with the same Fabric configuration) and a minimum Causal Cluster of three members for data storage and retrieval. This approach is suitable for production environments and it can be used by power users who intend to execute Cypher queries against disjoint graphs.

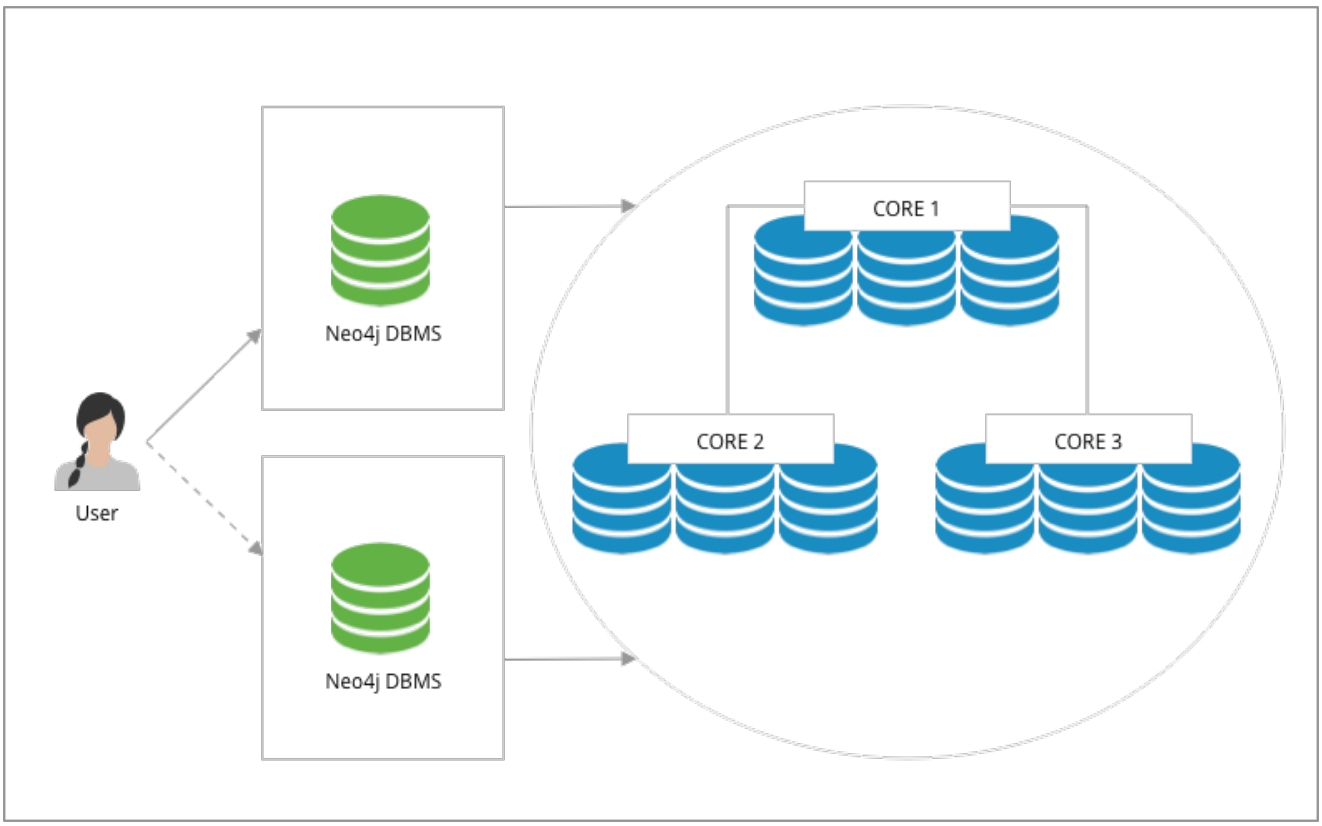


Figure 7. Fabric deployment with no single point of failure

Multi-cluster deployment

In this deployment Fabric provides high scalability and availability with no single point of failure. Disjoint clusters can be sized according to the expected workload and Databases may be colocated in the same cluster or they can be hosted in their own cluster to provide higher throughput. This approach is suitable for production environments where database can be sharded, federated or a combination of the two.

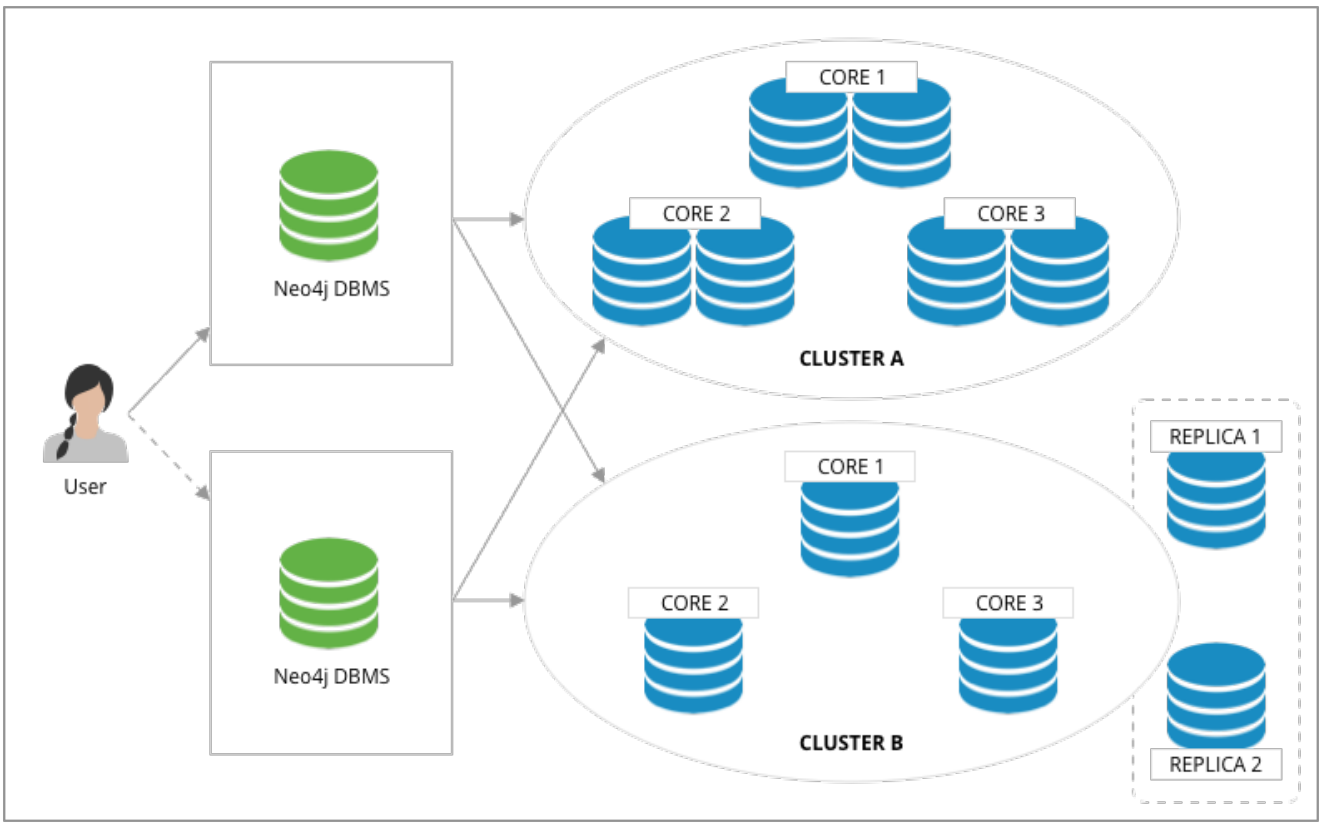


Figure 8. Fabric deployment for scalability with no single point of failure

8.2. Configuration

8.2.1. Fabric database setup

Fabric must be set on a standalone Neo4j DBMS: the settings in *neo4j.conf* are identified by the `fabric` namespace. The minimal requirements to setup Fabric are:

- A **virtual database name**: this is the entry point used by the client applications to access the Fabric environment.
- One or more **Fabric graph URI and database**: this a reference of a URI and a database for each graph set in the Fabric environment.

Local development setup example

Consider a standalone Neo4j DBMS, which has two databases, `db1` and `db2`. Note that all databases except for the default and `system` must be created using the `CREATE DATABASE` command.

Fabric is enabled by configuring:

```
fabric.database.name=example
```

This configuration enables Fabric and exposes the feature under the virtual database with the name `example`, which is accessible using the default URI, *neo4j://localhost:7687*. After connecting to the DBMS with the `example` database selected, you can run queries like the following:

```
USE db1
MATCH (n) RETURN n
 UNION
USE db2
MATCH (n) RETURN n
```

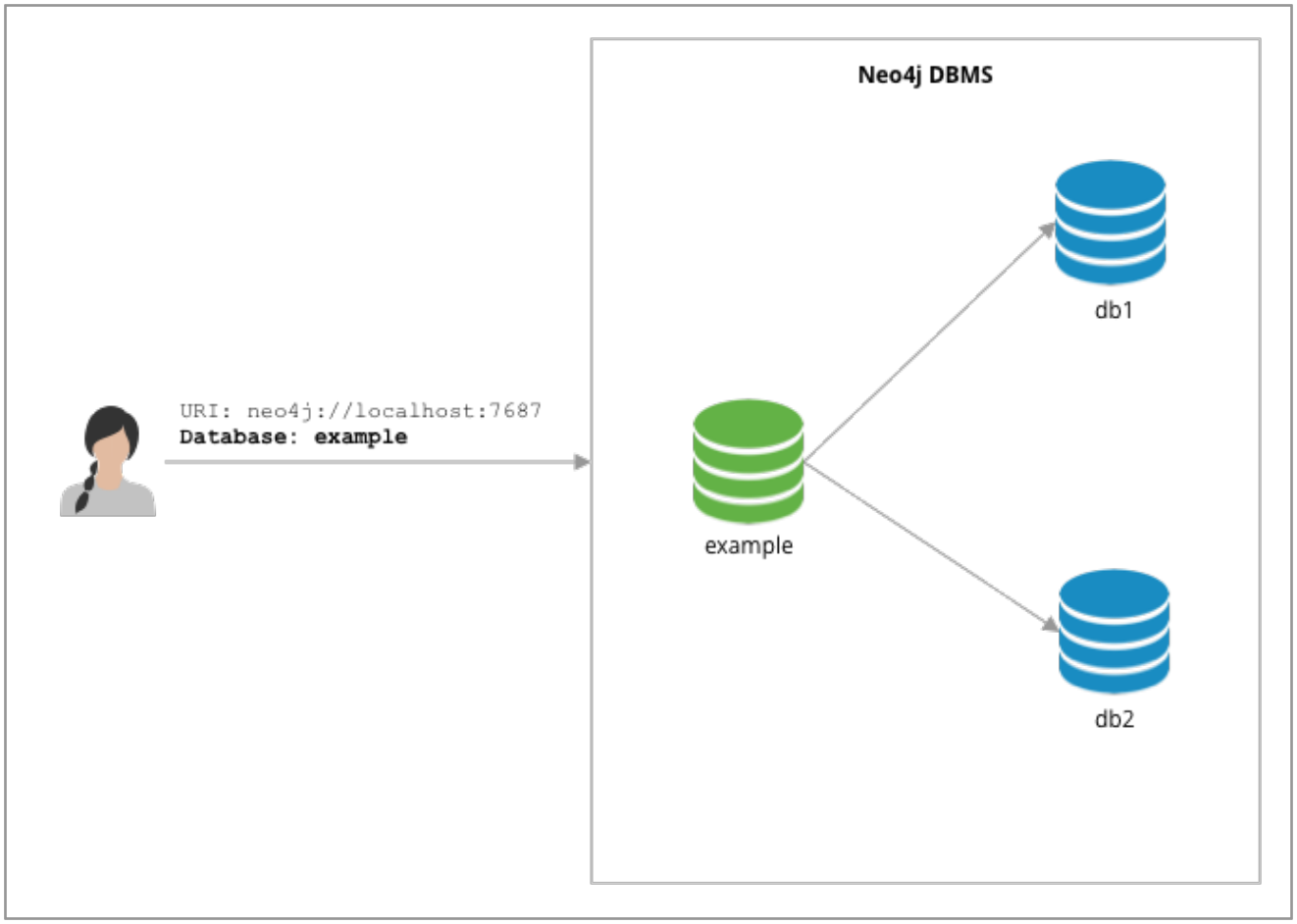


Figure 9. Minimal local Fabric setting in a development setup

Remote development setup example

This example consists of a setup with three standalone Neo4j DBMSs. One instance acts as the Fabric proxy, configured to enable Fabric. The other two instances contain the databases db1 and db2.

The following configuration enables Fabric on the proxy instance and allows it to access the databases in the other two instances.

```
fabric.database.name=example
fabric.graph.0.uri=neo4j://hostname-of-instance1:7687
fabric.graph.0.database=db1

fabric.graph.1.uri=neo4j://hostname-of-instance2:7687
fabric.graph.1.database=db2
```

This configuration enables Fabric and exposes the feature under the virtual database named example, which is accessible using the default URI, *neo4j://localhost:7687*. The Fabric graphs are uniquely identified by their IDs, 0 and 1.

After connecting to the DBMS with the selected database set to "example", you can run queries like the

following:

```
USE example.graph(0)
MATCH (n) RETURN n
  UNION
USE example.graph(1)
MATCH (n) RETURN n
```

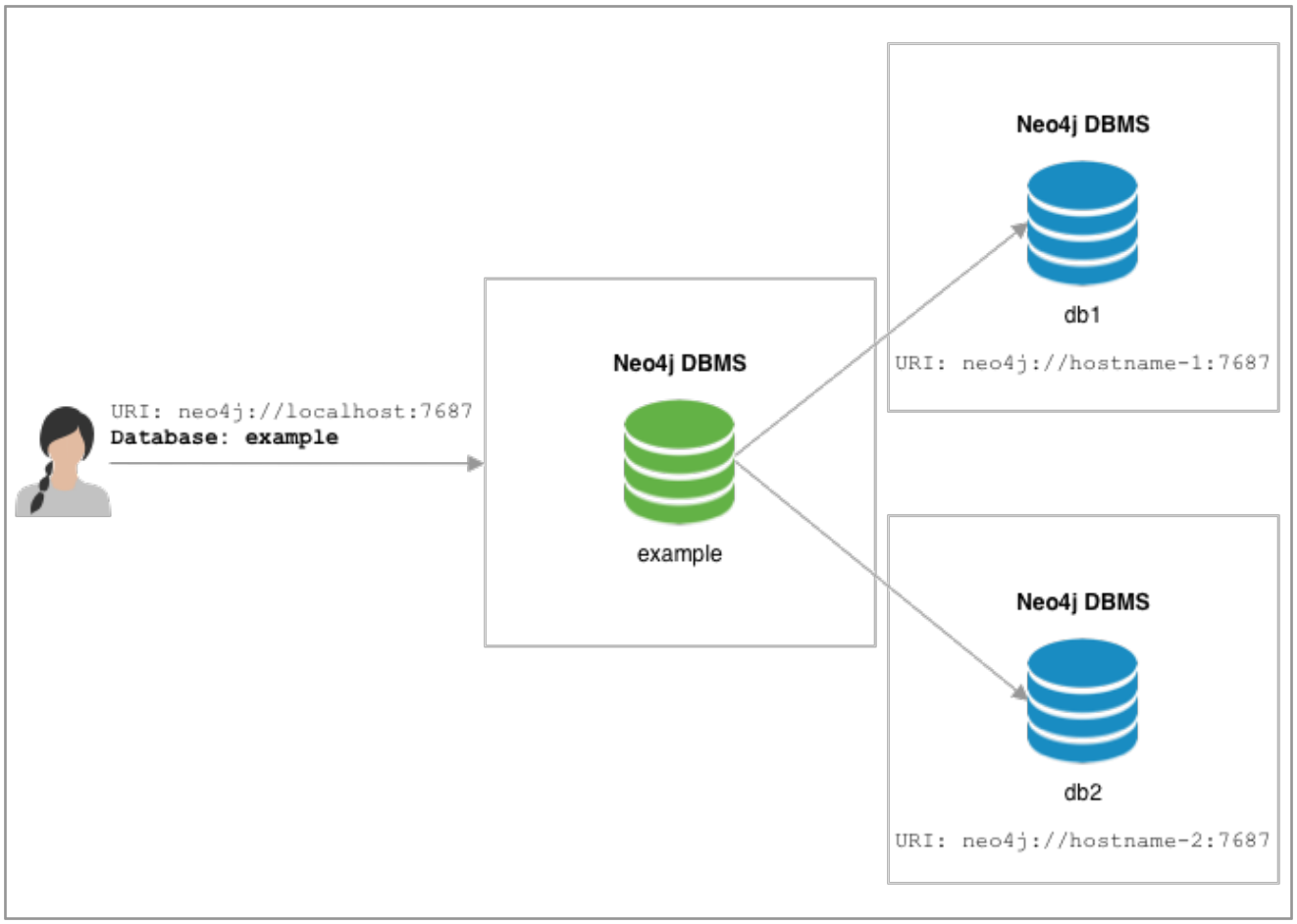


Figure 10. Minimal remote Fabric setting in a development setup

Naming graphs

Graphs can be identified by their ID or by a name. A graph can be named by adding an extra configuration setting, `fabric.graph.<ID>.name`.

For example, if the given names are `graphA` (associated to `db1`) and `graphB` (associated to `db2`), the two additional settings will be:

```
fabric.graph.0.name=graphA
fabric.graph.1.name=graphB
```

Giving names to graphs means you can refer to them by name in queries:

```
USE example.graphA
MATCH (n) RETURN n
 UNION
USE example.graphB
MATCH (n) RETURN n
```

Cluster setup with no single point of failure example

In this example, all components are redundant and data is stored in a Causal Cluster. In addition to the settings described in the previous example, a setting with no single point of failure requires the use of the *routing servers* parameter, which specifies a list of standalone Neo4j DBMSs that expose the same Fabric database and configuration. This parameter is required in order to simulate the same connectivity that client applications use with Causal Cluster, which means, in case of fault of one instance, the client application may revert to another existing instance.

Assume that in this example, the data is stored in three databases: `db1`, `db2` and `db3`. The configuration of Fabric would be:

```
dbms.mode=SINGLE

fabric.database.name=example
fabric.routing.servers=server1:7687,server2:7687

fabric.graph.0.name=graphA
fabric.graph.0.uri=neo4j://core1:7687,neo4j://core2:7687,neo4j://core3:7687
fabric.graph.0.database=db1

fabric.graph.1.name=graphB
fabric.graph.1.uri=neo4j://core1:7687,neo4j://core2:7687,neo4j://core3:7687
fabric.graph.1.database=db2

fabric.graph.2.name=graphC
fabric.graph.2.uri=neo4j://core1:7687,neo4j://core2:7687,neo4j://core3:7687
fabric.graph.2.database=db3
```

The configuration above must be added to the *neo4j.conf* file of the Neo4j DBMSs `server1` and `server2`. The parameter `fabric.routing.servers` contains the list of available standalone Neo4j DBMSs hosting the Fabric database. The parameter `fabric.graph.<ID>.uri` can contain a list of URIs, so in case the first server does not respond to the request, the connection can be established to another server that is part of the cluster. The URIs refer to the `neo4j://` schema so that Fabric can retrieve a routing table and can use one of the members of the cluster to connect.

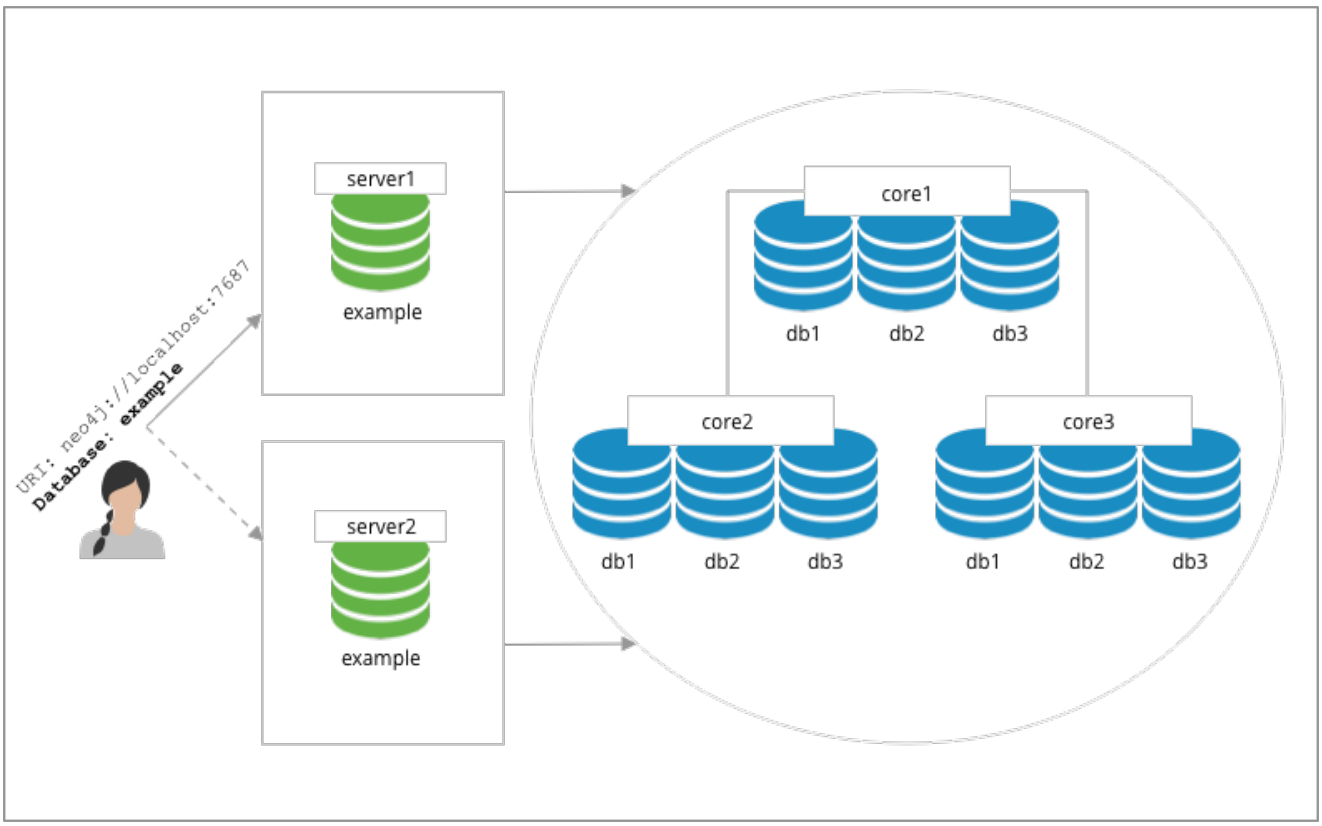


Figure 11. Fabric setting with Causal Cluster and no single point of failure

Cluster routing context

The URIs in the graph settings may include routing contexts, which are described in the Neo4j Driver manuals. This can be used to associate a Fabric graph with a filtered subset of Causal Cluster members, by selecting a routing policy.

As an example, assuming you have a server policy called `read_replicas` defined in the configuration of the cluster you are targeting, you might set up a Fabric graph that accesses only the read replicas of the cluster.

```
fabric.graph.0.name=graphA
fabric.graph.0.uri=neo4j://core1:7687?policy=read_replicas
fabric.graph.0.database=db1
```

This enables scenarios where queries executed through Fabric are explicitly offloaded to specific instances in clusters.

8.2.2. Authentication and authorization

Credentials

Connections between the Fabric database and the Neo4j DBMSs hosting the data are created using the same credentials that are supplied in the client connection to the Fabric database. It is recommended to maintain a set of user credentials on all the Neo4j DBMSs; if required, a subset of credentials may be set for local access on the remote DBMSs.

User and role administration

User and role administration actions are not automatically propagated to the Fabric environment, therefore security settings must be executed on any DBMS that is part of Fabric.

Privileges on the Fabric database

In order to use all Fabric features, users of Fabric databases need `ACCESS` and `READ` privileges.

8.2.3. Important settings

This section provides general information about Fabric settings and describes the ones important for creating a fabric set-up. For the full list of Fabric configuration options, see Configuration settings.

Fabric settings are divided in the following categories:

- **System Settings**: DBMS-level settings.
- **Graph Settings**: definition and configuration of Fabric graphs.
- **Drivers Settings**: configuration of drivers used to access Neo4j DBMSs and databases associated to Fabric graphs.

System settings

Table 32. Fabric system settings

Parameter	Description
`fabric.database.name`	Name of the Fabric database. Neo4j Fabric currently supports one Fabric database in a standalone Neo4j DBMS.
`fabric.routing.servers`	A comma-separated list of Neo4j DBMSs that share the same Fabric configuration. These DBMSs form a routing group. A client application will route transactions through a Neo4j driver or connector to one of the members of the routing group. A Neo4j DBMS is represented by its Bolt connector address. Example: `fabric.routing.servers=server1:7687,server2:7687`.

Graph settings

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

Table 33. Fabric graph settings

Parameter	Description
`fabric.graph.<ID>.uri`	URI of the Neo4j DBMS hosting the database associated to the Fabric graph. Example: `neo4j://somewhere:7687`
`fabric.graph.<ID>.database`	Name of the database associated to the Fabric graph.

Parameter	Description
`fabric.graph.<ID>.name`	Name assigned to the Fabric graph. The name can be used in Fabric queries.
`fabric.graph.<ID>.driver.*`	Any specific driver setting, that means, any setting related to a connection to a specific Neo4j DBMS and database. This setting overrides a global driver setting.

When configuring access to a remote DBMS, make sure that the remote is configured to advertise its address correctly, using either `dbms.default_advertised_address` or `dbms.connector.bolt.advertised_address`. Fabric reads the routing table from the remote DBMS and then connects back using an appropriate entry in that table.

Drivers settings

Fabric uses the Neo4j Java driver to connect to and access the data stored in Neo4j databases associated to Fabric graphs. This section presents the most important parameters available to configure the driver.

Drivers settings are configured with parameters with names of the format:

`fabric.driver.<suffix>`

A setting can be global, i.e. be valid for all the drivers used in Fabric, or it can be specific for a given connection to a Neo4j database associated to a graph. The graph-specific setting overrides the global configuration for that graph.

Example 42. Global drivers setting versus graph-specific drivers setting

A drivers setting for Fabric as the following is valid for all the connections established with the Neo4j DBMSs set in Fabric:

```
fabric.driver.api=RX
```

A graph-specific connection for the database with `ID=6` will override the `fabric.driver.api` setting for that database:

```
fabric.graph.6.driver.api=ASYNC
```

Table 34. Fabric drivers setting suffixes

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

Most driver options described in *Configuration* in the Neo4j Driver manuals have an equivalent in Fabric configuration.

8.3. Queries

In this section we will look at a few example queries that show how to perform a range of different tasks.

The examples in this section make use of the two Cypher clauses: `USE` and `CALL {}`. The syntax is explained in detail in the Cypher Manual:

- See Cypher Manual → CALL {} for details about the `CALL {}` clause.
- See Cypher Manual → USE for details about the `USE` clause.

8.3.1. Query a single graph

Example 43. Reading and returning data from a single graph.

```
USE example.graphA
MATCH (movie:Movie)
RETURN movie.title AS title
```

The `USE` clause at the beginning of the query causes the rest of the query to execute against the `example.graphA` graph.

8.3.2. Query multiple graphs

Example 44. Reading and returning data from two named graphs

```
USE example.graphA
MATCH (movie:Movie)
RETURN movie.title AS title
  UNION
USE example.graphB
MATCH (movie:Movie)
RETURN movie.title AS title
```

The first part of the `UNION` query executes against the `example.graphA` graph and the second part executes against the `example.graphB` graph.

8.3.3. Query all graphs

Example 45. Reading and returning data from all graphs

```
UNWIND example.graphIds() AS graphId
CALL {
  USE example.graph(graphId)
  MATCH (movie:Movie)
  RETURN movie.title AS title
}
RETURN title
```

We call the built-in function `example.graphIds()` to get the graph IDs for all remote graphs in our Fabric setup. We `UNWIND` the result of that function to get one record per graph ID. The `CALL {}` subquery is executed once per incoming record. We use a `USE` clause in the subquery with a dynamic graph lookup, causing the subquery to execute once against each remote graph. At the end of the main query we simply `RETURN` the title variable.

8.3.4. Query result aggregation

Example 46. Getting the earliest release year of all movies in all graphs

```
UNWIND example.graphIds() AS graphId
CALL {
  USE example.graph(graphId)
  MATCH (movie:Movie)
  RETURN movie.released AS released
}
RETURN min(released) AS earliest
```

From each remote graph we return the released property of each movie. At the end of the main query we aggregate across the full result to calculate the global minimum.

8.3.5. Correlated subquery

Example 47. Correlated subquery

Assume that `graphA` contains American movies and `graphB` contains European movies. Find all European movies released in the same year as the latest released American movie:

```
CALL {
  USE example.graphA
  MATCH (movie:Movie)
  RETURN max(movie.released) AS usLatest
}
CALL {
  USE example.graphB
  WITH usLatest
  MATCH (movie:Movie)
  WHERE movie.released = usLatest
  RETURN movie
}
RETURN movie
```

We query the `example.graphA` and return the release year of the latest release. We then query the `example.graphB`. `WITH usLatest` is an import clause which lets us refer to the `usLatest` variable inside the subquery. We find all the movies in this graph that fulfill our condition and return them.

It is not possible to switch the current graph in a nested query. For example, the following query is illegal:

Example 48. Illegal correlated subquery

```
USE example.graphA
MATCH (movie:Movie)
WITH movie.title AS title
CALL {
  USE example.graphB // Cannot swicth from example.graphA
  WITH title
  MATCH (otherMovie:Movie)
  WHERE otherMovie.title STARTS WITH title
  RETURN otherMovie.title AS otherTitle
}
RETURN title, otherTitle
```

This limitation can be circumvented by having subqueries after one another, but without nesting them.

8.3.6. Updating query

Example 49. Create a new movie node

```
USE example.graphB
CREATE (m:Movie)
SET m.title = 'Léon: The Professional'
SET m.tagline = 'If you want the job done right, hire a professional.'
SET m.released = 1994
```

8.3.7. Mapping functions

Mapping functions are a common Fabric usage pattern. In the previous examples, graphs were identified

by providing static graph names in the query. Fabric may be used in scenarios where graphs are identified by a mapping mechanism that can, for example, identify a key of an object contained in a graph. This can be achieved by using user-defined functions or other functions that may be already available. These functions ultimately return the ID of a graph in Fabric.

Mapping functions are commonly used in sharding scenarios. In Fabric, shards are associated to graphs, hence mapping functions are used to identify a graph, i.e. a shard.

Refer to Java Reference → User-defined functions for details on how to create user-defined functions.

Let's assume that Fabric is setup in order to store and retrieve data associated to nodes with the label `user`. User nodes are partitioned in several graphs (shards) in Fabric. Each `user` has a numerical `userId`, which is unique in all Fabric. We decide on a simple scheme where each `user` is located on a graph determined by taking the `userId` modulo the number of graphs. We create a user-defined function which implements the following pseudo code:

```
sharding.userIdToGraphId(userId) = userId % NUM_SHARDS
```

Assuming we have supplied a query parameter `$userId` with the specific userId that we are interested in, we use our function in this way:

```
USE example.graph( sharding.userIdToGraphId($userId) )
MATCH (u:User) WHERE u.userId = $userId
RETURN u
```

8.3.8. Fabric built-in functions

Fabric functions are located in a namespace corresponding to a Fabric database in which they are used. The following table provides a description of Fabric built-in functions:

Table 35. Fabric built-in functions

Function	Explanation
`<fabric database name>.graphIds()`	Provides a list of IDs of all remote graph configured for the given Fabric database.
`<fabric database name>.graph(graphId)`	Maps a graph ID to a Graph. It accepts a graph ID as a parameter and returns a graph representation accepted by USE clause. This function is supported only in `USE` clauses

8.4. Further considerations

DBMS mode

The DBMS hosting the Fabric virtual database cannot be part of a Causal Cluster: it can only be a DBMS with `dbms.mode=SINGLE`.

Sharding an existing database

An existing database can be sharded with the help of the `neo4j-admin copy` command. See Sharding

data with the copy command for an example.

Database compatibility

Fabric is part of Neo4j DBMS and does not require any special installation or plugin. Fabric databases can be associated to databases available on Neo4j DBMS version 4.1 or 4.2.

Fabric configuration

The Neo4j DBMSs that host the same Fabric virtual database must have the same configuration settings. The configuration must be kept in-sync and applied by the Database Administrator.

Security credentials

The Neo4j DBMSs that host the same Fabric virtual database must have the same user credentials. Any change of password on a machine that is part of Fabric, must be kept in-sync and applied to all the Neo4j DBMSs that are part of Fabric.

Administration commands

Fabric does not support running Cypher administration commands on or through the Fabric virtual database. Any database management commands, index and constraint management commands, or user and security management commands must be issued directly to the DBMSs and databases that are part of the Fabric setup.

Neo4j embedded

Fabric is not available when Neo4j is used as an embedded database in Java applications. Fabric can be used only in a typical client/server mode, when users connect to a Neo4j DBMS from their client application or tool, via Bolt or HTTP protocol.

8.5. Sharding data with the copy command

The copy command can be used to filter out data for a Fabric installation. In the following example, a sample database is separated into 3 shards.

Example 50. Use the copy command to filter out data for a Fabric installation.

The sample database contains the following data:

```
(p1 :Person :S2 {id:123, name: "Ava"})
(p2 :Person :S2 {id:124, name: "Bob"})
(p3 :Person :S3 {id:125, name: "Cat", age: 54})
(p4 :Person :S3 {id:126, name: "Dan"})
(t1 :Team :S1 :SAll {id:1, name: "Foo", mascot: "Pink Panther"})
(t2 :Team :S1 :SAll {id:2, name: "Bar", mascot: "Cookie Monster"})
(d1 :Division :SAll {name: "Marketing"})
(p1)-[:MEMBER]->(t1)
(p2)-[:MEMBER]->(t2)
(p3)-[:MEMBER]->(t1)
(p4)-[:MEMBER]->(t2)
```

The data has been prepared using queries to add the labels `:S1`,`:S2`, `:S3`, and `:SAll`, which denotes the target shard. *Shard 1* contains the team data. *Shard 2* and *Shard 3* contain person data.

1. Create *Shard 1* with:

```
$neo4j-home> bin/neo4j-admin copy --from-database=neo4j \
   --to-database=shard1 \
   --keep-only-nodes-with-labels=S1,SAll \ ①
   --skip-labels=S1,S2,S3,SAll ②
```

① The `--keep-only-node-with-labels` property is used to filter out everything that does not have the label `:S1` or `:SAll`.

② The `--skip-labels` property is used to exclude the temporary labels you created for the sharding process.

The resulting shard contains the following:

```
(t1 :Team {id:1, name: "Foo", mascot: "Pink Panther"})
(t2 :Team {id:2, name: "Bar", mascot: "Cookie Monster"})
(d1 :Division {name: "Marketing"})
```

2. Create *Shard 2*:

```
$neo4j-home> bin/neo4j-admin copy --from-database=neo4j \
   --to-database=shard2 \
   --keep-only-nodes-with-labels=S2,SAll \
   --skip-labels=S1,S2,S3,SAll \
   --keep-only-node-properties=Team.id
```

In *Shard 2*, you want to keep the `:Team` nodes as proxy nodes, to be able to link together information from the separate shards. The nodes will be included since they have the label `:SAll`, but you specify `--keep-only-node-properties` so as to not duplicate the team information from *Shard 1*.

```
(p1 :Person {id:123, name: "Ava"})
(p2 :Person {id:124, name: "Bob"})
(t1 :Team {id:1})
(t2 :Team {id:2})
(d1 :Division {name: "Marketing"})
(p1)-[:MEMBER]->(t1)
(p2)-[:MEMBER]->(t2)
```

Observe that `--keep-only-node-properties` did not filter out `Person.name` since the `:Person` label was not mentioned in the filter.

3. Create *Shard 3*, but with the filter `--skip-node-properties`, instead of `--keep-only-node-properties`.

```
$neo4j-home> bin/neo4j-admin copy --from-database=neo4j \
   --to-database=shard3 \
   --keep-only-nodes-with-labels=S3,SAll \
   --skip-labels=S1,S2,S3,SAll \
   --skip-node-properties=Team.name,Team.mascot
```

The result is:

```
(p3 :Person {id:125, name: "Cat", age: 54})
(p4 :Person {id:126, name: "Dan"})
(t1 :Team {id:1})
(t2 :Team {id:2})
(d1 :Division {name: "Marketing"})
(p3)-[:MEMBER]->(t1)
(p4)-[:MEMBER]->(t2)
```

As demonstrated, you can achieve the same result with both `--skip-node-properties` and `--keep-only-node-properties`. In this example, it is easier to use `--keep-only-node-properties` because only one property should be kept. The relationship property filters works in the same way.

Chapter 9. Backup and restore

This chapter describes the following:

- Backup and restore planning — What to consider when designing your backup and restore strategy.
- Backup modes — The supported backup modes.
- Back up an online database — How to back up an online database.
- Restore a database backup — How to restore a database backup in a live Neo4j deployment.
- Back up an offline database — How to back up an offline database.
- Restore a database dump — How to restore a database dump in a live Neo4j deployment.
- Copy a database store — How to copy data store from an existing database to a new database.

9.1. Backup and restore planning

There are two main reasons for backing up your Neo4j databases and storing them in a safe, off-site location:

- to be able to quickly recover your data in case of failure, for example related to hardware, human error, or natural disaster.
- to be able to perform routine administrative operations, such as moving a database from one instance to another, upgrading, or reclaiming space.

9.1.1. Backup and restore strategy

Depending on your particular deployment and environment, it is important to design an appropriate backup and restore strategy.

There are various factors to consider when deciding on your strategy, such as:

- Type of environment – development, test, or production.
- Data volumes.
- Number of databases.
- Available system resources.
- Downtime tolerance during backup and restore.
- Demands on Neo4j performance during backup and restore. This factor might lead your decision towards performing these operations during an off-peak period.
- Tolerance for data loss in case of failure.
- Tolerance for downtime in case of failure. If you have zero tolerance for downtime and data loss, you might want to consider performing an online or even a scheduled backup.
- Frequency of updates to the database.
- Type of backup and restore method (online or offline), which may depend on whether you want to:

- perform full backups (online or offline).
- automatically check the consistency of a database backup (online only).
- perform incremental backups (online only).
- use SSL/TLS for the backup network communication (online only).
- keep your databases as archive files (offline only).

- How many backups you want to keep.
- Where the backups will be stored — drive or remote server, cloud storage, different data center, different location, etc.

It is recommended to store your database backups on a separate off-site server (drive or remote) from the database files. This ensures that if for some reason your Neo4j DBMS crashes, you will be able to access the backups and perform a restore.

- How you will test recovery routines, and how often.

9.1.2. Backup and restore options

Neo4j supports backing up and restoring both online and offline databases. It uses Neo4j Admin tool commands, which can be run from a live, as well as from an offline Neo4j DBMS. All `neo4j-admin` commands must be invoked as the `neo4j` user to ensure the appropriate file permissions.

- `neo4j-admin backup/restore` (Enterprise only) -– used for performing online backup (full and incremental) and restore operations. The database to be backed up must be in **online** mode. This command is suitable for production environments, where you cannot afford downtime. However, it is more memory intensive and is not supported in Neo4j Aura.

When using `neo4j-admin backup` in Causal Cluster, it is recommended to back up from an external instance as opposed to reuse instances that form part of the cluster.

- `neo4j-admin dump/load` -– used for performing offline dump and load operations. The database to be dumped must be in **offline** mode. This dump command is suitable for environments, where downtime is not a factor. It is faster than the backup command, and produces an archive file, which occupies less space than a normal database structure.
- `neo4j-admin copy` -– used for copying an offline database or backup. This command can be used for cleaning up database inconsistencies, reclaiming unused space, and migrating Neo4j 3.5.any directly to any 4.x version of Neo4j, including the latest version, skipping the intermediate steps. For a detailed example, see Upgrade and Migration Guide → Tutorial: Back up and copy a database in a standalone instance.

File system copy-and-paste of databases is not supported.

In Fabric deployments, the Neo4j Admin commands `backup`, `restore`, `dump`, `load`, `copy`, and `check-consistency` are not supported for use on the Fabric virtual database. They must be run directly on the databases that are part of the Fabric setup.

Table 36. The following table describes the commands capabilities and usage.

Capability/ Usage	`neo4j-admin backup`	`neo4j-admin dump`	`neo4j-admin restore`	`neo4j-admin load`	`neo4j-admin copy`
Neo4j Edition	Enterprise	all	Enterprise	all	Enterprise
Live Neo4j DBMS	▮	▮	▮	▮	▮
Offline Neo4j DBMS	▮	▮	▮	▮	▮
Run against a user database	▮	▮	▮	▮	▮
Run against the `system` database	▮	▮	▮	▮	▮
Run against the `fabric` database	✖	✖	✖	✖	✖
Perform full backups	▮	▮	n/a	n/a	n/a
Perform incremental backups	▮	▮	n/a	n/a	n/a
Applied to an online database	▮	▮	▮	▮	▮
Applied to an offline database	▮	▮	▮	▮	▮
Can be run remotely (support SSL)	▮	▮	▮	▮	▮
Command input	database	database	database backup	archive (.dump)	database or database backup
Command output	database	archive (.dump)	database	database	database; no schema store
Run consistency check after completion	▮	▮	▮	▮	▮
Clean up database inconsistencies	▮	▮	▮	▮	▮
Compact data store	▮	▮	▮	▮	▮

9.1.3. Databases to backup

A Neo4j DBMS can host multiple databases. Both Neo4j Community and Enterprise Editions have a default user database, called `neo4j`, and a `system` database, which contains configurations, e.g., operational states of databases, security configuration, schema definitions, login credentials, and roles. In the Enterprise Edition, you can also create additional user databases. Each of these databases are backed up

independently of one another.

It is very important to back up each of your databases, including the `system` database, in a safe location.

9.1.4. Additional files to back up

The following files must be backed up separately from the databases:

- The *neo4j.conf* file. If you have a cluster deployment, you should back up the configuration file for each cluster member.
- All the files used for encryption, i.e., private key, public certificate, and the contents of the *trusted* and *revoked* directories. The locations of these are described in SSL framework. If you have a cluster, you should back up these files for each cluster member.
- If using custom plugins, make sure that you have the plugins in a safe location.

9.1.5. Storage considerations

For any backup, it is important that you store your data separately from the production system, where there are no common dependencies, and preferably off-site. If you are running Neo4j in the cloud, you may use a different availability zone or even a separate cloud provider. Since backups are kept for a long time, the longevity of archival storage should be considered as part of backup planning.

9.2. Backup modes

The backup client can operate in two different modes – a *full backup* and an *incremental backup*.

9.2.1. Full backup

A full backup is always required initially for the very first backup into a target location.

The full backup can be run against both an **online** (using `neo4j-admin backup`) and an **offline** (using `neo4j-admin dump`) database.

Example 51. Full backup against an online database

```
$neo4j-home> export HEAP_SIZE=2G
$neo4j-home> mkdir /mnt/backups
$neo4j-home> bin/neo4j-admin backup --from=192.168.1.34 --backup-dir=/mnt/backups/neo4j --database
=neo4j --pagecache=4G
Doing full backup...
2017-02-01 14:09:09.510+0000 INFO  [o.n.c.s.StoreCopyClient] Copying neostore.nodestore.db.labels
2017-02-01 14:09:09.537+0000 INFO  [o.n.c.s.StoreCopyClient] Copied neostore.nodestore.db.labels 8.00
kB
2017-02-01 14:09:09.538+0000 INFO  [o.n.c.s.StoreCopyClient] Copying neostore.nodestore.db
2017-02-01 14:09:09.540+0000 INFO  [o.n.c.s.StoreCopyClient] Copied neostore.nodestore.db 16.00 kB
...
...
...
```

For more information about online backup options and how to control memory usage, see Back up an online database.

For more information about performing a full backup against an **offline** database, see Back up an offline database.

9.2.2. Incremental backup

After the initial full backup, the subsequent backups attempt to use the incremental mode, where just the delta of the transaction logs since the last backup are transferred and applied onto the target location. If the required transaction logs are not available on the backup server, then the backup client falls back on performing a full backup instead, unless `--fallback-to-full` is disabled.

The incremental backup can be run only against an **online** database.

Example 52. Incremental backup against an online database

```
$neo4j-home> export HEAP_SIZE=2G
$neo4j-home> bin/neo4j-admin backup --from=192.168.1.34 --backup-dir=/mnt/backups/neo4j --database
=neo4j --pagecache=4G
Destination is not empty, doing incremental backup...
Backup complete.
```

For more information about online backup options and how to control memory usage, see Back up an online database.

9.3. Back up an online database

Remember to plan your backup carefully and to back up each of your databases, including the `system` database.

9.3.1. Command

A Neo4j database can be backed up in **online mode** using the `backup` command of `neo4j-admin`. The command must be invoked as the `neo4j` user to ensure the appropriate file permissions.

Usage

The `neo4j-admin backup` command can be used for performing both full and incremental backups of an **online** database. The command can be run both locally and remotely. By default, `neo4j-admin backup` also checks the database consistency at the end of every backup operation. However, it uses a significant amount of resources, such as memory and CPU. Therefore, it is recommended to perform the backup on a separate dedicated machine. The `neo4j-admin backup` command also supports SSL/TLS. For more information, see Online backup configurations.

`neo4j-admin backup` is not supported in Neo4j Aura.

`neo4j-admin backup` is not supported for use on the Fabric virtual database. It must be run directly on the databases that are part of the Fabric setup.

Syntax

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

Options

Option	Default	Description
`--backup-dir`		Target directory.
`--verbose`		Enable verbose output.
`--from`	`localhost:6362`	Host and port of Neo4j.
`--database`	`neo4j`	Name of the remote database to backup. Can contain `*` and `?` for globbing.
`--fallback-to-full`	`true`	If an incremental backup fails, backup will move the old backup to `<name>.err.<N>` and fallback on a full backup instead.
`--pagecache`	`8m`	The size of the page cache to use for the backup process.
`--check-consistency`	`true`	Run a consistency check against the database backup.
`--report-dir`	`.`	Directory where consistency report will be written.
`--check-graph`	`true`	Perform consistency checks between nodes, relationships, properties, types, and tokens.
`--check-indexes`	`true`	Perform consistency checks on indexes.
`--check-index-structure`	`true`	Perform structure checks on indexes.

Option	Default	Description
`--check-label-scan-store`	`true`	Perform consistency checks on the label scan store.
`--check-property-owners`	`false`	Perform additional consistency checks on property ownership. This check is very expensive in time and memory.
`--additional-config`		Configuration file to provide additional or override the existing configuration settings in the *neo4j.conf* file.
`--include-metadata`		Include metadata in the backup. Metadata contains security settings related to the database. Cannot be used for backing up the `system` database. - `roles` - commands to create the roles and privileges (for both database and graph) that affect the use of the database. - `users` - commands to create the users that can use the database and their role assignments. - `all` - include roles and users.

Exit codes

Depending on whether the backup was successful or not, `neo4j-admin backup` exits with different codes. The error codes include details of what error was encountered.

Table 37. Neo4j Admin backup exit codes when backing up one database

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

Table 38. Neo4j Admin backup exit codes when backing multiple databases

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

9.3.2. Online backup configurations

Server configuration

The table below lists the basic server parameters relevant to backups. Note that, by default, the backup service is enabled but only listens on localhost (127.0.0.1). This needs to be changed if backups are to be taken from another machine.

Make this change only if you need the remote backup. If your network is not adequately isolated, this change might expose your system to threats.

Table 39. Server parameters for backups

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

It is not recommended to use an NFS mount for backup purposes as this is likely to corrupt and slow down the backup.

Make sure to follow the Security Configurations in order to prevent unauthorized users from accessing the DBMS by having access to the backup server.

Memory configuration

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

Configure heap size for the backup

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

Configure page cache for the backup

The page cache size can be configured by using the `--pagecache` option of the `neo4j-admin backup` command. If not explicitly defined, the page cache defaults to `8MB`.

You should give the Neo4J page cache as much memory as possible, as long as it satisfies the following constraint:

Neo4J page cache + OS page cache < available RAM, where 2 to 4GB should be dedicated to the operating system's page cache.

For example, if your current database has a `Total mapped size` of `128GB` as per the *debug.log*, and you have enough free space (meaning you have left aside 2 to 4 GB for the OS), then you can set `--pagecache` to `128GB`.

Computational resources configurations

Consistency checking

Checking the consistency of the backup is a major operation which may consume significant computational resources, such as, memory, CPU, I/O. When backing up an online database, the consistency checker is invoked at the end of the process by default. Therefore, it is highly recommended to perform the backup and consistency check on a dedicated machine, which has sufficient free resources, to avoid adversely affecting the running server.

Alternatively, you can decouple the backup operation from the consistency check (using the `neo4j-admin backup` option `--check-consistency=false`) and schedule that part of the workflow to happen at a later point in time, on a dedicated machine. Consistency checking a backup is vital for safeguarding and ensuring the quality of the data, and should not be underestimated. For more information, see Consistency checker.

To avoid running out of resources on the running server, it is recommended to perform the backup on a separate dedicated machine.

Transaction log files

The transaction log files, which keep track of recent changes, are rotated and pruned based on a provided configuration. For example, setting `dbms.tx_log.rotation.retention_policy=3` files keeps 3 transaction log files in the backup. Because recovered servers do not need all of the transaction log files that have already been applied, it is possible to further reduce storage size by reducing the size of the files to the bare minimum. This can be done by setting `dbms.tx_log.rotation.size=1M` and `dbms.tx_log.rotation.retention_policy=3` files. You can use the `--additional-config` parameter to override the configurations in the *neo4j.conf* file.

Removing transaction logs manually can result in a broken backup.

Security configurations

Securing your backup network communication with an SSL policy and a firewall protects your data from unwanted intrusion and leakage. When using the `neo4j-admin backup` command, you can configure the backup server to require SSL/TLS, and the backup client to use a compatible policy. For more information on how to configure SSL in Neo4j, see SSL framework.

For a detailed list of recommendations regarding security in Neo4j, see Security checklist.

The following table provides details on how the configured SSL policies map to the configured ports.

Table 40. Mapping backup configurations to SSL policies

Topology	Backup target address on database server	SSL policy setting on database server	SSL policy setting on backup client	Default port

Standalone instance	`dbms.backup.listen_address`	`dbms.ssl.policy.backup`	`dbms.ssl.policy.backup`	`6362`
Causal cluster	`dbms.ssl.policy.cluster` `causal_clustering.transaction_listen_address`	`dbms.ssl.policy.cluster`	`dbms.ssl.policy.backup`	`6000`

It is very important to ensure that there is no external access to the port specified by the setting `dbms.backup.listen_address`. Failing to protect this port may leave a security hole open by which an unauthorized user can make a copy of the database onto a different machine. In production environments, external access to the backup port should be blocked by a firewall.

Cluster configurations

In a cluster topology, it is possible to take a backup from any server, and each server has two configurable ports capable of serving a backup. These ports are configured by `dbms.backup.listen.address` and `causal_clustering.transaction_listen_address` respectively. Functionally, they are equivalent for backups, but separating them can allow some operational flexibility, while using just a single port can simplify the configuration. It is generally recommended to select Read Replicas to act as backup servers, since they are more numerous than Core members in typical cluster deployments. Furthermore, the possibility of performance issues on a Read Replica, caused by a large backup, will not affect the performance or redundancy of the Core members. If a Read Replica is not available, then a Core can be selected based on factors, such as its physical proximity, bandwidth, performance, and liveness.

To avoid taking a backup from a cluster member that is lagging behind, you can look at the transaction IDs by exposing Neo4j metrics or via Neo4j Browser. To view the latest processed transaction IDs (and other metrics) in Neo4j Browser, type `:sysinfo` at the prompt.

9.3.3. Examples

The following are examples of how to back up a single database, e.g., the default database `neo4j`, and multiple databases, using the `neo4j-admin backup` command. The target directory */mnt/backups/neo4j* must exist before calling the command and the database(s) must be online.

Example 53. Use `neo4j-admin backup` to back up a single database.

```
bin/neo4j-admin backup --backup-dir=/mnt/backups/neo4j --database=neo4j
```

To backup several databases that match database pattern you can use name globbing. For example, to backup all databases that start with **n** you should run:

Example 54. Use `neo4j-admin backup` to back up multiple databases.

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

For a detailed example on how to back up and restore a database in a Causal cluster, see Back up and restore a database in Causal Cluster.

9.4. Restore a database backup

9.4.1. Command

A database backup or an **offline** database can be restored using the `restore` command of `neo4j-admin`. You must create the database (using `CREATE DATABASE` against the `system` database) after the restore operation finishes, unless you are replacing an existing database. `neo4j-admin restore` must be invoked as the `neo4j` user to ensure the appropriate file permissions.

For more information, see Administrative commands.

Syntax

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

Options

Option	Default	Description
`--from`		Path or paths to the database backup that is going to be restored. Every path may contain asterisks or question marks in the last subpath. Multiple paths can be separated by a comma, but paths themselves must not contain commas.
`--verbose`		Enables verbose output.
`--database`	`neo4j`	Name for the restored database.
`--force`		Replaces an existing database.

Option	Default	Description
`--move`		Moves the backup files to the destination, rather than copying. This makes the restoring process faster and with no need for extra disk space. However, for this procedure to work properly, backup and database files must be on the same filesystem. In case they are not, the command will only copy the files and delete the backup, resulting in no performance benefits.
`--to-data-directory`		Base directory for databases. Usage of this option is only allowed if the `--from` parameter points to one directory.
`--to-data-tx-directory`		Base directory for transaction logs. Usage of this option is only allowed if the `--from` parameter points to one directory.

9.4.2. Example

The following is an example of how to perform an online restore of the database backup created in the section Back up an online database, using the `neo4j-admin restore` command.

```
bin/neo4j-admin restore --from=/mnt/backups/neo4j --database=neo4j --force
```

Unless you are replacing an existing database, you must create the database (using `CREATE DATABASE` against the `system` database) after the restore operation finishes.

If you have backed up a database with the option `--include-metadata`, you have to restore the users and roles metadata manually.

From the *<neo4j-home>* directory, you run the Cypher script *data/scripts/databasename/restore_metadata.cypher*, which the `neo4j-admin restore` command outputs, using Cypher Shell:

Using `cat` (UNIX)

```
cat data/scripts/databasename/restore_metadata.cypher | bin/cypher-shell -u user -p password -a
ip_address:port -d system --param "database => 'databasename'"
```

Using `type` (Windows)

```
type data\scripts\databasename\restore_metadata.cypher | bin\cypher-shell.bat -u user -p password -a
ip_address:port -d system --param "database => 'databasename'"
```

For a detailed example on how to back up and restore a database in a Causal cluster, see Back up and restore a database in Causal Cluster.

`neo4j-admin restore` cannot be applied to the Fabric virtual database. It must be run directly on the databases that are part of the Fabric setup.

9.5. Back up an offline database

Remember to plan your backup carefully and to back up each of your databases, including the `system` database.

9.5.1. Command

A Neo4j database can be backed up in offline mode using the `dump` command of `neo4j-admin`.

Usage

The `neo4j-admin dump` command can be used for performing a full backup of an **offline** database. It dumps a database into a single-file archive, called *<database>.dump*. The command can be run only locally from an online or an offline Neo4j DBMS. It does not support SSL/TLS.

Syntax

```
neo4j-admin dump --database=<database>
                 --to=<destination-path>
                [--verbose]
```

Options

Option	Default	Description
`--database`	`neo4j`	Name of the database to dump.
`--to`		Destination (file or folder) of database dump.
`--verbose`		Enable verbose output.

9.5.2. Example

The following is an example of how to create a dump of the default database `neo4j`, called *neo4j-<timestamp>.dump*, using the `neo4j-admin dump` command. The target directory */dumps/neo4j* must exist before running the command and the database must be offline.

```
bin/neo4j-admin dump --database=neo4j --to=/dumps/neo4j/neo4j-<timestamp>.dump
```

`neo4j-admin dump` cannot be applied to the Fabric virtual database. It must be run directly on the databases that are part of the Fabric setup.

9.6. Restore a database dump

A database dump can be loaded to a Neo4j instance using the `load` command of `neo4j-admin`.

9.6.1. Command

The `neo4j-admin load` command loads a database from an archive created with the `neo4j-admin dump` command. The command can be run from an online or an offline Neo4j DBMS. If you are replacing an existing database, you have to shut it down before running the command. If you are not replacing an existing database, you must create the database (using `CREATE DATABASE` against the `system` database) after the load operation finishes. `neo4j-admin load` must be invoked as the `neo4j` user to ensure the appropriate file permissions.

Syntax

```
neo4j-admin load [--verbose]
                  --from=<archive-path>
                  --database=<database>
                 [--force]
                 [--info]
```

Options

Option	Default	Description
`--verbose`		Enable verbose output.
`--from`		Path to archive created with the `neo4j-admin dump` command.
`--database`	`neo4j`	Name for the loaded database.
`--force`		Replace an existing database.
`--info`		Print meta-data information about the archive file, such as, file count, byte count, and format of the load file.

9.6.2. Example

The following is an example of how to load the dump of the `neo4j` database created in the section Back up an offline database, using the `neo4j-admin load` command. When replacing an existing database, you have to shut it down before running the command.

```
bin/neo4j-admin load --from=/dumps/neo4j/neo4j-<timestamp>.dump --database=neo4j --force
```

Unless you are replacing an existing database, you must create the database (using `CREATE DATABASE` against the `system` database) after the load operation finishes.

When using the `load` command to seed a Causal Cluster, and a previous version of the database exists, you must delete it (using `DROP DATABASE`) first. Alternatively, you can stop the Neo4j instance and unbind it from the cluster using `neo4j-admin unbind` to remove its cluster state data. If you fail to DROP or unbind before loading the dump, that database's store files will be out of sync with its cluster state, potentially leading to logical corruptions. For more information, see Seed a cluster from a database backup (online).

`neo4j-admin load` cannot be applied to the Fabric virtual database. It must be run directly on the databases that are part of the Fabric setup.

9.7. Copy a database store

A user database or backup can be copied to a Neo4j instance using the `copy` command of `neo4j-admin`.

`neo4j-admin copy` is not supported for use on the `system` database.

In Fabric deployments, `neo4j-admin copy` cannot be applied to the Fabric virtual database. It must be run directly on the databases that are part of the Fabric setup.

It is important to note that `neo4j-admin copy` is an IOPS-intensive process. Using this process for upgrading or migration purposes can have significant performance implications, depending on your disc specification. It is therefore not appropriate for all use cases.

Estimating the processing time

Estimations for how long the `neo4j-admin copy` command will take can be made based upon the following:

- Neo4j, like many other databases, do IO in 8K pages.
- Your disc manufacturer will have a value for the maximum IOPS it can process.

For example, if your disc manufacturer has provided a maximum of 5000 IOPS, you can reasonably expect up to 5000 such page operations a second. Therefore, the maximal theoretical throughput you can expect is 40MB/s (or 144 GB/hour) on that disc. You may then assume that the best-case scenario for running `neo4j-admin copy` on that 5000 IOPS disc is that it will take at least 1 hour to process a 144 GB database. [10]

However, it is important to remember that the process must read 144 GB from the source database, and must also write to the target store (assuming the target store is of comparable size). Additionally, there are internal processes during the copy that will read/modify/write the store multiple times. Therefore, with an additional 144 GB of both read and write, the best-case scenario for running `neo4j-admin copy` on a 5000 IOPS disc is that it will take **at least 3 hours to process a 144 GB database**.

Finally, it is also important to consider that in almost all Cloud environments, the published IOPS value may not be the same as the actual value, or be able to continuously maintain the maximum possible IOPS. The real processing time for this example *could* be well above that estimation of 3 hours.

For detailed information about supported methods of upgrade and migration, see the Neo4j Upgrade and Migration Guide.

9.7.1. Command

`neo4j-admin copy` copies the data store of an existing **offline** database to a new database.

Usage

The `neo4j-admin copy` command can be used to clean up database inconsistencies, compact stores, and do a direct migration from Neo4j 3.5 to any 4.x version. It can process an optional set of filters, which you can use to remove any unwanted data before copying the database. The command also reclaims the unused space of a database and creates a defragmented copy of that database or backup in the destination Neo4j instance.

`neo4j-admin copy` copies the *data store* without the *schema* (indexes and constraints). However, if the database has a schema defined, the command will output Cypher statements, which you can run to recreate the indexes and constraints.

For a detailed example of how to reclaim unused space, see Reclaim unused space.
For a detailed example of how to back up a 3.5 database and use the `neo4j-admin copy` command to compact its store and migrate it to a 4.x Neo4j standalone instance, see Upgrade and Migration Guide → Tutorial: Back up and copy a database in a standalone instance.

`neo4j-admin copy` preserves the node IDs; however, the relationships get new IDs.

Syntax

```
neo4j-admin copy  [--verbose]
                  [--from-database=<database>]
                  [--from-path=<path>]
                  [--from-path-tx=<path>]
                   --to-database=<database>
                  [--neo4j-home-directory=<path>]
                  [--force]
                  [--compact-node-store]
                  [--to-format=<format>]
                  [--delete-nodes-with-labels=<label>[,<label>...]]
                  [--keep-only-node-properties=<label.property>[,<label.property>...]]
                  [--keep-only-nodes-with-labels=<label>[,<label>...]]
                  [--keep-only-relationship-
properties=<relationship.property>[,<relationship.property>...]]
                  [--skip-labels=<label>[,<label>...]]
                  [--skip-node-properties=<label.property>[,<label.property>...]]
                  [--skip-properties=<property>[,<property>...]]
                  [--skip-relationship-properties=<relationship.property>[,<relationship.property>...]]
                  [--skip-relationships=<relationship>[,<relationship>...]]
                  [--from-pagecache=<size>]
                  [--to-pagecache=<size>]
```

Options

Option	Description
`--verbose`	Enable verbose output.
`--from-database`	The database name to copy from.
`--from-path`	The path to the database to copy from. It can be used to target databases outside of the installation, e.g., backups.
`--from-path-tx`	The path to the transaction log files. Use if the command cannot determine where they are located.

Option	Description
`--to-database`	The destination database name.
`--neo4j-home-directory=<path>`	Path to the home directory for the copied database. Default: The same as the database copied from.
`--force`	Force the command to proceed even if the integrity of the database can not be verified.
`--compact-node-store`	Enforce node store compaction. By default, the node store is not compacted on copy since it changes the node IDs.  When using Neo4j 4.2, this option is only available in minor releases ≥4.2.10.
`--to-format`	Set the format for the new database. Valid values are `same`, `standard`, `high_limit`, and `aligned`. The `high_limit` format is only available in Enterprise Edition. If you go from `high_limit` to `standard`, there is no validation that the data will fit. Default: The format of the source database.
`--delete-nodes-with-labels`	A comma-separated list of labels. All nodes that have ANY of the specified labels will be deleted. Any node matching any of the labels will be ignored during copy.
`--keep-only-node-properties`	A list of property keys to keep for nodes with the specified label. Any labels not explicitly mentioned will keep their properties. Cannot be combined with `--skip-properties` or `--skip-node-properties`.
`--keep-only-nodes-with-labels`	A list of labels. All nodes that have any of the specified labels will be kept. Cannot be combined with `--delete-nodes-with-labels`.

Option	Description
`--keep-only-relationship-properties`	A list of property keys to keep for relationships with the specified type. Any relationship types not explicitly mentioned will keep their properties. Cannot be combined with `--skip-properties` or `--skip-relationship-properties`.
`--skip-labels`	A comma-separated list of labels to ignore during the copy.
`--skip-node-properties`	A list of property keys to ignore for nodes with the specified label. Cannot be combined with `--skip-properties` or `--keep-only-node-properties`.
`--skip-properties`	A comma-separated list of property keys to ignore during the copy. Cannot be combined with `--skip-node-properties`, `--keep-only-node-properties`, `--skip-relationship-properties`, and `--keep-only-relationship-properties`.
`--skip-relationships`	A comma-separated list of relationship types to ignore during the copy.
`--skip-relationship-properties`	A list of property keys to ignore for relationships with the specified type. Cannot be combined with `--skip-properties` or `--keep-only-relationship-properties`.
`--from-pagecache`	The size of the page cache to use for reading.
`--to-pagecache`	The size of the page cache to use for writing.

You can use the `--from-pagecache` and `--to-pagecache` options to speed up the copy operation by specifying how much cache to allocate when reading the source and writing the destination. As a rule of thumb, `--to-pagecache` should be around `1-2GB` since it mostly does sequential writes. The `--from-pagecache` should then be assigned whatever memory you can spare since Neo4j does random reads from the source.

9.7.2. Examples

Example 55. Use `neo4j-admin copy` to copy the data store of the database `neo4j`.

1. Stop the database named `neo4j`:

```
STOP DATABASE neo4j
```

2. Copy the data store from `neo4j` to a new database called `copy`:

```
bin/neo4j-admin copy --from-database=neo4j --to-database=copy
```

3. Run the following command to verify that database has been successfully copied.

```
ls -al ../data/databases
```

Copying a database does not automatically create it. Therefore, it will not be visible if you do `SHOW DATABASES` at this point.

4. Create the copied database.

```
CREATE DATABASE copy
```

5. Verify that the `copy` database is online.

```
SHOW DATABASES
```

6. If your original database has a schema defined, change your active database to `copy` and recreate the schema using the `neo4j-admin copy` output.

The console output is saved to *logs/neo4j-admin-copy-<timestamp>.log*.

Example 56. Use `neo4j-admin copy` to filter the data you want to copy.

The command can perform some basic forms of processing. You can filter the data that you want to copy by removing nodes, labels, properties, and relationships.

```
bin/neo4j-admin copy --from-database=neo4j --to-database=copy --delete-nodes-with-labels="Cat,Dog"
```

The command creates a copy of the database `neo4j` but without the nodes with the labels `:Cat` and `:Dog`.

Labels are processed independently, i.e., the filter deletes any node with a label `:Cat`, `:Dog`, or both.

For a detailed example of how to use `neo4j-admin copy` to filter out data for a Fabric installation, see Sharding data with the copy command.

[10] The calculations are based on `MB/s = (IOPS * B) ÷ 10^6`, where `B` is the block size in bytes; in the case of Neo4j, this is `8000`. GB/hour can then be calculated from `(MB/s * 3600) ÷ 1000`.

Chapter 10. Authentication and authorization

Ensure that your Neo4j deployment adheres to your company's information security guidelines by setting up the appropriate authentication and authorization rules.

Ths section describes the following:

- Introduction
- Built-in roles
- Fine-grained access control
- Integration with LDAP directory services
- Manage procedure and user-defined function permissions
- Terminology

The functionality described in this section is applicable to Enterprise Edition. A limited set of user management functions are also available in Community Edition. Native roles overview gives a quick overview of these.

10.1. Introduction

Authentication is the process of ensuring that a user is who the user claims to be, while authorization pertains to checking whether the authenticated user is allowed to perform a certain action. Authorization is managed using role-based access control (*RBAC*). Permissions that define access control are assigned to roles, which are in turn assigned to users.

Neo4j has the following auth providers, that can perform user authentication and authorization:

Native auth provider

Neo4j provides a native auth provider that stores user and role information in the `system` database. The following parameters control this provider:

- `dbms.security.auth_enabled` (Default: `true`) — Enable auth requirement to access Neo4j.

If you need to disable authentication, for example, to recover an `admin` user password or assign a user to the `admin` role, make sure you block all network connections during the recovery phase so users can connect to Neo4j only via `localhost`. For more information, see Password and user recovery.

- `dbms.security.auth_lock_time` (Default: `5s`) — The amount of time a user account is locked after a configured number of unsuccessful authentication attempts.
- `dbms.security.auth_max_failed_attempts` (Default: `3`) — The maximum number of unsuccessful authentication attempts before imposing a user lock for a configured amount of time.
  When triggered, Neo4j logs an error containing a timestamp and the message `failed to log in: too many failed attempts` in the *security.log*.

The Cypher commands to manage users, roles, and permissions are described in detail in Cypher Manual → Administration. Various scenarios that illustrate the use of the native auth provider are available in Fine-grained access control.

LDAP auth provider

Another way of controlling authentication and authorization is through external security software such as Active Directory or OpenLDAP, which is accessed via the built-in LDAP connector. A description of the LDAP plugin using Active Directory is available in Integration with LDAP directory services.

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.

10.2. Built-in roles

Neo4j provides built-in roles with default privileges. The built-in roles and the default privileges are:

`PUBLIC`

- Access to the default database.
- Allows executing procedures with the users own privileges.
- Allows executing user-defined functions with the users own privileges.

`reader`

- Access to all databases.

- Traverse and read on the data graph (all nodes, relationships, properties).

`editor`

- Access to all databases.
- Traverse, read, and write on the data graph.
- Write access limited to creating and changing **existing** property keys, node labels, and relationship types of the graph. In other words, the `editor` role cannot add to the schema but can only make changes to already existing objects.

`publisher`

- Access to all databases.
- Traverse, read, and write on the data graph.

`architect`

- Access to all databases.
- Traverse, read, and write on the data graph.
- Create/drop/show indexes and constraints along with any other future schema constructs.

`admin`

- Access to all databases.
- Traverse, read, and write on the data graph.
- Create/drop/show indexes and constraints along with any other future schema constructs.
- Allows executing procedures with the users own privileges or boosted privileges.
- Allows executing admin procedures.
- Allows executing user-defined functions with the users own privileges or boosted privileges.
- View/terminate queries.
- Manage databases, users, roles, and privileges.

All users will be assigned the `PUBLIC` role, which by default does not give any rights or capabilities regarding the data, not even read privileges. A user may have more than one assigned role, and the union of these determine what action(s) on the data may be undertaken by the user. For instance, a user assigned to the `reader` role will be able to execute procedures because all users are also assigned to the `PUBLIC` role, which enables that capability.

When an administrator suspends or deletes another user, the following rules apply:

- Administrators can suspend or delete any other user (including other administrators), but not themselves.
- The user will no longer be able to log back in (until re-activated by an administrator if suspended).
- There is no need to remove assigned roles from a user prior to deleting the user.

Deleting a user will not automatically terminate associated connections, sessions, transactions, or queries.

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

Table 41. Native roles overview

Action	reader	editor	publisher	architect	admin	PUBLIC	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
Execute procedures					X	X	X
Execute functions					X	X	X
Execute admin procedures					X		X
View own queries	X	X	X	X	X		
Terminate own queries	X	X	X	X	X		
Write/update/delete existing data		X	X	X	X		X
Create new types of properties key			X	X	X		X
Create new types of nodes labels			X	X	X		X
Create new types of relationship types			X	X	X		X
Create/drop/show index/constraint				X	X		X
Create/delete user					X		X

Action	reader	editor	publisher	architect	admin	PUBLIC	Available in Community Edition
Change another user's password					X		X
Suspend/activate user					X		
Create/drop roles					X		
Assign/remove role to/from user					X		
Create/drop databases					X		
Start/stop databases					X		
Manage database access					X		
Access default database	X	X	X	X	X	X	X
Access all databases	X	X	X	X	X		X
View all users					X		X
View all roles					X		
View all roles for a user					X		
View all users for a role					X		
View all queries					X		
View all databases					X		
View own privileges	X	X	X	X	X	X	
View another user's privileges					X		
Grant/deny/revoke privileges					X		

Action	reader	editor	publisher	architect	admin	PUBLIC	Available in Community Edition
Terminate all queries					X		
Dynamically change configuration (see Dynamic settings)					X		

More information about the built-in roles and their privileges can be found in Neo4j Cypher Manual.

10.3. Recover admin user and password

This page describes how to reset a password to recover a user's access when their password is lost. It specifically focuses on how to recover an admin user if all the admin users have been unassigned the admin role, and how to recreate the built-in admin role if it has been dropped.

10.3.1. Disable authentication

1. Stop Neo4j:

```
$ bin/neo4j stop
```

2. Open the *neo4j.conf* file and set `dbms.security.auth_enabled` parameter to `false` to disable the authentication:

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

It is recommended to block network connections during the recovery phase, so users can connect to Neo4j only via `localhost`. This can be achieved by either:

- Temporarily commenting out the `dbms.default_listen_address` parameter:

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

or

- Providing the specific localhost value:

```
dbms.default_listen_address=127.0.0.1
```

3. Start Neo4j:

```
$ bin/neo4j start
```

1. Stop the cluster (all Core servers and Read Replicas).

```
$ bin/neo4j stop
```

2. On each Core server, open the *neo4j.conf* file and modify the following settings:

   a. Set `dbms.security.auth_enabled` parameter to `false` to disable the authentication:

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

   b. Disable the HTTP and HTTPS network connections and restrict the `bolt` connector to use only `localhost`. This ensures that no one from outside can access the cluster during the recovery period.

```
#dbms.connector.http.enabled=true
#dbms.connector.https.enabled=true
dbms.connector.bolt.listen_address:127.0.0.1
```

3. Start all Core servers:

```
$ bin/neo4j start
```

10.3.2. Recover a lost password

You can use a client such as Cypher Shell or the Neo4j Browser to connect to the `system` database and set a new password for the admin user.

In a cluster deployment, you should complete the steps only on one of the Core servers.

1. Complete the steps in Disable authentication as per your deployment.
2. Connect to the `system` database using Cypher shell. Alternatively, log into Neo4j Browser.

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

Cluster If you have specified a non-default port for your `bolt` connector, add `-a neo4j://<your-core>:<non-default-bolt-port>` to the `cypher-shell` command to be able to connect to your Core server.

3. Set a new password for the admin user. In this example, the admin user is named `neo4j`.

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

4. Exit the `cypher-shell` console:

```
:exit;
```

5. Proceed with the post-recovery steps as per your deployment.

10.3.3. Recover an unassigned admin role

You can use a client such as Cypher Shell or the Neo4j Browser to connect to the `system` database and grant the admin user role to an existing user.

In a cluster deployment, you should complete the steps only on one of the Core servers.

1. Complete the steps in Disable authentication as per your deployment.
2. Connect to the `system` database using Cypher shell. Alternatively, log into Neo4j Browser.

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

Cluster If you have specified a non-default port for your `bolt` connector, add `-a neo4j://<your-core>:<non-default-bolt-port>` to the `cypher-shell` command to be able to connect to your Core server.

3. Grant the admin user role to an existing user. In this example, the user is named `neo4j`.

```
GRANT ROLE admin TO neo4j
```

4. Exit the `cypher-shell` console:

```
:exit;
```

5. Proceed with the post-recovery steps as per your deployment.

10.3.4. Recover the admin role

If you have removed the admin role from your system entirely, you can use a client such as Cypher Shell or the Neo4j Browser to connect to the `system` database and recreate the role with its original capabilities.

In a cluster deployment, you should complete the steps only on one of the Core servers.

1. Complete the steps in Disable authentication as per your deployment.
2. Connect to the `system` database using Cypher shell. Alternatively, log into Neo4j Browser.

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

Cluster If you have specified a non-default port for your `bolt` connector, add `-a neo4j://<your-core>:<non-default-bolt-port>` to the `cypher-shell` command to be able to connect to your Core server.

3. Recreate the admin role with its original capabilities.

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

4. Grant the admin user role to an existing user.

Before running the `:exit` command, we suggest granting the newly created role to a user. Although this is optional, without this step you will have only collected all admin privileges in a role that no one is assigned to.

To grant the role to a user (assuming your existing user is named `neo4j`), you can run `GRANT ROLE admin TO neo4j;`

5. Exit the `cypher-shell` console:

```
:exit;
```

6. Proceed with the post-recovery steps as per your deployment.

10.3.5. Post-recovery steps

1. Stop Neo4j:

```
$ bin/neo4j stop
```

2. Enable the authentication and restore your Neo4j to its original configuration (See Disable authentication).
3. Start Neo4j:

```
$ bin/neo4j start
```

1. Stop the Core servers.

```
$ bin/neo4j stop
```

2. Enable the authentication and restore each Core server to its original configuration (See Disable authentication).
3. Start the cluster (all Core servers and Read Replicas):

```
$ bin/neo4j start
```

10.4. Fine-grained access control

10.4.1. The data model

Consider a *healthcare* database, as could be relevant in a medical clinic or hospital. A simple version of this might contain only three labels, representing three entity types:

`(:Patient)`

Nodes of this type represent patients that visit the clinic because they have some symptoms. Information specific to the patient can be captured in properties:

- `name`
- `ssn`
- `address`
- `dateOfBirth`

`(:Symptom)`

A medical database contains a catalog of known illnesses and associated symptoms, which can be described using properties:

- `name`
- `description`

`(:Disease)`

A medical database contains a catalog of known illnesses and associated symptoms, which can be described using properties:

- `name`
- `description`

These entities will be modelled as nodes, and connected using relationships of the following types:

`(:Patient)-[:HAS]→(:Symptom)`

When a patient reports to the clinic, they will describe their symptoms to the nurse or the doctor. The nurse or doctor will then enter this information into the database in the form of connections between the patient node and a graph of known symptoms. Possible properties of interest on this relationship could be:

- `date` - date when symptom was reported

`(:Symptom)-[:OF]→(:Disease)`

The graph of known symptoms is part of a graph of diseases and their symptoms. The relationship between a symptom and a disease can include a probability factor for how likely or common it is for people with that disease to express that symptom. This will make it easier for the doctor to make a diagnosis using statistical queries.

- `probability` - probability of symptom matching disease

`(:Patient)-[:DIAGNOSIS]→(:Disease)`

The doctor can use the graph of diseases and their symptoms to perform an initial investigation into the most likely diseases to match the patient. Based on this, and their own assessment of the patient, they may make a diagnosis which they would persist to the graph through the addition of this relationship with appropriate properties:

- `by`: doctor's name
- `date`: date of diagnosis
- `description`: additional doctors' notes

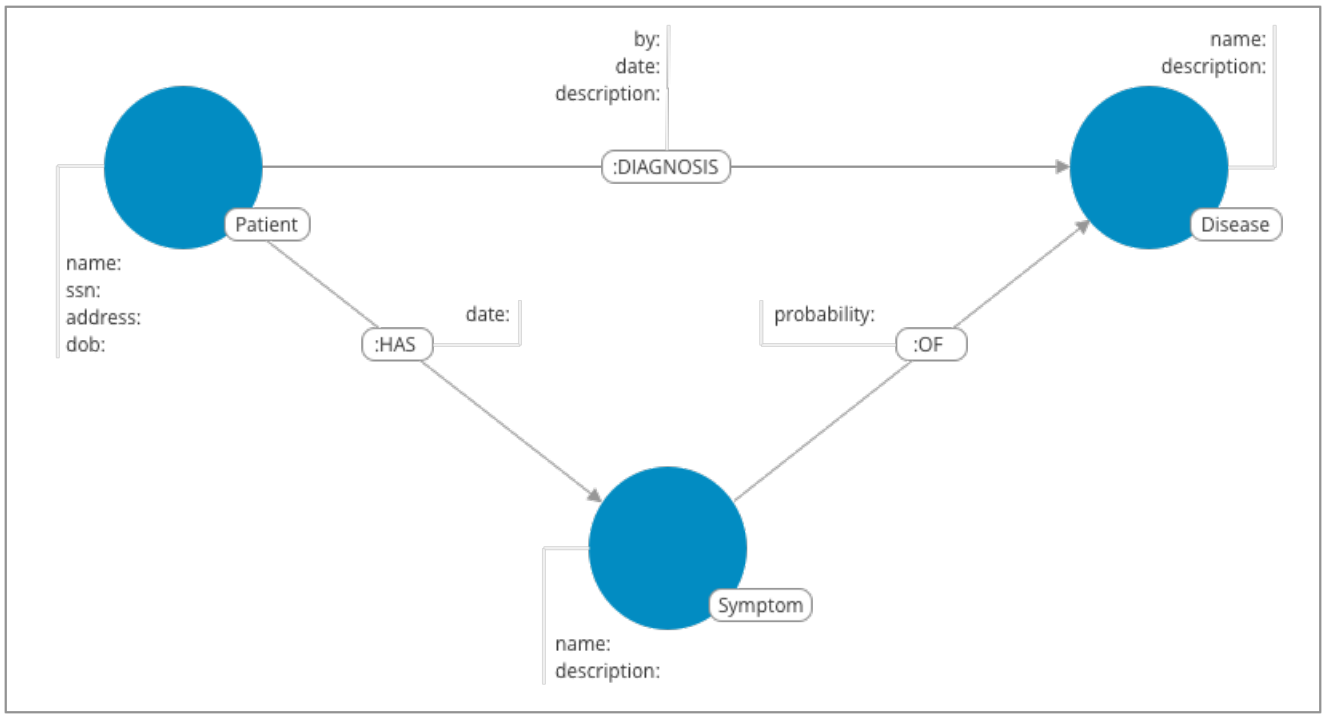


Figure 12. Healthcare use case

The database would be used by a number of different user types, with different needs for access.

- **Doctors** who need to perform diagnosis on patients.
- **Nurses** who need to treat patients.
- **Receptionists** who need to identify and record patient information.
- **Researchers** who need to perform statistical analysis of medical data.
- **IT administrators** who need to administer the database, creating and assigning users.

10.4.2. Security

When building an application for a specific domain, it is common to model the different users within the application itself. However, when working with a database that provides rich user management with roles and privileges, it is possible to model these entirely within the database security model (for more information, see Cypher Manual → Administration → Security). This results in separation of concerns for the access control to the data and the data itself. We will show two approaches to using Neo4j security features to support the *healthcare* database application. First, a simple approach using built-in roles, and then a more advanced approach using fine-grained privileges for sub-graph access control.

Our *healthcare* example involves five users of the database:

- Alice the doctor
- Daniel the nurse
- Bob the receptionist
- Charlie the researcher
- Tina the IT administrator

These users can be created using the `CREATE USER` command (from the `system` database):

Example 57. Creating users

```
CREATE USER charlie SET PASSWORD $secret1 CHANGE NOT REQUIRED;
CREATE USER alice SET PASSWORD $secret2 CHANGE NOT REQUIRED;
CREATE USER daniel SET PASSWORD $secret3 CHANGE NOT REQUIRED;
CREATE USER bob SET PASSWORD $secret4 CHANGE NOT REQUIRED;
CREATE USER tina SET PASSWORD $secret5 CHANGE NOT REQUIRED;
```

At this point the users have no ability to interact with the database, so we need to grant those capabilities using roles. There are two different ways of doing this, either by using the built-in roles, or through more fine-grained access control using privileges and custom roles.

10.4.3. Access control using built-in roles

Neo4j comes with a number of built-in roles that cover a number of common needs:

- `PUBLIC` - All users have this role, can by default access the default database and run all procedures and user-defined functions.
- `reader` - Can read data from all databases.
- `editor` - Can read and update all databases, but not expand the schema with new labels, relationship types or property names.
- `publisher` - Can read and edit, as well as add new labels, relationship types and property names.
- `architect` - Has all the capabilities of the publisher as well as the ability to manage indexes and constraints.
- `admin` - Can perform architect actions as well as manage databases, users, roles and privileges.

Charlie is a researcher and will not need write access to the database, and so he is assigned the `reader` role. Alice the doctor, Daniel the nurse and Bob the receptionist all need to update the database with new patient information, but do not need to expand the schema with new labels, relationship types, property names or indexes. We assign them all the `editor` role. Tina is the IT administrator that installs and manages the database. In order to create all other users, Tina is assigned the `admin` role.

Example 58. Granting roles

```
GRANT ROLE reader TO charlie;
GRANT ROLE editor TO alice;
GRANT ROLE editor TO daniel;
GRANT ROLE editor TO bob;
GRANT ROLE admin TO tina;
```

A limitation of this approach is that it does allow all users to see all data in the database, and in many real-world scenarios it would be preferable to restrict the users' access. In this example, we would want to restrict the researcher from being able to read any of the patients' personal information, and the receptionist should only be able to see the patient records and nothing more.

These, and more restrictions, could be coded into the application layer. However, it is possible and more secure to enforce these kinds of fine-grained restrictions directly within the Neo4j security model, by creating custom roles and assigning specific privileges to those roles.

Since we will be creating new custom roles, the first thing to do is revoke the current roles from the users:

Example 59. Revoking roles

```
REVOKE ROLE reader FROM charlie;
REVOKE ROLE editor FROM alice;
REVOKE ROLE editor FROM daniel;
REVOKE ROLE editor FROM bob;
REVOKE ROLE admin FROM tina;
```

Now the users are once again unable to do anything, and so we need to start over by building the set of new privileges based on a complete understanding of what we want each user to be able to do.

10.4.4. Sub-graph access control using privileges

With the concept of *privileges*, we can take much more control over what each user is capable of doing. We start by identifying each type of user:

Doctor

Should be able to read and write most of the graph. We would, however, like to prevent the doctor from reading the patient's address. We would also like to make sure the doctor can save *diagnoses* to the database, but not expand the schema of the database with new concepts.

Receptionist

Should be able to read and write all patient data, but not be able to see the symptoms, diseases or diagnoses.

Researcher

Should be able to perform statistical analysis on all data, except patients' personal information, and as such should not be able to read most patient properties. To illustrate two different ways of setting up the same effective privileges, we will create two roles and compare them.

Nurse

The nurse should be able to perform all tasks that both the doctor and the receptionist can do. At first one might be tempted to simply grant both those roles, but this does not work as expected. We will demonstrate why below, and instead create a dedicated `nurse` role.

Junior nurse

The senior nurse above is able to save diagnoses just as a doctor can. However, we might wish to have nurses that are not allowed to make that update to the graph. While we could build another role from scratch, this could more easily be achieved by combining the `nurse` role with a new `disableDiagnoses` role that specifically restricts that activity.

IT administrator

This role is very similar to the built-in `admin` role, except that we wish to restrict access to the patients `SSN`, as well as prevent the administrator from performing the very critical action of saving a diagnosis, something specific to medical professionals. To achieve this, we can create this role by copying the built-in `admin` role and modifying the privileges of that copy.

User manager

It is possible that we would like the IT administrator to be less powerful than described above. We can create a new role from scratch, granting only the specific administrative capabilities we actually desire.

Before we create the new roles and assign them to Alice, Bob, Daniel, Charlie and Tina, we should define the privileges of each role. Since all users need `ACCESS` privilege to the `healthcare` database, we can add this to the `PUBLIC` role instead of all the individual roles:

```
GRANT ACCESS ON DATABASE healthcare TO PUBLIC;
```

Privileges of `itadmin`

This role can be created as a copy of the built-in `admin` role:

```
CREATE ROLE itadmin AS COPY OF admin;
```

Then all we need to do is deny the two specific actions this role is not supposed to do:

- Should not be able to read any patients social security number.
- Should not be able to perform medical diagnosis.

```
DENY READ {ssn} ON GRAPH healthcare NODES Patient TO itadmin;
DENY CREATE ON GRAPH healthcare RELATIONSHIPS DIAGNOSIS TO itadmin;
```

The complete set of privileges available to users assigned the `itadmin` role can be viewed using the following command:

```
SHOW ROLE itadmin PRIVILEGES;
```

```
+-----------------------------------------------------------------------------------------------------
----+
| access    | action           | resource         | graph        | segment                    | role
|
+-----------------------------------------------------------------------------------------------------
----+
| "GRANTED" | "match"          | "all_properties" | "*"          | "NODE(*)"                  |
"itadmin" |
| "GRANTED" | "write"          | "graph"          | "*"          | "NODE(*)"                  |
"itadmin" |
| "GRANTED" | "match"          | "all_properties" | "*"          | "RELATIONSHIP(*)"          |
"itadmin" |
| "GRANTED" | "write"          | "graph"          | "*"          | "RELATIONSHIP(*)"          |
"itadmin" |
| "GRANTED" | "access"         | "database"       | "*"          | "database"                 |
"itadmin" |
| "GRANTED" | "admin"          | "database"       | "*"          | "database"                 |
"itadmin" |
| "GRANTED" | "constraint"     | "database"       | "*"          | "database"                 |
"itadmin" |
| "GRANTED" | "index"          | "database"       | "*"          | "database"                 |
"itadmin" |
| "GRANTED" | "token"          | "database"       | "*"          | "database"                 |
"itadmin" |
| "DENIED"  | "read"           | "property(ssn)"  | "healthcare" | "NODE(Patient)"            |
"itadmin" |
| "DENIED"  | "create_element" | "graph"          | "healthcare" | "RELATIONSHIP(DIAGNOSIS)" |
"itadmin" |
+-----------------------------------------------------------------------------------------------------
----+
```

Privileges that were granted or denied earlier can be revoked using the `REVOKE` command. See the Cypher Manual → The `REVOKE` command.

In order for the IT administrator `tina` to be provided these privileges, she must be assigned the new role `itadmin`.

```
neo4j@system> GRANT ROLE itadmin TO tina;
```

To demonstrate that Tina is not able to see the patients `SSN`, we can login to `healthcare` as `tina` and run the query:

```
MATCH (n:Patient)
 WHERE n.dateOfBirth < date('1972-06-12')
RETURN n.name, n.ssn, n.address, n.dateOfBirth;
```

```
+-------------------------------------------------------------------+
| n.name          | n.ssn | n.address               | n.dateOfBirth |
+-------------------------------------------------------------------+
| "Mary Stone"    | NULL  | "1 secret way, downtown" | 1970-01-15    |
| "Ally Anderson" | NULL  | "1 secret way, downtown" | 1970-08-20    |
| "Sally Stone"   | NULL  | "1 secret way, downtown" | 1970-03-12    |
| "Jane Stone"    | NULL  | "1 secret way, downtown" | 1970-07-21    |
| "Ally Svensson" | NULL  | "1 secret way, downtown" | 1971-08-15    |
| "Jane Svensson" | NULL  | "1 secret way, downtown" | 1972-05-12    |
| "Ally Svensson" | NULL  | "1 secret way, downtown" | 1971-07-30    |
+-------------------------------------------------------------------+
```

The results make it seem as if these nodes do not even have an `ssn` field. This is a key feature of the security model, that users cannot tell the difference between data that is not there, and data that is hidden using fine-grained read privileges.

Now remember that we also denied the administrator from saving diagnoses, because that is a critical medical function reserved for only doctors and senior medical staff. We can test this by trying to create `DIAGNOSIS` relationships:

```
MATCH (n:Patient), (d:Disease)
CREATE (n)-[:DIAGNOSIS]->(d);
```

```
Create relationship with type 'DIAGNOSIS' is not allowed for user 'tina' with roles [PUBLIC, itadmin].
```

While restrictions on reading data do not result in errors and only make it appear as if the data is not there, restrictions on updating, i.e. writing to the graph will produce an appropriate error when the user attempts to perform an update they are not permitted to do.

Privileges of `researcher`

Charlie the researcher was previously our only read-only user. We could do something similar to what we did with the `itadmin` role, by copying and modifying the `reader` role. However, we would like to explicitly illustrate how to build a role from scratch. There are various possibilities for building this role using the concepts of either granting or denying a list of privileges:

- **Denying privileges**:

  We could grant the role the ability to find all nodes and read all properties (much like the `reader` role) and then deny read access to the `Patient` properties we want to restrict the researcher from seeing, such as `name`, `SSN` and `address`. This approach is simple but suffers from one problem. If `Patient` nodes are assigned additional properties, *after* we have restricted access, these new properties will automatically be visible to the researcher, which may not be desirable.

Example 60. Denying specific privileges

```
// First create the role
CREATE ROLE researcherB;
// Then grant access to everything
GRANT MATCH {*}
    ON GRAPH healthcare
    TO researcherB;
// And deny read on specific node properties
DENY READ {name, address, ssn}
    ON GRAPH healthcare
    NODES Patient
    TO researcherB;
// And finally deny traversal of the doctors diagnosis
DENY TRAVERSE
    ON GRAPH healthcare
    RELATIONSHIPS DIAGNOSIS
    TO researcherB;
```

- Granting privileges:

  An alternative is to only provide specific access to the properties we wish the researcher to see. Then, the addition of new properties will not automatically make them visible to the researcher. In this case, adding new properties to a `Patient` will not mean that the researcher can see them by default. If we wish to have them visible, we need to explicitly grant read access.

  Example 61. Granting specific privileges

```
// Create the role first
CREATE ROLE researcherW
// We allow the researcher to find all nodes
GRANT TRAVERSE
    ON GRAPH healthcare
    NODES *
    TO researcherW;
// Now only allow the researcher to traverse specific relationships
GRANT TRAVERSE
    ON GRAPH healthcare
    RELATIONSHIPS HAS, OF
    TO researcherW;
// Allow reading of all properties of medical metadata
GRANT READ {*}
    ON GRAPH healthcare
    NODES Symptom, Disease
    TO researcherW;
// Allow reading of all properties of the disease-symptom relationship
GRANT READ {*}
    ON GRAPH healthcare
    RELATIONSHIPS OF
    TO researcherW;
// Only allow reading dateOfBirth for research purposes
GRANT READ {dateOfBirth}
    ON GRAPH healthcare
    NODES Patient
    TO researcherW;
```

In order to test that Charlie now has the privileges we have specified, we assign him to the `researcherB` role (with specifically denied privileges):

```
GRANT ROLE researcherB TO charlie;
```

We can use a version of the SHOW PRIVILEGES command to see Charlies access rights:

```
neo4j@system> SHOW USER charlie PRIVILEGES;
```

```
+-----------------------------------------------------------------------------------------------------
-----------------+
| access    | action     | resource            | graph        | segment                      | role
| user      |
+-----------------------------------------------------------------------------------------------------
-----------------+
| "GRANTED" | "access"   | "database"          | "DEFAULT"    | "database"                   | "PUBLIC"
| "charlie" |
| "GRANTED" | "access"   | "database"          | "healthcare" | "database"                   | "PUBLIC"
| "charlie" |
| "GRANTED" | "execute"  | "database"          | "*"          | "FUNCTION(*)"                | "PUBLIC"
| "charlie" |
| "GRANTED" | "execute"  | "database"          | "*"          | "PROCEDURE(*)"               | "PUBLIC"
| "charlie" |
| "GRANTED" | "match"    | "all_properties"    | "healthcare" | "NODE(*)"                    |
"researcherB" | "charlie" |
| "DENIED"  | "read"     | "property(address)" | "healthcare" | "NODE(Patient)"              |
"researcherB" | "charlie" |
| "DENIED"  | "read"     | "property(name)"    | "healthcare" | "NODE(Patient)"              |
"researcherB" | "charlie" |
| "DENIED"  | "read"     | "property(ssn)"     | "healthcare" | "NODE(Patient)"              |
"researcherB" | "charlie" |
| "GRANTED" | "match"    | "all_properties"    | "healthcare" | "RELATIONSHIP(*)"            |
"researcherB" | "charlie" |
| "DENIED"  | "traverse" | "graph"             | "healthcare" | "RELATIONSHIP(DIAGNOSIS)" |
"researcherB" | "charlie" |
+-----------------------------------------------------------------------------------------------------
-----------------+
```

Now when Charlie logs into the healthcare database and tries to run a command similar to the one used by the itadmin above, we will see different results:

```
MATCH (n:Patient)
 WHERE n.dateOfBirth < date('1972-06-12')
RETURN n.name, n.ssn, n.address, n.dateOfBirth;
```

```
+---------------------------------------------+
| n.name | n.ssn | n.address | n.dateOfBirth |
+---------------------------------------------+
| NULL   | NULL  | NULL      | 1971-05-31    |
| NULL   | NULL  | NULL      | 1971-04-17    |
| NULL   | NULL  | NULL      | 1971-12-27    |
| NULL   | NULL  | NULL      | 1970-02-13    |
| NULL   | NULL  | NULL      | 1971-02-04    |
| NULL   | NULL  | NULL      | 1971-05-10    |
| NULL   | NULL  | NULL      | 1971-02-21    |
+---------------------------------------------+
```

Only the date of birth is available, so Charlie the researcher may perform statistical analysis, for example. Another query Charlie could try is to find the ten diseases a patient younger than 25 is most likely to be diagnosed with, listed by probability:

```
WITH datetime() - duration({years:25}) AS timeLimit
MATCH (n:Patient)
WHERE n.dateOfBirth > date(timeLimit)
MATCH (n)-[h:HAS]->(s:Symptom)-[o:OF]->(d:Disease)
WITH d.name AS disease, o.probability AS prob
RETURN disease, sum(prob) AS score ORDER BY score DESC LIMIT 10;
```

```
+-----------------------------------------+
| disease               | score           |
+-----------------------------------------+
| "Acute Argitis"       | 95.05395287286318 |
| "Chronic Someitis"    | 88.7220337139605  |
| "Chronic Placeboitis" | 88.43609533058974 |
| "Acute Whatitis"      | 83.23493746472457 |
| "Acute Otheritis"     | 82.46129768949129 |
| "Chronic Otheritis"   | 82.03650063794025 |
| "Acute Placeboitis"   | 77.34207326583929 |
| "Acute Yellowitis"    | 76.34519967465832 |
| "Chronic Whatitis"    | 73.73968070128234 |
| "Chronic Yellowitis"  | 71.58791287376775 |
+-----------------------------------------+
```

Now if we revoke the `researcherB` and instead grant the `researcherW` role to Charlie, and re-run these queries, we will see the same results.

Privileges that were granted or denied earlier can be revoked using the `REVOKE` command. See the Cypher Manual → The `REVOKE` command.

Privileges of `doctor`

Doctors should be given the ability to read and write almost everything. We would, however, like to remove the ability to read the patients' `address` property. This role can be built from scratch by assigning full read and write access, and then specifically denying access to the `address` property:

```
CREATE ROLE doctor;
GRANT TRAVERSE ON GRAPH healthcare TO doctor;
GRANT READ {*} ON GRAPH healthcare TO doctor;
GRANT WRITE ON GRAPH healthcare TO doctor;
DENY READ {address} ON GRAPH healthcare NODES Patient TO doctor;
DENY SET PROPERTY {address} ON GRAPH healthcare NODES Patient TO doctor;
```

To allow Alice to have these privileges, we grant her this new role:

```
neo4j@system> GRANT ROLE doctor TO alice;
```

To demonstrate that Alice is not able to see patient addresses, we log in as `alice` to `healthcare` and run the query:

```
MATCH (n:Patient)
 WHERE n.dateOfBirth < date('1972-06-12')
RETURN n.name, n.ssn, n.address, n.dateOfBirth;
```

```
+-----------------------------------------------------+
| n.name          | n.ssn   | n.address | n.dateOfBirth |
+-----------------------------------------------------+
| "Jack Anderson" | 1234647 | NULL      | 1970-07-23    |
| "Joe Svensson"  | 1234659 | NULL      | 1972-06-07    |
| "Mary Jackson"  | 1234568 | NULL      | 1971-10-19    |
| "Jack Jackson"  | 1234583 | NULL      | 1971-05-04    |
| "Ally Smith"    | 1234590 | NULL      | 1971-12-07    |
| "Ally Stone"    | 1234606 | NULL      | 1970-03-29    |
| "Mark Smith"    | 1234610 | NULL      | 1971-03-30    |
+-----------------------------------------------------+
```

As we can see, the doctor has the expected privileges, including being able to see the SSN, but not the address of each patient.

The doctor is also able to see all other node types:

```
MATCH (n) WITH labels(n) AS labels
RETURN labels, count(*);
```

```
+------------------------+
| labels      | count(*) |
+------------------------+
| ["Patient"] | 101      |
| ["Symptom"] | 10       |
| ["Disease"] | 12       |
+------------------------+
```

In addition, the doctor can traverse the graph, finding symptoms and diseases connected to patients:

```
MATCH (n:Patient)-[:HAS]->(s:Symptom)-[:OF]->(d:Disease)
  WHERE n.ssn = 1234657
RETURN n.name, d.name, count(s) AS score ORDER BY score DESC;
```

The resulting table shows which are the most likely diagnoses based on symptoms. The doctor can use this table to facilitate further questioning and testing of the patient in order to decide on the final diagnosis.

```
+-------------------------------------------------+
| n.name           | d.name                | score |
+-------------------------------------------------+
| "Sally Anderson" | "Chronic Otheritis"   | 4     |
| "Sally Anderson" | "Chronic Yellowitis"  | 3     |
| "Sally Anderson" | "Chronic Placeboitis" | 3     |
| "Sally Anderson" | "Acute Whatitis"      | 2     |
| "Sally Anderson" | "Acute Yellowitis"    | 2     |
| "Sally Anderson" | "Chronic Someitis"    | 2     |
| "Sally Anderson" | "Chronic Argitis"     | 2     |
| "Sally Anderson" | "Chronic Whatitis"    | 2     |
| "Sally Anderson" | "Acute Someitis"      | 1     |
| "Sally Anderson" | "Acute Argitis"       | 1     |
| "Sally Anderson" | "Acute Otheritis"     | 1     |
+-------------------------------------------------+
```

Once the doctor has investigated further, they would be able to decide on the diagnosis and save that result to the database:

```
WITH datetime({epochmillis:timestamp()}) AS now
WITH now, date(now) as today
MATCH (p:Patient)
  WHERE p.ssn = 1234657
MATCH (d:Disease)
  WHERE d.name = "Chronic Placeboitis"
MERGE (p)-[i:DIAGNOSIS {by: 'Alice'}]->(d)
  ON CREATE SET i.created_at = now, i.updated_at = now, i.date = today
  ON MATCH SET i.updated_at = now
RETURN p.name, d.name, i.by, i.date, duration.between(i.created_at, i.updated_at) AS updated;
```

This allows this doctor to record their diagnosis as well as take note of previous diagnoses:

```
+-------------------------------------------------------------------------------------+
| p.name           | d.name                | i.by    | i.date     | updated             |
+-------------------------------------------------------------------------------------+
| "Sally Anderson" | "Chronic Placeboitis" | "Alice" | 2020-05-29 | P0M0DT213.076000000S |
+-------------------------------------------------------------------------------------+
```

In order to create the `DIAGNOSIS` relationship for the first time, it is required to have the privilege to create new types. This is also true of the property names `doctor`, `created_at` and `updated_at`. This can be fixed by either granting the doctor `NAME MANAGEMENT` privileges or by pre-creating the missing types. The latter would be more precise and can be achieved by running, as an administrator, the procedures `db.createRelationshipType` and `db.createProperty` with appropriate arguments.

Privileges of `receptionist`

Receptionists should only be able to manage patient information. They are not allowed to find or read any other parts of the graph. In addition, they should be able to create and delete patients, but not any other nodes:

```
CREATE ROLE receptionist;
GRANT MATCH {*} ON GRAPH healthcare NODES Patient TO receptionist;
GRANT CREATE ON GRAPH healthcare NODES Patient TO receptionist;
GRANT DELETE ON GRAPH healthcare NODES Patient TO receptionist;
GRANT SET PROPERTY {*} ON GRAPH healthcare NODES Patient TO receptionist;
```

It would have been simpler to grant global `WRITE` privileges. However, this would have the unfortunate side effect of allowing the receptionist the ability to create other nodes, like new `Symptom` nodes, even though they would subsequently be unable to find or read those same nodes. While there are use cases for being able to create data you cannot read, that is not desired for this model.

```
neo4j@system> GRANT ROLE receptionist TO bob;
```

With these privileges, if Bob tries to read the entire database, he will still only see the patients:

```
MATCH (n) WITH labels(n) AS labels
RETURN labels, count(*);
```

```
+------------------------+
| labels      | count(*) |
+------------------------+
| ["Patient"] | 101      |
+------------------------+
```

However, Bob is able to see all fields of the Patient records:

```
MATCH (n:Patient)
 WHERE n.dateOfBirth < date('1972-06-12')
RETURN n.name, n.ssn, n.address, n.dateOfBirth;
```

```
+-----------------------------------------------------------------------+
| n.name          | n.ssn   | n.address                | n.dateOfBirth |
+-----------------------------------------------------------------------+
| "Mark Stone"    | 1234666 | "1 secret way, downtown" | 1970-08-04    |
| "Sally Jackson" | 1234633 | "1 secret way, downtown" | 1970-10-21    |
| "Bob Stone"     | 1234581 | "1 secret way, downtown" | 1972-02-16    |
| "Ally Anderson" | 1234582 | "1 secret way, downtown" | 1970-05-13    |
| "Mark Svensson" | 1234594 | "1 secret way, downtown" | 1970-01-16    |
| "Bob Anderson"  | 1234597 | "1 secret way, downtown" | 1970-09-23    |
| "Jack Svensson" | 1234599 | "1 secret way, downtown" | 1971-02-13    |
| "Mark Jackson"  | 1234618 | "1 secret way, downtown" | 1970-03-28    |
| "Jack Jackson"  | 1234623 | "1 secret way, downtown" | 1971-04-02    |
+-----------------------------------------------------------------------+
```

We have granted Bob the receptionist the ability to delete patient nodes. This will allow him to delete any new patients he has just created, but will not allow him the ability to delete patients that have already received diagnoses, because those are connected to parts of the graph the receptionist cannot see. Let's demonstrate both these scenarios:

```
CREATE (n:Patient {
  ssn:87654321,
  name: 'Another Patient',
  email: 'another@example.com',
  address: '1 secret way, downtown',
  dateOfBirth: date('2001-01-20')
})
RETURN n.name, n.dateOfBirth;
```

```
+-----------------------------------+
| n.name            | n.dateOfBirth |
+-----------------------------------+
| "Another Patient" | 2001-01-20    |
+-----------------------------------+
```

The receptionist is able to modify any patient record:

```
MATCH (n:Patient)
WHERE n.ssn = 87654321
SET n.address = '2 streets down, uptown'
RETURN n.name, n.dateOfBirth, n.address;
```

```
+-------------------------------------------------------------+
| n.name            | n.dateOfBirth | n.address               |
+-------------------------------------------------------------+
| "Another Patient" | 2001-01-20    | "2 streets down, uptown" |
+-------------------------------------------------------------+
```

The receptionist is also able to delete this recently created patient because it is not connected to any other records:

```
MATCH (n:Patient)
 WHERE n.ssn = 87654321
DETACH DELETE n;
```

However, if the receptionist attempts to delete a patient that has existing diagnoses, this will fail:

```
MATCH (n:Patient)
 WHERE n.ssn = 1234610
DETACH DELETE n;
```

```
org.neo4j.graphdb.ConstraintViolationException: Cannot delete node<42>, because it still has
relationships. To delete this node, you must first delete its relationships.
```

The reason this fails is that Bob can find the `(:Patient)` node, but does not have sufficient traverse rights to find nor delete the outgoing relationships from it. Either he needs to ask Tina the `itadmin` for help for this task, or we can add more privileges to the `receptionist` role:

```
GRANT TRAVERSE ON GRAPH healthcare NODES Symptom, Disease TO receptionist;
GRANT TRAVERSE ON GRAPH healthcare RELATIONSHIPS HAS, DIAGNOSIS TO receptionist;
GRANT DELETE ON GRAPH healthcare RELATIONSHIPS HAS, DIAGNOSIS TO receptionist;
```

Privileges that were granted or denied earlier can be revoked using the `REVOKE` command. See the Cypher Manual → The `REVOKE` command.

Privileges of nurses

As previously described, nurses have the capabilities of both doctors and receptionists. As such it would be tempting to assign them both the `doctor` and the `receptionist` roles. However, this might not have the effect you would expect. If those two roles were created with `GRANT` privileges only, combining them would be simply cumulative. But it turns out the doctor contains some `DENY` privileges, and these always overrule `GRANT`. This means that the nurse will still have the same restrictions as a doctor, which is not what we wanted.

To demonstrate this, let's give it a try:

```
neo4j@system> GRANT ROLE doctor, receptionist TO daniel;
```

Now we can see that the user 'Daniel' has a combined set of privileges:

```
SHOW USER daniel PRIVILEGES;
```

```
+-----------------------------------------------------------------------------------------------------
--------------+
| access    | action           | resource            | graph        | segment           | role
| user     |
+-----------------------------------------------------------------------------------------------------
--------------+
| "GRANTED" | "access"         | "database"          | "DEFAULT"    | "database"        | "PUBLIC"
| "daniel" |
| "GRANTED" | "access"         | "database"          | "healthcare" | "database"        | "PUBLIC"
| "daniel" |
| "GRANTED" | "execute"        | "database"          | "*"          | "FUNCTION(*)"     | "PUBLIC"
| "daniel" |
| "GRANTED" | "execute"        | "database"          | "*"          | "PROCEDURE(*)"    | "PUBLIC"
| "daniel" |
| "GRANTED" | "read"           | "all_properties"    | "healthcare" | "NODE(*)"         | "doctor"
| "daniel" |
| "GRANTED" | "traverse"       | "graph"             | "healthcare" | "NODE(*)"         | "doctor"
| "daniel" |
| "GRANTED" | "write"          | "graph"             | "healthcare" | "NODE(*)"         | "doctor"
| "daniel" |
| "DENIED"  | "read"           | "property(address)" | "healthcare" | "NODE(Patient)"   | "doctor"
| "daniel" |
| "DENIED"  | "set_property"   | "property(address)" | "healthcare" | "NODE(Patient)"   | "doctor"
| "daniel" |
| "GRANTED" | "read"           | "all_properties"    | "healthcare" | "RELATIONSHIP(*)" | "doctor"
| "daniel" |
| "GRANTED" | "traverse"       | "graph"             | "healthcare" | "RELATIONSHIP(*)" | "doctor"
| "daniel" |
| "GRANTED" | "write"          | "graph"             | "healthcare" | "RELATIONSHIP(*)" | "doctor"
| "daniel" |
| "GRANTED" | "match"          | "all_properties"    | "healthcare" | "NODE(Patient)"   |
"receptionist" | "daniel" |
| "GRANTED" | "set_property"   | "all_properties"    | "healthcare" | "NODE(Patient)"   |
"receptionist" | "daniel" |
| "GRANTED" | "create_element" | "graph"             | "healthcare" | "NODE(Patient)"   |
"receptionist" | "daniel" |
| "GRANTED" | "delete_element" | "graph"             | "healthcare" | "NODE(Patient)"   |
"receptionist" | "daniel" |
+-----------------------------------------------------------------------------------------------------
--------------+
```

Privileges that were granted or denied earlier can be revoked using the `REVOKE` command. See the Cypher Manual → The `REVOKE` command.

Now the intention is that a nurse can perform the actions of a receptionist. This would mean they should be able to read and write the `address` field of the `Patient` nodes.

```
MATCH (n:Patient)
 WHERE n.dateOfBirth < date('1972-06-12')
RETURN n.name, n.ssn, n.address, n.dateOfBirth;
```

```
+-----------------------------------------------------+
| n.name          | n.ssn   | n.address | n.dateOfBirth |
+-----------------------------------------------------+
| "Jane Anderson" | 1234572 | NULL      | 1971-05-26    |
| "Mark Stone"    | 1234586 | NULL      | 1972-06-07    |
| "Joe Smith"     | 1234595 | NULL      | 1970-12-28    |
| "Joe Jackson"   | 1234603 | NULL      | 1970-08-31    |
| "Jane Jackson"  | 1234628 | NULL      | 1972-01-31    |
| "Mary Anderson" | 1234632 | NULL      | 1971-01-07    |
| "Jack Svensson" | 1234639 | NULL      | 1970-01-06    |
+-----------------------------------------------------+
```

Clearly the address field is invisible. This is due to the DENIED privileges we could see in the table earlier. If we tried to write to the address field we would receive an error. This is not the intended behavior. We have two choices to correct otherwise:

- We could redefine the doctor role with only grants, requiring that we define each Patient property we wish the doctor to be able to read.
- We can redefine the nurse role with the actual intended behavior.

It turns out that the latter choice is by far the simplest. The nurse is essentially the doctor without the address restrictions:

```
CREATE ROLE nurse
GRANT TRAVERSE ON GRAPH healthcare TO nurse;
GRANT READ {*} ON GRAPH healthcare TO nurse;
GRANT WRITE ON GRAPH healthcare TO nurse;
```

Now let's assign this role to Daniel and test the new behavior:

```
REVOKE ROLE doctor FROM daniel;
REVOKE ROLE receptionist FROM daniel;
GRANT ROLE nurse TO daniel;
```

When the *improved* nurse Daniel takes another look at the patient records, he will see the address fields:

```
MATCH (n:Patient)
 WHERE n.dateOfBirth < date('1972-06-12')
RETURN n.name, n.ssn, n.address, n.dateOfBirth;
```

```
+-------------------------------------------------------------------+
| n.name          | n.ssn   | n.address                | n.dateOfBirth |
+-------------------------------------------------------------------+
| "Jane Anderson" | 1234572 | "1 secret way, downtown" | 1971-05-26    |
| "Mark Stone"    | 1234586 | "1 secret way, downtown" | 1972-06-07    |
| "Joe Smith"     | 1234595 | "1 secret way, downtown" | 1970-12-28    |
| "Joe Jackson"   | 1234603 | "1 secret way, downtown" | 1970-08-31    |
| "Jane Jackson"  | 1234628 | "1 secret way, downtown" | 1972-01-31    |
| "Mary Anderson" | 1234632 | "1 secret way, downtown" | 1971-01-07    |
| "Jack Svensson" | 1234639 | "1 secret way, downtown" | 1970-01-06    |
+-------------------------------------------------------------------+
```

Now Daniel can see the previously hidden address field. The other main action we want the nurse to be

able to perform, is the primary doctor action of saving a diagnosis to the database:

```
WITH date(datetime({epochmillis:timestamp()})) AS today
MATCH (p:Patient)
  WHERE p.ssn = 1234657
MATCH (d:Disease)
  WHERE d.name = "Chronic Placeboitis"
MERGE (p)-[i:DIAGNOSIS {by: 'Daniel'}]->(d)
  ON CREATE SET i.date = today
RETURN p.name, d.name, i.by, i.date;
```

```
+-------------------------------------------------------------------+
| p.name           | d.name                | i.by     | i.date     |
+-------------------------------------------------------------------+
| "Sally Anderson" | "Chronic Placeboitis" | "Daniel" | 2020-05-29 |
+-------------------------------------------------------------------+
```

Performing an action otherwise reserved for the doctor role involves more responsibility for the nurse. Perhaps it is not desirable to entrust all nurses with this option, which is why we can divide the nurses into *senior* and *junior* nurses. Daniel is currently a senior nurse.

Privileges of junior nurses

When we tried to create the senior nurse by combining the doctor and receptionist roles, that did not work out. As previously mentioned, it would work to combine two roles if the intention is to increase capabilities and the roles were created with GRANT privileges only. It is also possible to combine two roles if the intention is to reduce capabilities and the combination brings in DENY privileges.

Consider this case, we would like a junior nurse to be able to perform the same actions as a senior nurse, but not be able to save diagnoses. We could create a special role that contains specifically only the additional restrictions:

```
CREATE ROLE disableDiagnoses;
DENY CREATE ON GRAPH healthcare RELATIONSHIPS DIAGNOSIS TO disableDiagnoses;
```

Now let's assign this role to Daniel and test the new behaviour:

```
GRANT ROLE disableDiagnoses TO daniel;
```

If we look at what privileges Daniel now has, it will be the combination of the two roles nurse and disableDiagnoses:

```
neo4j@system> SHOW USER daniel PRIVILEGES;
```

```
+-------------------------------------------------------------------------------------------------------
------------------------+
| access    | action           | resource         | graph        | segment                    | role
| user     |
+-------------------------------------------------------------------------------------------------------
------------------------+
| "GRANTED" | "access"         | "database"       | "DEFAULT"    | "database"                 |
"PUBLIC"           | "daniel" |
| "GRANTED" | "access"         | "database"       | "healthcare" | "database"                 |
"PUBLIC"           | "daniel" |
| "GRANTED" | "execute"        | "database"       | "*"          | "FUNCTION(*)"              |
"PUBLIC"           | "daniel" |
| "GRANTED" | "execute"        | "database"       | "*"          | "PROCEDURE(*)"             |
"PUBLIC"           | "daniel" |
| "DENIED"  | "create_element" | "graph"          | "healthcare" | "RELATIONSHIP(DIAGNOSIS)" |
"disableDiagnoses" | "daniel" |
| "GRANTED" | "read"           | "all_properties" | "healthcare" | "NODE(*)"                  |
"nurse"            | "daniel" |
| "GRANTED" | "traverse"       | "graph"          | "healthcare" | "NODE(*)"                  |
"nurse"            | "daniel" |
| "GRANTED" | "write"          | "graph"          | "healthcare" | "NODE(*)"                  |
"nurse"            | "daniel" |
| "GRANTED" | "read"           | "all_properties" | "healthcare" | "RELATIONSHIP(*)"          |
"nurse"            | "daniel" |
| "GRANTED" | "traverse"       | "graph"          | "healthcare" | "RELATIONSHIP(*)"          |
"nurse"            | "daniel" |
| "GRANTED" | "write"          | "graph"          | "healthcare" | "RELATIONSHIP(*)"          |
"nurse"            | "daniel" |
+-------------------------------------------------------------------------------------------------------
------------------------+
```

Daniel can still see address fields, and can even perform the diagnosis investigation that the `doctor` can perform:

```
MATCH (n:Patient)-[:HAS]->(s:Symptom)-[:OF]->(d:Disease)
WHERE n.ssn = 1234650
RETURN n.ssn, n.name, d.name, count(s) AS score ORDER BY score DESC;
```

```
+-----------------------------------------------------+
| n.ssn   | n.name       | d.name                | score |
+-----------------------------------------------------+
| 1234650 | "Mark Smith" | "Chronic Whatitis"    | 3     |
| 1234650 | "Mark Smith" | "Chronic Someitis"    | 3     |
| 1234650 | "Mark Smith" | "Acute Someitis"      | 2     |
| 1234650 | "Mark Smith" | "Chronic Otheritis"   | 2     |
| 1234650 | "Mark Smith" | "Chronic Yellowitis"  | 2     |
| 1234650 | "Mark Smith" | "Chronic Placeboitis" | 2     |
| 1234650 | "Mark Smith" | "Acute Otheritis"     | 2     |
| 1234650 | "Mark Smith" | "Chronic Argitis"     | 2     |
| 1234650 | "Mark Smith" | "Acute Placeboitis"   | 2     |
| 1234650 | "Mark Smith" | "Acute Yellowitis"    | 1     |
| 1234650 | "Mark Smith" | "Acute Argitis"       | 1     |
| 1234650 | "Mark Smith" | "Acute Whatitis"      | 1     |
+-----------------------------------------------------+
```

But when he tries to save a diagnosis to the database, he will be denied:

```
WITH date(datetime({epochmillis:timestamp()})) AS today
MATCH (p:Patient)
  WHERE p.ssn = 1234650
MATCH (d:Disease)
  WHERE d.name = "Chronic Placeboitis"
MERGE (p)-[i:DIAGNOSIS {by: 'Daniel'}]->(d)
  ON CREATE SET i.date = today
RETURN p.name, d.name, i.by, i.date;
```

```
Create relationship with type 'DIAGNOSIS' is not allowed for user 'daniel' with roles [PUBLIC,
disableDiagnoses, nurse].
```

Promoting Daniel back to senior nurse will be as simple as revoking the role that introduced the restriction:

```
REVOKE ROLE disableDiagnoses FROM daniel;
```

Building a custom administrator role

Originally we created the `itadmin` role by copying the built-in `admin` role and adding restrictions. However, we have also shown cases where having denys can be less convenient than only having grants. So can we instead build the administrator role from the ground up?

Let's review the purpose of this role. The intention is that Tina, the administrator, can create new users and assign them to the product roles. We can create a new role called `userManager` and grant it the appropriate privileges:

```
CREATE ROLE userManager;
GRANT USER MANAGEMENT ON DBMS TO userManager;
GRANT ROLE MANAGEMENT ON DBMS TO userManager;
GRANT SHOW PRIVILEGE ON DBMS TO userManager;
```

We need to revoke the `itadmin` role from Tina and grant her the `userManager` role instead:

```
REVOKE ROLE itadmin FROM tina
GRANT ROLE userManager TO tina
```

The three privileges we've granted will allow:

- `USER MANAGEMENT` allows creating, updating and dropping users
- `ROLE MANAGEMENT` allows assigning roles to users
- `SHOW PRIVILEGE` allows listing the users privileges

Listing Tina's new privileges should show a much shorter list than when she was a more powerful administrator:

```
neo4j@system> SHOW USER tina PRIVILEGES;
```

```
+-----------------------------------------------------------------------------------------------------
-+
| access    | action            | resource   | graph        | segment        | role          | user
|
+-----------------------------------------------------------------------------------------------------
-+
| "GRANTED" | "access"          | "database" | "DEFAULT"    | "database"     | "PUBLIC"      | "tina"
|
| "GRANTED" | "access"          | "database" | "healthcare" | "database"     | "PUBLIC"      | "tina"
|
| "GRANTED" | "execute"         | "database" | "*"          | "FUNCTION(*)"  | "PUBLIC"      | "tina"
|
| "GRANTED" | "execute"         | "database" | "*"          | "PROCEDURE(*)" | "PUBLIC"      | "tina"
|
| "GRANTED" | "role_management" | "database" | "*"          | "database"     | "userManager" | "tina"
|
| "GRANTED" | "show_privilege"  | "database" | "*"          | "database"     | "userManager" | "tina"
|
| "GRANTED" | "user_management" | "database" | "*"          | "database"     | "userManager" | "tina"
|
+-----------------------------------------------------------------------------------------------------
-+
```

We have not granted any other privilege management privileges. How much power this role should have would depend on the requirements of the system. Refer to the section Cypher Manual → Security of Administration for a complete list of privileges to consider.

Now Tina should be able to create new users and assign them to roles:

```
CREATE USER sally SET PASSWORD 'secret' CHANGE REQUIRED;
GRANT ROLE receptionist TO sally;
SHOW USER sally PRIVILEGES;
```

```
+-----------------------------------------------------------------------------------------------------
---------+
| access    | action           | resource         | graph        | segment         | role           |
user    |
+-----------------------------------------------------------------------------------------------------
---------+
| "GRANTED" | "access"         | "database"       | "DEFAULT"    | "database"      | "PUBLIC"       |
"sally" |
| "GRANTED" | "access"         | "database"       | "healthcare" | "database"      | "PUBLIC"       |
"sally" |
| "GRANTED" | "execute"        | "database"       | "*"          | "FUNCTION(*)"   | "PUBLIC"       |
"sally" |
| "GRANTED" | "execute"        | "database"       | "*"          | "PROCEDURE(*)"  | "PUBLIC"       |
"sally" |
| "GRANTED" | "match"          | "all_properties" | "healthcare" | "NODE(Patient)" | "receptionist" |
"sally" |
| "GRANTED" | "set_property"   | "all_properties" | "healthcare" | "NODE(Patient)" | "receptionist" |
"sally" |
| "GRANTED" | "create_element" | "graph"          | "healthcare" | "NODE(Patient)" | "receptionist" |
"sally" |
| "GRANTED" | "delete_element" | "graph"          | "healthcare" | "NODE(Patient)" | "receptionist" |
"sally" |
+-----------------------------------------------------------------------------------------------------
---------+
```

10.5. Integration with LDAP directory services

10.5.1. Introduction

Neo4j supports LDAP, which allows for integration with Active Directory (AD), OpenLDAP, or other LDAP-compatible authentication services. This means that you use the LDAP service for managing federated users, while the native Neo4j user and role administration are completely turned off.

The following configuration settings are important to consider when configuring LDAP. For a more detailed overview of the LDAP configuration options, see Configuration settings.

10.5.2. LDAP configuration parameters

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. Common values: `memberOf` and `gidNumber`.

Parameter name	Default value	Description
dbms.security.ldap.authorization.group_to_role_mapping		Lists an authorization mapping from groups to the pre-defined built-in roles `admin`, `architect`, `publisher`, `editor`, and `reader`, or to any other custom-defined roles.

All settings are defined at server startup time in the default configuration file *neo4j.conf*.

10.5.3. Set Neo4j to use LDAP

First, you configure Neo4j to use LDAP as an authentication and authorization provider.

1. Uncomment the setting `dbms.security.auth_enabled=false` and change its value to `true` to turn on the security feature.
2. Uncomment the settings `dbms.security.authentication_providers` and `dbms.security.authorization_providers` and change their value to `ldap`. This way, the LDAP connector will be used as a security provider for both authentication and authorization.

10.5.4. Map the LDAP groups to the Neo4j roles

To access the user and role management procedures, you have to map the LDAP groups to the Neo4j built-in and custom-defined roles. To do that, you need to know what privileges the Neo4j roles have, and based on these privileges, to create the mapping to the groups defined in the LDAP server. The map must be formatted as a semicolon separated list of key-value pairs, where the key is a comma-separated list of the LDAP group names and the value is a comma-separated list of the corresponding role names. For example, `group1=role1;group2=role2;group3=role3,role4,role5;group4,group5=role6`.

Example 62. Example of LDAP groups to Neo4j roles mapping

```
dbms.security.ldap.authorization.group_to_role_mapping=\
    "cn=Neo4j Read Only,cn=users,dc=example,dc=com"      = reader;    \ ①
    "cn=Neo4j Read-Write,cn=users,dc=example,dc=com"     = editor,publisher; \ ②
    "cn=Neo4j Read-Write,cn=users,dc=example,dc=com","cn=Neo4j Create
Data,cn=users,dc=example,dc=com"     = publisher; \ ③
    "cn=Neo4j Create Data,cn=users,dc=example,dc=com","cn=Neo4j Schema
Manager,cn=users,dc=example,dc=com" = architect; \
    "cn=Neo4j Administrator,cn=users,dc=example,dc=com"  = admin; \
    "cn=Neo4j Procedures,cn=users,dc=neo4j,dc=com"       = rolename ④
```

① Mapping of an LDAP group to a Neo4j built-in role.

② Mapping of an LDAP group to two Neo4j built-in roles.

③ Mapping of two LDAP groups to a Neo4j built-in role.

④ Mapping of an LDAP group to a custom-defined role. Custom-defined roles, such as `rolename`, must be explicitly created using the `CREATE ROLE rolename` command before they can be used to grant privileges. See the Cypher Manual → Creating roles.

10.5.5. Configure Neo4j to use Active Directory

You configure Neo4j to use the LDAP security provider to access and manage your Active Directory. There are three alternative ways to do that depending on your specific use case.

Configure Neo4j to support LDAP user ID authentication

This option allows users to log in with their LDAP user ID.

In the *neo4j.conf* file, uncomment and configure the following settings:

1. Configure LDAP to point to the AD server:

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

2. Provide details on the user structure of the LDAP directory:

```
dbms.security.ldap.authentication.user_dn_template=cn={0},cn=Users,dc=example,dc=com
dbms.security.ldap.authorization.user_search_base=cn=Users,dc=example,dc=com
dbms.security.ldap.authorization.user_search_filter=(&(objectClass=*)(cn={0}))
dbms.security.ldap.authorization.group_membership_attributes=memberOf
```

3. Map the groups in the LDAP system to the Neo4j built-in and custom roles. See Map the LDAP groups to the Neo4j roles.

Configure Neo4j to support `sAMAccountName` authentication by creating a system account

This is an alternative configuration for Active Directory that allows users to log in using `sAMAccountName`. You create a system account that has read-only access to the parts of the LDAP directory that you want. However, it does not need to have access rights to Neo4j or any other systems.

In the *neo4j.conf* file, uncomment and configure the following settings:

1. Configure LDAP to point to the AD server:

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

2. Provide details on the user structure of the LDAP directory:

```
dbms.security.ldap.authorization.user_search_base=cn=Users,dc=example,dc=com
dbms.security.ldap.authorization.user_search_filter=(&(objectClass=*)(samaccountname={0}))
dbms.security.ldap.authorization.group_membership_attributes=memberOf
```

3. Map the groups in the LDAP system to the Neo4j built-in and custom roles. See Map the LDAP groups to the Neo4j roles.
4. Configure Neo4j to use a system account with read access to all users and groups in the LDAP server.
    a. Set `dbms.security.ldap.authorization.use_system_account` value to `true`.

b. Set `dbms.security.ldap.authorization.system_username` value to the full Distinguished Name (DN) as the `dbms.security.ldap.authentication.user_dn_template` will not be applied to this username. For example,

```
dbms.security.ldap.authorization.system_username=cn=search-account,cn=Users,dc=example,dc=com
```

c. Configure the LDAP system account password.

```
dbms.security.ldap.authorization.system_password=mypassword
```

`sAMAccountName` requires `dbms.security.ldap.authorization.system_username` and `dbms.security.ldap.authorization.system_password` to be used, since there is no way to log in through LDAP directly with the `sAMAccountName`. Instead, the login name will be resolved to a DN that will be used to log in with.

d. Add the following line to the *neoj.conf* file:

```
dbms.security.ldap.authentication.use_samaccountname=true
```

Configure Neo4j to support `sAMAccountName` authentication by setting `user_dn_template`

This is an alternative configuration for Active Directory that allows all users from the specified domain to log in using `sAMAccountName`. With this option, you do not have to create a system account and store a system username/password in the config file. Instead, you set `+\{0}+@example.com` as a value of the `user_dn_template` to enable the authentication to start at the root domain. This way, the whole tree is checked to find the user, regardless of where it is located within the LDAP directory tree.

In the *neo4j.conf* file, uncomment and configure the following settings:

1. Configure LDAP to point to the AD server:

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

2. Provide details on the user structure of the LDAP directory:

```
dbms.security.ldap.authentication.user_dn_template={0}@example.com
dbms.security.ldap.authorization.user_search_base=dc=example,dc=com
dbms.security.ldap.authorization.user_search_filter=(&(objectClass=user)(sAMAccountName={0}))
dbms.security.ldap.authorization.group_membership_attributes=memberOf
```

3. Map the groups in the LDAP system to the Neo4j built-in and custom roles. For more information, see Map the LDAP groups to the Neo4j roles.

The setting `dbms.security.ldap.authentication.use_samaccountname` is not configured.

10.5.6. Configure Neo4j to use OpenLDAP

You configure the LDAP security provider to access and manage your OpenLDAP directory service.

In the neo4j.conf file, uncomment and configure the following settings:

1. Configure LDAP to point to the OpenLDAP server:

```
dbms.security.ldap.host=myopenldap.example.com
```

2. Provide details on the user structure of the LDAP directory:

```
dbms.security.ldap.authentication.user_dn_template=cn={0},ou=users,dc=example,dc=com
dbms.security.ldap.authorization.user_search_base=ou=users,dc=example,dc=com
dbms.security.ldap.authorization.user_search_filter=(&(objectClass=*)(uid={0}))
dbms.security.ldap.authorization.group_membership_attributes=gidNumber
```

3. Map the groups in the LDAP system to the Neo4j built-in and custom roles. For more information, see Map the LDAP groups to the Neo4j roles.

10.5.7. Verify the LDAP configuration

You can verify that your LDAP configuration is correct, and that the LDAP server responds, by using the LDAP command-line tool `ldapsearch`.

The `ldapsearch` command accepts the LDAP configuration setting values as input and verifies both the authentication (using the `simple` mechanism) and authorization of a user. See the ldapsearch official documentation for more advanced usage and how to use SASL authentication mechanisms.

1. Verify the authentication and authorization of a user. For example, `john`.
   - With `dbms.security.ldap.authorization.use_system_account=false` (default):

```
# ldapsearch -v -H ldap://<dbms.security.ldap.host> -x -D
<dbms.security.ldap.authentication.user_dn_template : replace {0}> -W -b
<dbms.security.ldap.authorization.user_search_base>
"<dbms.security.ldap.authorization.user_search_filter : replace {0}>"
<dbms.security.ldap.authorization.group_membership_attributes>

ldapsearch -v -H ldap://myactivedirectory.example.com:389 -x -D cn=john,cn=Users,dc=example,dc=com
-W -b cn=Users,dc=example,dc=com "(&(objectClass=*)(cn=john))" memberOf
```

   - With `dbms.security.ldap.authorization.use_system_account=true`:

```
# ldapsearch -v -H ldap://<dbms.security.ldap.host> -x -D
<dbms.security.ldap.authorization.system_username> -w
<dbms.security.ldap.authorization.system_password> -b
<dbms.security.ldap.authorization.user_search_base>
"<dbms.security.ldap.authorization.user_search_filter>"
<dbms.security.ldap.authorization.group_membership_attributes>

ldapsearch -v -H ldap://myactivedirectory.example.com:389 -x -D cn=search-account,cn=Users,dc
=example,dc=com -w mypassword -b cn=Users,dc=example,dc=com "(&(objectClass=*)(cn=john))" memberOf
```

2. Verify that the value of the returned membership attribute is a group that is mapped to a role in

dbms.security.ldap.authorization.group_to_role_mapping.

```
# extended LDIF
#
# LDAPv3
# base <cn=Users,dc=example,dc=com> with scope subtree
# filter: (cn=john)
# requesting: memberOf
#

# john, Users, example.com
dn: CN=john,CN=Users,DC=example,DC=com
memberOf: CN=Neo4j Read Only,CN=Users,DC=example,DC=com

# search result
search: 2
result: 0 Success

# numResponses: 2
# numEntries: 1
```

10.5.8. The auth cache

The *auth cache* is the mechanism by which Neo4j caches the result of authentication via the LDAP server in order to aid performance. It is configured with the parameters dbms.security.ldap.authentication.cache_enabled, and dbms.security.auth_cache_ttl.

```
# Turn on authentication caching to ensure performance.

dbms.security.ldap.authentication.cache_enabled=true
dbms.security.auth_cache_ttl=10m
```

Table 42. Auth cache parameters

Parameter name	Default value	Description
dbms.security.ldap.authentication.cache_enabled	true	Determines whether or not to cache the result of authentication via the LDAP server. Whether authentication caching should be enabled or not must be considered in view of your company's security guidelines.

Parameter name	Default value	Description
dbms.security.auth_cache_ttl	600 seconds	Is the time to live (TTL) for cached authentication and authorization info. Setting the TTL to 0 disables all auth caching. A short TTL requires more frequent re-authentication and re-authorization, which can impact performance. A very long TTL means that changes to the users settings on an LDAP server may not be reflected in the Neo4j authorization behaviour in a timely manner. Valid units are ms, s, m; default unit is s.

An administrator can clear the auth cache to force the re-querying of authentication and authorization information from the federated auth provider system. Use Neo4j Browser or Neo4j Cypher Shell to execute this statement:

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

10.5.9. Available methods of encryption

Specifying the dbms.security.ldap.host parameter configures using LDAP without encryption. Not specifying the protocol or port results in ldap being used over the default port 389.

```
dbms.security.ldap.host=myactivedirectory.example.com
dbms.security.ldap.host=myactivedirectory.example.com:389
dbms.security.ldap.host=ldap://myactivedirectory.example.com
dbms.security.ldap.host=ldap://myactivedirectory.example.com:389
```

Use LDAP with encryption via StartTLS

To configure Active Directory with encryption via StartTLS, set the following parameters:

```
dbms.security.ldap.use_starttls=true
dbms.security.ldap.host=ldap://myactivedirectory.example.com
```

Use LDAP with encrypted LDAPS

To configure Active Directory with encrypted LDAPS, set dbms.security.ldap.host to one of the following. If you do not specify the port, the default one 636 will be used.

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

10.5.10. Use a self-signed certificate (SSL) in a test environment

Production environments should always use an SSL certificate issued by a Certificate Authority for secure access to the LDAP server. However, there are scenarios, for example in test environments, where you may want to use an SSL certificate on the LDAP server.

To configure an SSL certificate on LDAP server, enter the details of the certificate using `dbms.jvm.additional` in *neo4j.conf*. The path to the certificate file `MyCert.jks` is an absolute path to the Neo4j server.

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

10.6. Manage procedure and user-defined function permissions

10.6.1. Introduction

To be able to run a procedure or user-defined function, the user needs to have the corresponding execute privilege. Procedures and user-defined functions are executed according to the same security rules as regular Cypher statements, e.g. a procedure performing writes will fail if called by a user that only has *read* privileges.

Procedures and user-defined functions can also be run with privileges exceeding the users own privileges. This is called *execution boosting*. The elevated privileges only apply within the procedure or user-defined function; any operation performed outside will still use the users original privileges.

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

Please refer to Java Reference → Extending Neo4j for a description on creating and using user-defined procedures and functions.

10.6.2. Manage procedure permissions

Procedure permissions can be managed using the native execute privileges. These control whether the user is allowed to both execute a procedure, and which set of privileges apply during the execution.

A procedure may be run using the `EXECUTE PROCEDURE` privilege.

This allows the user to execute procedures that match the globbed procedures.

Example 63. Grant privilege to execute procedure

```
GRANT EXECUTE PROCEDURE db.schema.visualization ON DBMS TO visualizer
```

This will allow any user with the `visualizer` role to execute the `db.schema.visualization`. E.g. a user that also have the following privileges:

```
GRANT TRAVERSE ON GRAPH * NODES A, B TO role
GRANT TRAVERSE ON GRAPH * RELATIONSHIP R1 TO role
```

When calling the `db.schema.visualization` procedure that user will only see the `A` and `B` nodes and `R1` relationships, even though there might exist other nodes and relationships.

A procedure may also be executed with elevated privileges using the `EXECUTE BOOSTED PROCEDURE` privilege.

This allows the user to successfully execute procedures that would otherwise fail during execution with their assigned roles. The user is given full privileges for the procedure, during the execution of the procedure only.

Example 64. Grant privilege to execute procedure with elevated privileges

```
GRANT EXECUTE BOOSTED PROCEDURE db.schema.visualization ON DBMS TO visualizer
```

This will allow any user with the `visualizer` role to execute the `db.schema.visualization` with elevated privileges. When calling the `db.schema.visualization` procedure that user will see all nodes and relationships that exist in the graph, even though they have no traversal privileges.

10.6.3. Manage user-defined function permissions

User-defined function permissions can be managed using the native execute privileges. These control if the user is both allowed to execute a user-defined function, and which set of privileges apply during the execution.

A user-defined function may be executed using the `EXECUTE USER DEFINED FUNCTION` privilege.

This allows the user to execute user-defined functions that match the globbed user-defined function.

Example 65. Grant privilege to execute user-defined function

```
GRANT EXECUTE USER DEFINED FUNCTION apoc.any.properties ON DBMS TO custom
```

This will allow any user with the `custom` role to execute the `apoc.any.properties`. E.g. a user that also have the following privilege:

```
GRANT MATCH {visibleProp} ON GRAPH * NODES A TO role
```

When calling the user-defined function `MATCH (a:A) RETURN apoc.any.properties(a) AS properties`, they will only see the `visibleProp` even though there might exist other properties.

A user-defined function may also be executed with elevated privileges using the `EXECUTE BOOSTED USER DEFINED FUNCTION` privilege.

This allows the user to successfully execute user-defined functions that would otherwise fail during execution with their assigned roles. The user is given full privileges for the user-defined function, during the execution of the function only.

Example 66. Grant privilege to execute user-defined function with elevated privileges

```
GRANT EXECUTE BOOSTED USER DEFINED FUNCTION apoc.any.properties ON DBMS TO custom
```

This will allow any user with the `custom` role to execute the `apoc.any.properties` with elevated privileges. E.g. a user that also have the following privileges:

```
GRANT TRAVERSE ON GRAPH * NODES A TO role
```

When calling the user-defined function `MATCH (a:A) RETURN apoc.any.properties(a) AS properties`, they will see all properties that exist on the matched nodes even though they have no read privileges.

10.6.4. Manage procedure and user-defined function permissions from config setting Deprecated

It is possible to grant boosting for procedures and user-defined functions through config settings. These settings will be translated to temporary `execute boosted procedure` and `execute boosted function` privileges that cannot be revoked.

`dbms.security.procedures.default_allowed`

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

Example 67. Configure a default role that can execute procedures and user-defined functions

Assume that we have the following configuration:

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

This will create the following temporary privileges:

- `GRANT EXECUTE BOOSTED PROCEDURE * ON DBMS TO superAdmin`
- `GRANT EXECUTE BOOSTED USER DEFINED FUNCTION * ON DBMS TO superAdmin`
- If the setting dbms.security.procedures.roles has some roles to name defined, then for any procedure/function not also granted to the superAdmin role, create temporary privileges:
  - `DENY EXECUTE BOOSTED PROCEDURE name ON DBMS TO superAdmin`
  - `DENY EXECUTE BOOSTED USER DEFINED FUNCTION name ON DBMS TO superAdmin`

`dbms.security.procedures.roles`

The `dbms.security.procedures.roles` setting provides fine-grained control over procedures and user-defined functions.

Example 68. Configure roles for the execution of specific procedures and user-defined functions

Assume that we have the following configuration:

```
dbms.security.procedures.default_allowed=superAdmin
dbms.security.procedures.roles=apoc.coll.*:Collector;apoc.trigger.add:TriggerHappy,superAdmin
```

This will have create the following temporary privileges:

- `GRANT EXECUTE BOOSTED PROCEDURE apoc.coll.* ON DBMS TO Collector`
- `GRANT EXECUTE BOOSTED USER DEFINED FUNCTION apoc.coll.* ON DBMS TO Collector`
- `GRANT EXECUTE BOOSTED PROCEDURE apoc.trigger.add ON DBMS TO TriggerHappy, superAdmin`
- `GRANT EXECUTE BOOSTED USER DEFINED FUNCTION apoc.trigger.add ON DBMS TO TriggerHappy, superAdmin`
- `GRANT EXECUTE BOOSTED PROCEDURE * ON DBMS TO superAdmin`
- `GRANT EXECUTE BOOSTED USER DEFINED FUNCTION * ON DBMS TO superAdmin`
- `DENY EXECUTE BOOSTED PROCEDURE apoc.coll.* ON DBMS TO superAdmin`
- `DENY EXECUTE BOOSTED USER DEFINED FUNCTION apoc.coll.* ON DBMS TO superAdmin`

10.7. Terminology

The following terms are relevant to role-based access control within Neo4j:

active user

A user who is active within the system and can perform actions prescribed by any assigned roles on the data. This is in contrast to a suspended user.

administrator

This is a user who has been assigned the admin role.

current user

This is the currently logged-in user invoking the commands described in this chapter.

password policy

The password policy is a set of rules of what makes up a valid password. For Neo4j, the following rules apply:

- The password cannot be the empty string.
- When changing passwords, the new password cannot be the same as the previous password.

role

This is a collection of actions — such as read and write — permitted on the data.

suspended user

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

user

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

Chapter 11. Security

Ensure your physical data security by following industry best practices with regard to server and network security.

This chapter includes the following:

- Securing extensions
- SSL framework
- Credentials handling in Neo4j Browser
- Security checklist

Additionally, logs can be useful for continuous analysis, or for specific investigations. Facilities are available for producing security event logs as well as query logs as described in Monitoring.

Refer to Authentication and authorization for information on how to manage users and their authentication and authorization.

11.1. Securing extensions

Neo4j can be extended by writing custom code which can be invoked directly from Cypher, as described in Java Reference → User-defined functions. This section describes how to ensure the security of these additions.

11.1.1. Allow listing

Allow listing can be used to allow the loading of only a few extensions from a larger library.

The configuration setting `dbms.security.procedures.allowlist` is used to name certain procedures that should be available from a library. It defines a comma-separated list of procedures that are to be loaded. The list may contain both fully qualified procedure names, and partial names with the wildcard `*`.

Example 69. Allow 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 allow listing
dbms.security.procedures.allowlist=apoc.coll.*,apoc.load.*
```

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

- If using this setting, no extensions other than those listed will be loaded. In particular, if it is set to the empty string, no extensions will be loaded.

- The default of the setting is `*`. This means that if you do not explicitly give it a value (or no value), all libraries in the *plugins* directory will be loaded.

11.2. SSL framework

The SSL framework provides support for securing the following Neo4j communication channels using standard SSL/TLS technology:

- `bolt` (port - `7687`)
- `https` (port - `7473`)
- `cluster` (ports - `5000`, `6000`, `7000`, and `7688`)
- `backups` (port - `6362`)

11.2.1. SSL providers

The secure networking in Neo4j is provided through the Netty library, which supports both the native JDK SSL provider as well as Netty-supported OpenSSL derivatives.

Follow these steps to use OpenSSL:

- Install a suitable dependency into the `plugins/` folder of Neo4j.

Dependencies can be downloaded from https://netty.io/wiki/forked-tomcat-native.html. Which dependencies you need depends upon the Neo4j version. Each version of Neo4j ships with a version of Netty and Netty requires specific tcnative versions. Make sure to install the version that matches your OS processor. For more details, see the Netty support per Neo4j version.

- Using non static versions of tcnative will require installation of platform-specific OpenSSL dependencies as described in https://netty.io/wiki/forked-tomcat-native.html.
- Set `dbms.netty.ssl.provider=OPENSSL`.
- Restart Neo4j.

Most supported versions of Neo4j use Netty 4.1.77.Final, which requires tcnative 2.0.52. Only Neo4j 3.5 still uses older versions of Netty. See the table below for detailed information:

Table 43. Netty support per Neo4j version

Neo4j version	Netty version	tcnative version	Direct link
5.0	4.1.77.Final	2.0.52.Final. Both netty-tcnative-boringssl-static and netty-tcnative-classes are required	https://search.maven.org/artifact/io.netty/netty-tcnative-boringssl-static/2.0.52.Final/jar https://search.maven.org/artifact/io.netty/netty-tcnative-classes/2.0.52.Final/jar
4.4.9	4.1.77.Final	2.0.52.Final	https://search.maven.org/artifact/io.netty/netty-tcnative-boringssl-static/2.0.52.Final/jar https://search.maven.org/artifact/io.netty/netty-tcnative-classes/2.0.52.Final/jar

Neo4j version	Netty version	tcnative version	Direct link
4.3.15	4.1.77.Final	2.0.52.Final	https://search.maven.org/artifact/io.netty/netty-tcnative-boringssl-static/2.0.52.Final/jar https://search.maven.org/artifact/io.netty/netty-tcnative-classes/2.0.52.Final/jar
3.5.34	3.9.9.Final + 4.1.68.Final	2.0.42.Final	https://search.maven.org/artifact/io.netty/netty-tcnative/2.0.42.Final/jar

Using OpenSSL can significantly improve performance, especially for AES-GCM-cryptos, e.g. TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256.

11.2.2. Certificates

The SSL configuration requires SSL certificates to be issued by Certificate Authority (CA). All Certificates and the private key must be in PEM format.

If the same certificates are used across all instances of the cluster, make sure that when generating the certificates to include the DNS names of all the cluster instances in the certificates. Multi-host and wildcard certificates are also supported.

Valid trusted certificates can be generated for free using non-profit CAs such as Let's Encrypt.

The instructions on this page assume that you have already obtained the required certificates from the CA.

Validate the key and the certificate

If you need, you can validate the key file and the certificate as follows:

Validate the key

```
openssl rsa -in private.key -check
```

Validate certificate in the PEM format

```
PEM - $openssl x509 -in public.crt -text -noout
DER - $openssl x509 -in certificate.der -inform der -text -noout
```

Transform the certificates

Neo4j requires all SSL certificates to be in the PEM format. If your certificate is in the DER format, you must transform it into PEM format.

Transform DER format certificate to PEM format

```
openssl x509 -in cert.crt -inform der -outform pem -out cert.pem
```

Transform `PEM` format certificate to `DER` format

```
openssl x509 -in cert.crt -outform der -out cert.der
```

11.2.3. Connectors

Before enabling SSL support, you must ensure the following connector configurations to avoid errors:

- Set `dbms.connector.https.enabled` to `true` when using HTTPS.
- Set `dbms.connector.bolt.tls_level` to `REQUIRED` or `OPTIONAL` when using Bolt.

For more information on configuring connectors, see Configure connectors.

11.2.4. Configuration

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

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

- `scope` is the name of the communication channel, such as `bolt`, `https`, `cluster`, `backup`, and `fabric`.
- `setting-suffix` can be any of the following:

Setting suffix	Description	Default value
Basic		
`enabled`	Setting this to `true` enables this policy.	`false`
`base_directory`	The base directory under which <term-ssl-cryptographic-objects, cryptographic objects>> are searched for by default.	`certificates/<scope>`
`private_key`	The private key used for authenticating and securing this instance.	`private.key`
`private_key_password`	The passphrase to decode the private key. Only applicable for encrypted private keys.	
`public_certificate`	A public certificate matching the private key signed by a CA.	`public.crt`
`trusted_dir`	A directory populated with certificates of trusted parties.	`trusted/`
`revoked_dir`	A directory populated with certificate revocation lists (CRLs).	`revoked/`
Advanced		

Setting suffix	Description	Default value
`verify_hostname`	Enabling this setting turns on client-side hostname verification. After receiving the server's public certificate, the client compares the address it uses against the certificate Common Name (CN) and Subject Alternative Names (SAN) fields. If the address does not match those fields, the client disconnects.	`false`
`ciphers`	A comma-separated list of ciphers suits allowed during cipher negotiation. Valid values depend on the current JRE and SSL provider. For Ciphers supported by the Oracle JRE, see the Oracle official documentation.	Java platform default allowed cipher suites.
`tls_versions`	A comma-separated list of allowed TLS versions.	`TLSv1.2`
`client_auth`	Whether or not clients must be authenticated. Setting this to `REQUIRE` enables mutual authentication for servers. Other possible values are `NONE` and `OPTIONAL`.	`OPTIONAL` for `bolt` and `https`; `REQUIRE` for `cluster` and `backup`.
`trust_all`	Setting this to `true` results in all clients and servers to be trusted and the content of the `trusted_dir` directory to be ignored. Use this only as a mean of debugging, since it does not offer security.	`false`

For security reasons, Neo4j does not automatically create any of these directories. Therefore, the creation of an SSL policy requires the appropriate file system structure to be set up manually. Note that the existence of the directories, the certificate file, and the private key are mandatory. Ensure that only the Neo4j user can read the private key.

Each policy needs to be explicitly enabled by setting:

`dbms.ssl.policy.<scope>.enabled=true`

Configure SSL over Bolt

Bolt protocol is based on the PackStream serialization and supports the Cypher type system, protocol versioning, authentication, and TLS via certificates. For Neo4j clusters, Bolt provides smart client routing with load balancing and failover. When server side routing is enabled, an additional Bolt port is open on `7688`. It can be used only within the cluster and with all the same settings as the external Bolt port.

Bolt connector is used by Cypher Shell, Neo4j Browser, and by the officially supported language drivers. Bolt connector is enabled by default but its encryption is disabled. To enable the encryption over Bolt, create the folder structure and place the key file and the certificates under those. Then, you need to

configure the SSL Bolt policies in the *neo4j.conf* file.

1. Enable the Bolt connector to enable SSL over Bolt:

```
dbms.connector.bolt.enabled=true (default is true)
```

2. Set up the *bolt* folder under *certificates*.

   a. Create a directory *bolt* under *<neo4j-home>/certificates* folder:

```
mkdir certificates/bolt
```

   b. Create a directory *trusted* and *revoked* under *<neo4j-home>/certificates/bolt* folder:

```
mkdir certificates/bolt/trusted
mkdir certificates/bolt/revoked
```

3. Place the certificates *private.key* and the *public.crt* files under *<neo4j-home>/certificates/bolt* folder:

```
cp /path/to/certs/private.key certificates/bolt
cp /path/to/certs/public.crt certificates/bolt
```

4. Place the *public.crt* file under the *<neo4j-home>/certificates/bolt/trusted* folder.

```
cp /path/to/certs/public.crt certificates/bolt/trusted
```

5. (Optional) If a particular certificate is revoked, then place it under *<neo4j-home>/certificates/bolt/revoked* folder.

```
cp /path/to/certs/public.crt certificates/bolt/revoked
```

   The folder structure should look like this with the right file permissions and the groups and ownerships:

Path	Directory/File	Owner	Group	Permission	Unix/Linux View
/data/neo4j/certificates/bolt	Directory	neo4j	neo4j	0755	drwxr-xr-x
/data/neo4j/certificates/bolt/public.crt	File	neo4j	neo4j	0644	-rw-r--r--
/data/neo4j/certificates/bolt/private.key	File	neo4j	neo4j	0400	-r--------
/data/neo4j/certificates/bolt/trusted	Directory	neo4j	neo4j	0755	drwxr-xr-x
/data/neo4j/certificates/bolt/trusted/public.crt	File	neo4j	neo4j	0644	-rw-r--r--
/data/neo4j/certificates/bolt/revoked	Directory	neo4j	neo4j	0755	drwxr-xr-x

The owner/group should be configured to the user/group that will be running the `neo4j` service. Default user/group is neo4j/neo4j.

6. Set the Bolt SSL configuration in *neo4j.conf*.
   a. Set the SSL Bolt policy to `true`:

```
dbms.ssl.policy.bolt.enabled=true
```

   b. Set the appropriate certificates path and the right key and cert files:

```
dbms.ssl.policy.bolt.base_directory=certificates/bolt
dbms.ssl.policy.bolt.private_key=private.key
dbms.ssl.policy.bolt.public_certificate=public.crt
```

If the certificate is a different path outside of NEO4J_HOME, then set the absolute path for the certificates directory.

   c. Set the Bolt client authentication to `NONE` to disable the mutual authentication:

```
dbms.ssl.policy.bolt.client_auth=NONE
```

   d. Set the Bolt TLS level to allow the connector to accept encrypted and/or unencrypted connections:

```
dbms.connector.bolt.tls_level=REQUIRED (default is DISABLED)
```

In Neo4j version 3.5, the default value is `OPTIONAL`. In the Neo4j 4.x versions, the default value is `DISABLED`, where only unencrypted client connections are to be accepted by this connector, and all encrypted connections will be rejected. Use `REQUIRED` when only encrypted client connections are to be accepted by this connector, and all unencrypted connections will be rejected. Use `OPTIONAL` where either encrypted or unencrypted client connections are accepted by this connector.

7. Test the SSL connection to the specified host and Bolt port and view the certificate:

```
openssl s_client -connect my_domain.com:7687
```

Connect with SSL over Bolt

Each of the `neo4j` and `bolt` URI schemes permit variants that contain extra encryption and trust information. The `+s` variants enable encryption with a full certificate check. The `+ssc` variants enable encryption with no certificate check. This latter variant is designed specifically for use with self-signed certificates.

URI Scheme	Routing	Description
`neo4j`	Yes	Unsecured
`neo4j+s`	Yes	Secured with full certificate
`neo4j+ssc`	Yes	Secured with self-signed certificate
`bolt`	No	Unsecured
`bolt+s`	No	Secured with full certificate
`bolt+ssc`	No	Secured with self-signed certificate

Once SSL is enabled over Bolt, you can connect to the Neo4j DBMS using `neo4j+s` or `bolt+s`:

Cypher Shell

```
cypher-shell -a neo4j+s://<Server DNS or IP>:<Bolt port>

or

cypher-shell -a bolt+s://<Server DNS or IP>:<Bolt port>
```

Neo4j Browser

From the **Connect URL** dropdown menu, select the URI scheme you want to use (`neo4j+s` or `bolt+s`).

URI schemes ending `+ssc` are not supported by Neo4j Browser since the browser's OS handles certificate trust. If it is necessary to connect to a Neo4j instance using a self-signed certificate from Neo4j Browser, first visit a web page that uses the self-signed certificate in order to prompt the browser to request that certificate trust be granted. Once that trust has been granted, you can connect with URI schemes ending `+s`.

Configure SSL over HTTPS

HTTP(s) is used by the Neo4j Browser and the HTTP API. HTTPS (secure HTTP) is set to encrypt network communications. To enable the encryption over HTTPS, create the folder structure and place the key file and the certificates under those. Then, you need to configure the SSL HTTPS policies in the *neo4j.conf* file and disable the HTTP connector.

1. Enable the HTTPS connector to enable SSL over HTTPS:

   ```
   dbms.connector.https.enabled=true (default is false)
   ```

2. Set up the *https* folder under *certificates*.

   a. Create a directory *https* under *<neo4j-home>/certificates* folder:

      ```
      mkdir certificates/https
      ```

   b. Create a directory *trusted* and *revoked* under *<neo4j-home>/certificates/https* folder:

```
mkdir certificates/https/trusted
mkdir certificates/https/revoked
```

3. Place the certificates *private.key* and the *public.crt* files under *<neo4j-home>/certificates/https* folder:

```
cp /path/to/certs/private.key certificates/https
cp /path/to/certs/public.crt certificates/https
```

4. Place the *public.crt* file under the *<neo4j-home>/certificates/https/trusted* folder.

```
cp /path/to/certs/public.crt certificates/https/trusted
```

5. (Optional) If a particular certificate is revoked, then place it under *<neo4j-home>/certificates/https/revoked* folder.

```
cp /path/to/certs/public.crt certificates/https/revoked
```

The folder structure should look like this with the right file permissions and the groups and ownerships:

Path	Directory/File	Owner	Group	Permission	Unix/Linux View
/data/neo4j/certificates/https	Directory	neo4j	neo4j	0755	drwxr-xr-x
/data/neo4j/certificates/https/public.crt	File	neo4j	neo4j	0644	-rw-r--r--
/data/neo4j/certificates/https/private.key	File	neo4j	neo4j	0400	-r--------
/data/neo4j/certificates/https/trusted	Directory	neo4j	neo4j	0755	drwxr-xr-x
/data/neo4j/certificates/https/trusted/public.crt	File	neo4j	neo4j	0644	-rw-r--r--
/data/neo4j/certificates/https/revoked	Directory	neo4j	neo4j	0755	drwxr-xr-x

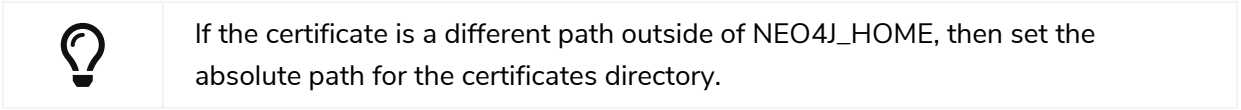
The owner/group should be configured to the user/group that will be running the `neo4j` service. Default user/group is neo4j/neo4j.

6. Set the HTTPS SSL configuration in *neo4j.conf*.
   a. Set the SSL HTTPS policy to `true`:

```
dbms.ssl.policy.https.enabled=true
```

   b. Set the appropriate certificates path and the right key and cert files:

```
dbms.ssl.policy.https.base_directory=certificates/https
dbms.ssl.policy.https.private_key=private.key
dbms.ssl.policy.https.public_certificate=public.crt
```



c. Set the HTTPS client authentication to NONE to disable the mutual authentication:

```
dbms.ssl.policy.https.client_auth=NONE
```

d. Disable HTTP connector:

```
dbms.connector.http.enabled=false
```

7. Test the SSL connection to the specified host and HTTPS port and view the certificate:

```
openssl s_client -connect my_domain.com:7473
```

Configure SSL for intra-cluster communications

Intra-cluster encryption is the security solution for the cluster communication. The Neo4j cluster communicates on 4 ports:

- 5000 - Discovery management
- 6000 - Transactions
- 7000 - Raft communications
- 7688 - Server side routing

To set up intra-cluster encryption, on each server create the folder structure and place the key file and the certificates under those. Then, you need to configure the SSL cluster policies in the *neo4j.conf* file and test that the intra-cluster communication is encrypted.

1. Set up the *cluster* folder under *certificates*.
   a. Create a directory *cluster* under_<neo4j-home>/certificates_ folder:

```
mkdir certificates/cluster
```

   b. Create a directory *trusted* and *revoked* under *<neo4j-home>/certificates/cluster* folder:

```
mkdir certificates/cluster/trusted
mkdir certificates/cluster/revoked
```

2. Place the certificates *private.key* and the *public.crt* files under *<neo4j-home>/certificates/cluster* folder:

```
cp /path/to/certs/private.key certificates/cluster
cp /path/to/certs/public.crt certificates/cluster
```

3. Place the *public.crt* file under the *<neo4j-home>/certificates/cluster/trusted* folder.

```
cp /path/to/certs/public.crt certificates/cluster/trusted
```

If each server has a certificate of its own, signed by a CA, then each server's public certificate has to be put in the *trusted* folder on each instance of the cluster. Thus, the servers are able to establish trust relationships with each other.

4. (Optional) If a particular certificate is revoked, then place it under *<neo4j-home>/certificates/cluster/revoked* folder.

```
cp /path/to/certs/public.crt certificates/cluster/revoked
```

The folder structure should look like this with the right file permissions and the groups and ownerships:

Path	Directory/File	Owner	Group	Permission	Unix/Linux View
/data/neo4j/certificates/cluster	Directory	neo4j	neo4j	0755	drwxr-xr-x
/data/neo4j/certificates/cluster/public.crt	File	neo4j	neo4j	0644	-rw-r--r--
/data/neo4j/certificates/cluster/private.key	File	neo4j	neo4j	0400	-r--------
/data/neo4j/certificates/cluster/trusted	Directory	neo4j	neo4j	0755	drwxr-xr-x
/data/neo4j/certificates/cluster/trusted/public.crt	File	neo4j	neo4j	0644	-rw-r--r--
/data/neo4j/certificates/cluster/revoked	Directory	neo4j	neo4j	0755	drwxr-xr-x

The owner/group should be configured to the user/group that will be running the `neo4j` service. Default user/group is neo4j/neo4j.

5. Set the cluster SSL configuration in *neo4j.conf*.
   a. Set the cluster SSL policy to `true`:

```
dbms.ssl.policy.cluster.enabled=true
```

   b. Set the appropriate certificates path and the right key and cert files:

```
dbms.ssl.policy.cluster.base_directory=certificates/cluster
dbms.ssl.policy.cluster.private_key=private.key
dbms.ssl.policy.cluster.public_certificate=public.crt
```

If the certificate is a different path outside of NEO4J_HOME, then set the absolute path for the certificates directory.

   c. Set the cluster client authentication to `REQUIRE` to enable the mutual authentication, which means that both ends of a channel have to authenticate:

```
dbms.ssl.policy.cluster.client_auth=REQUIRE
```

The policy must be configured on every server with the same settings. The actual cryptographic objects installed will be mostly different since they do not share the same private keys and corresponding certificates. The trusted CA certificate will be shared however.

d. Verify that the intra-cluster communication is encrypted. You may use an external tooling, such as Nmap (https://nmap.org/download.html):

```
nmap --script ssl-enum-ciphers -p <port> <hostname>
```

The hostname and port have to be adjusted according to your configuration. This can prove that TLS is in fact enabled and that only the intended cipher suites are enabled. All servers and all applicable ports should be tested. If the intra-cluster encryption is enabled, the output should indicate the port is open and it is using TLS with the ciphers used.

For more details on securing the comunication between the cluster servers, see Intra-cluster encryption.

Configure SSL for backup communication

In a single instance, by default the backup communication happens on port `6362`. In a cluster topology, it is possible to take a backup from any server, and each server has two configurable ports capable of serving a backup. These ports are configured by `dbms.backup.listen.address` (port `6362`) and `causal_clustering.transaction_listen_address` (port `6000`) respectively. If the intra-cluster encryption is enabled and the backup communication is using port `6000`, then your communication channels are already encrypted. The following steps assumes that your backup is set up on a different port.

To set up SSL for backup communication, create the folder structure and place the key file and the certificates under those. Then, you need to configure the SSL backup policies in the *neo4j.conf* file.

1. Set up the *backup* folder under *certificates*.
   a. Create a directory *backup* under *<neo4j-home>/certificates* folder:

```
mkdir certificates/backup
```

   b. Create a directory *trusted* and *revoked* under *<neo4j-home>/certificates/backup* folder:

```
mkdir certificates/backup/trusted
mkdir certificates/backup/revoked
```

2. Place the certificates *private.key* and the *public.crt* files under *<neo4j-home>/certificates/backup* folder:

```
cp /path/to/certs/private.key certificates/backup
cp /path/to/certs/public.crt certificates/backup
```

3. Place the *public.crt* file under the *<neo4j-home>/certificates/backup/trusted* folder.

```
cp /path/to/certs/public.crt certificates/backup/trusted
```

4. (Optional) If a particular certificate is revoked, then place it under *<neo4j-home>/certificates/backup/revoked* folder.

```
cp /path/to/certs/public.crt certificates/backup/revoked
```

The folder structure should look like this with the right file permissions and the groups and ownerships:

Path	Directory/File	Owner	Group	Permission	Unix/Linux View
/data/neo4j/certificates/backup	Directory	neo4j	neo4j	0755	drwxr-xr-x
/data/neo4j/certificates/backup/public.crt	File	neo4j	neo4j	0644	-rw-r--r--
/data/neo4j/certificates/backup/private.key	File	neo4j	neo4j	0400	-r--------
/data/neo4j/certificates/backup/trusted	Directory	neo4j	neo4j	0755	drwxr-xr-x
/data/neo4j/certificates/backup/trusted/public.crt	File	neo4j	neo4j	0644	-rw-r--r--
/data/neo4j/certificates/backup/revoked	Directory	neo4j	neo4j	0755	drwxr-xr-x

The owner/group should be configured to the user/group that will be running the `neo4j` service. Default user/group is neo4j/neo4j.

5. Set the backup SSL configuration in *neo4j.conf*.
   a. Set the backup SSL policy to `true`:

```
dbms.ssl.policy.backup.enabled=true
```

   b. Set the appropriate certificates path and the right key and cert files:

```
dbms.ssl.policy.backup.base_directory=certificates/backup
dbms.ssl.policy.backup.private_key=private.key
dbms.ssl.policy.backup.public_certificate=public.crt
```

If the certificate is a different path outside of NEO4J_HOME, then set the absolute path for the certificates directory.

   c. Set the backup client authentication to `REQUIRE` to enable the mutual authentication, which means that both ends of a channel have to authenticate:

```
dbms.ssl.policy.backup.client_auth=REQUIRE
```

Other configurations for SSL

Using encrypted private key

To use an encrypted private key, configure the following settings. The private key password must be clear text format without any quotes.

Bolt

```
dbms.ssl.policy.bolt.private_key_password=<clear text password>
```

HTTPS

```
dbms.ssl.policy.https.private_key_password=<password>
```

Intra-cluster encryption

```
dbms.ssl.policy.cluster.private_key_password=<password>
```

Backup

```
dbms.ssl.policy.backup.private_key_password=<password>
```

If hardcoding of clear text private key password is not feasible due to security constraints, it can be set up to use dynamic password pickup by following these steps:

1. Create a file containing the `cleartext` password for the private key password and encrypt it with the certificate (assuming private key for cert has password set and certificate is in `pwd`):

   ```
   echo "password123" > passwordfile

   openssl aes-256-cbc -a -salt -in passwordfile -out password.enc -pass file:certificate.crt
   ```

   Delete the password file and set file permissions for `password.enc` to `400` (e.g. `chmod 400 password.enc`).

2. Verify that encrypted password can be read from password.enc:

   ```
   openssl aes-256-cbc -a -d -in password.enc -pass file:certificate.crt
   ```

3. Set the neo4j.conf `dbms.ssl.policy.<type>.private_key_password` to be able to read out encrypted password. To adjust paths to cert and encrypted password file, use full paths:

   ```
   dbms.ssl.policy.bolt.private_key_password=$(openssl aes-256-cbc -a -d -in password.enc -pass
   file:certificate.crt)
   ```

Using a dynamic command requires Neo4j to be started with the `--expand-commands` option. For more information, see Command expansion.

Using specific cipher

There are cases where Neo4j Enterprise requires the use of specific ciphers for encryptions. One can set up a Neo4j configuration by specifying the list of cipher suits that will be allowed during cipher negotiation. Valid values depend on the current JRE and SSL provider. For Oracle JRE here is the list of supported ones - https://docs.oracle.com/en/java/javase/11/docs/specs/security/standard-names.html#jsse-cipher-suite-names.

Bolt

```
dbms.ssl.policy.bolt.ciphers=TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
,TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
```

HTTPS

```
dbms.ssl.policy.https.ciphers=TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA38
4,TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
```

Intra-cluster encryption

```
dbms.ssl.policy.cluster.ciphers=TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA
384,TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
```

Backup

```
dbms.ssl.policy.backup.ciphers=TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA3
84,TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
```

Using OCSP stapling

From version 4.2, Neo4j supports OCSP stapling, which is implemented on the server side, and can be configured in the *neo4j.config* file. OCSP stapling is also available for Java Bolt driver and HTTP API.

On the server side in the *neo4j.conf* file, configure the following settings:

1. Set the SSL Bolt policy to `true`:

   ```
   dbms.ssl.policy.bolt.enabled=true
   ```

2. Enable the OCSP stapling for Bolt:

   ```
   dbms.connector.bolt.ocsp_stapling_enabled=true (default = false)
   ```

11.2.5. SSL logs

All information related to SSL can be found in the *debug.log* file. You can also enable additional debug logging for SSL by adding the following configuration to the *neo4j.conf* file and restarting Neo4j.

```
dbms.jvm.additional=-Djavax.net.debug=ssl:handshake
```

This will log additional information in the *neo4j.log* file. In some installations done using `rpm` based installs, *neo4j.log* is not created. To get the contents of this, since *neo4j.log* just contains `STDOUT` content, look for the `neo4j` service log contents using `journalctl`:

```
neo4j@ubuntu:/var/log/neo4j$ journalctl -u neo4j -b > neo4j.log
neo4j@ubuntu:/var/log/neo4j$ vi neo4j.log
```

Beware that the SSL debug option logs a new statement every time a client connects over SSL, which can make *neo4j.log* grow large reasonably quickly. To avoid that scenario, make sure this setting is only enabled for a short term duration.

11.2.6. Terminology

The following terms are relevant to SSL support within Neo4j:

Certificate Authority (CA)

A trusted entity that issues electronic documents that can verify the identity of a digital entity. The term commonly refers to globally recognized CAs, but can also include internal CAs that are trusted inside of an organization. The electronic documents are digital certificates. They are an essential part of secure communication, and play an important part in the Public Key Infrastructure.

Certificate Revocation List (CRL)

In the event of a certificate being compromised, that certificate can be revoked. This is done by means of a list (located in one or several files) spelling out which certificates are revoked. The CRL is always issued by the CA which issues the corresponding certificates.

cipher

An algorithm for performing encryption or decryption. In the most general implementation of encryption of Neo4j communications, we make implicit use of ciphers that are included as part of the Java platform. The configuration of the SSL framework also allows for the explicit declaration of allowed ciphers.

communication channel

A means for communicating with the Neo4j database. Available channels are:

- Bolt client traffic
- HTTPS client traffic
- intra-cluster communication
- backup traffic

cryptographic objects

A term denoting the artifacts private keys, certificates and CRLs.

configuration parameters

These are the parameters defined for a certain ssl policy in *neo4j.conf*.

certificate

SSL certificates are issued by a trusted certificate authority (CA). The public key can be obtained and used by anyone to encrypt messages intended for a particular recipient. The certificate is commonly stored in a file named *<file name>.crt*. This is also referred to as the public key.

SAN

SAN is an acronym for *Subject Alternative Names*. It is an extension to certificates that one can include optionally. When presented with a certificate that includes SAN entries, it is recommended that the address of the host is checked against this field. Verifying that the hostname matches the certificate SAN helps prevent attacks where a rogue machine has access to a valid key pair.

SSL

SSL is an acronym for *Secure Sockets Layer*, and is the predecessor of TLS. It is common to refer to SSL/TLS as just SSL. However, the modern and secure version is TLS, and this is also the default in Neo4j.

SSL policy

An SSL policy in Neo4j consists of a a digital certificate and a set of configuration parameters defined in *neo4j.conf*.

private key

The private key ensures that encrypted messages can be deciphered only by the intended recipient. The private key is commonly stored in a file named *<file name>.key*. It is important to protect the private key to ensure the integrity of encrypted communication.

Public Key Infrastructure (PKI)

A set of roles, policies, and procedures needed to create, manage, distribute, use, store, and revoke digital certificates and manage public-key encryption.

public key

The public key can be obtained and used by anyone to encrypt messages intended for a particular recipient. This is also referred to as the certificate.

TLS protocol

The cryptographic protocol that provides communications security over a computer network. The Transport Layer Security (TLS) protocol and its predecessor, the Secure Sockets Layer (SSL) protocol are both frequently referred to as "SSL".

TLS version

A version of the TLS protocol.

11.3. Browser credentials handling

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.

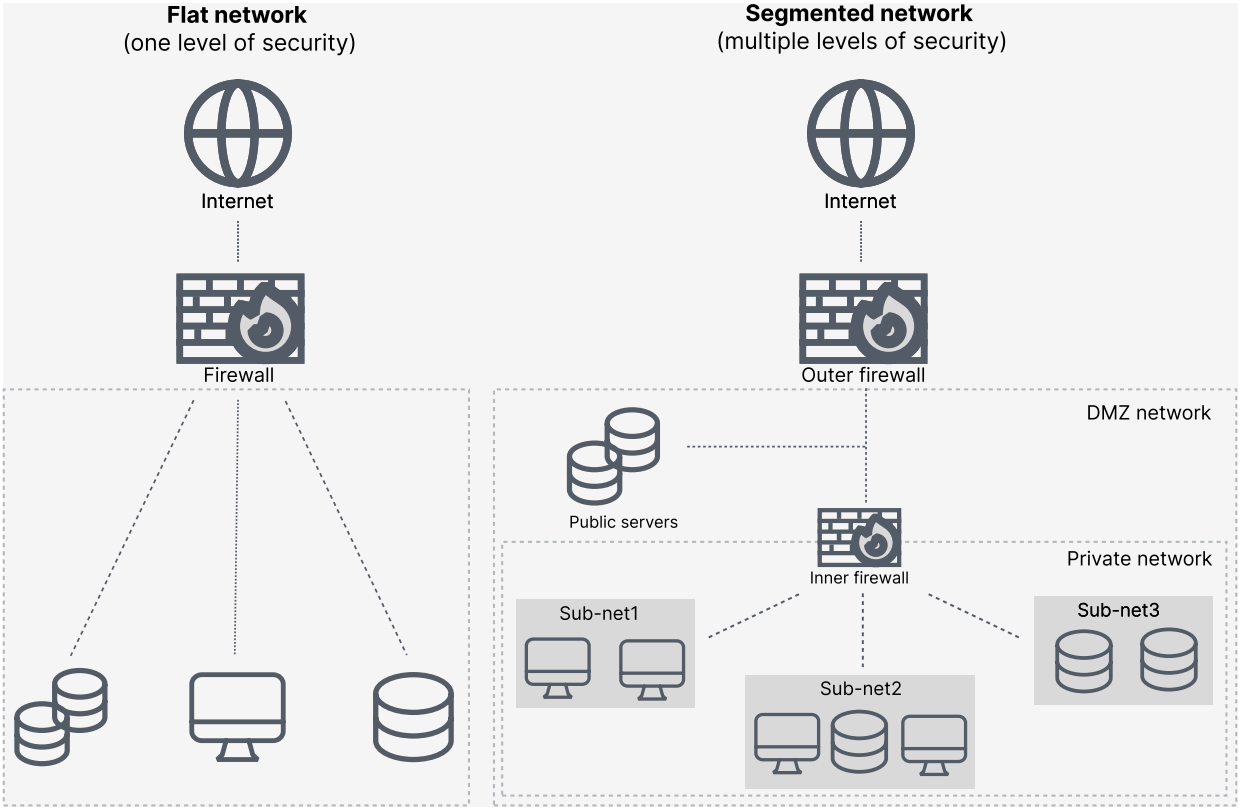
If the user issues a `:server disconnect` command then any existing session is terminated and the credentials are cleared from local storage.

For more information on how to administer and use Neo4j Browser, see the Neo4j Browser manual → Browser operations.

11.4. Security checklist

The following checklist highlights the specific areas within Neo4j that may need extra attention to ensure the appropriate level of security for your application after Neo4j is deployed.

1. Deploy Neo4j on safe servers in secure networks:
    a. Use subnets and firewalls to segment the network.



b. Open only the ports that you need. For a list of relevant ports, see Ports.

   In particular, ensure that there is no external access to the port specified by the setting `dbms.backup.listen_address`. Failing to protect this port may open a security hole by which an unauthorized user can make a copy of the database onto a different machine.

2. Protect data-at-rest:
   a. Use volume encryption (e.g., Bitlocker).
   b. Manage access to database dumps and backups. Refer to Back up an offline database and backups Back up an online database for more information.
   c. Manage access to configuration files, data files, and transaction logs by ensuring the correct file permissions on the Neo4j files. Refer to File permissions for instructions on permission levels.
3. Protect data-in-transit:
   a. For remote access to the Neo4j database, only use encrypted Bolt or HTTPS.
   b. Use SSL certificates issued from a trusted Certificate Authority.
   c. For configuring your Neo4j installation to use encrypted communication, refer to SSL framework.
   d. If using Causal Clustering, configure and use encryption for intra-cluster communication. For details, see Intra-cluster encryption.
   e. If using Causal Clustering, configure and use encryption for backups. This ensures that only servers with the specified SSL policy and SSL certificates can access the server and perform the backup.
   f. For configuring your Bolt and HTTPS connectors, refer to Configure connectors.
   g. If using LDAP, configure your LDAP system with encryption via StartTLS. For more information, see Use LDAP with encryption via StartTLS.
4. Be on top of the security for custom extensions:

a. Validate any custom code you deploy (procedures and unmanaged extensions) and ensure that they do not unintentionally expose any parts of the product or data.

b. Survey the settings `dbms.security.procedures.unrestricted` and `dbms.security.procedures.allowlist` to ensure that they exclusively contain intentionally exposed extensions.

5. Make sure you have the correct file permissions on the Neo4j files.
6. Protect against the execution of unauthorized extensions by restricting access to the *bin*, *lib*, and *plugins* directories. Only the operating system user that Neo4j runs as should have permissions to those files. Refer to File permissions for instructions on permission levels.
7. With `LOAD CSV` enabled, ensure that it does not allow unauthorized users to import data. How to configure `LOAD CSV` is described in Cypher Manual → `LOAD CSV`.
8. Use Neo4j authentication. The setting `dbms.security.auth_enabled` controls native authentication. The default value is `true`.
9. Survey your JVM-specific configuration settings in the *neo4j.conf* file for ports relating to deprecated functions, such as remote JMX (controlled by the parameter setting `dbms.jvm.additional=-Dcom.sun.management.jmxremote.port=3637`).
10. Review Browser credentials handling to determine whether the default credentials handling in Neo4j Browser complies with your security regulations. Follow the instructions to configure it if necessary.
11. Use the latest patch version of Neo4j and set up a process to update it when security advisories are published.

Chapter 12. Monitoring

Neo4j provides mechanisms for continuous analysis through the output of metrics as well as the inspection and management of currently-executing queries.

Logs can be harvested for continuous analysis, or for specific investigations. Facilities are available for producing security event logs as well as query logs. The query management functionality is provided for specific investigations into query performance. Monitoring features are also provided for ad-hoc analysis of a Causal Cluster.

This chapter describes the following:

12.1. Metrics

This section describes the following:

12.2. Types of metrics

Neo4j provides a built-in metrics subsystem. Reported metrics can be queried via JMX, retrieved from CSV files, or consumed by third-party monitoring tools.

Neo4j has the following types of metrics:

- Global - covers the whole Neo4j DBMS.
- Per database - covers an individual database.

12.2.1. Global metrics

Global metrics cover the whole database management system, and represents the status of the system as a whole.

Global metrics have the following name format: `<user-configured-prefix>.<metric-name>` if `metrics.namespaces.enabled` is `false`, or `<user-configured-prefix>.dbms.<metric-name>` if the setting is `true`.

Metrics of this type are reported as soon as the database management system is available. For example, all JVM related metrics are global. In particular, the `neo4j.vm.thread.count` metric has a default user-configured-prefix `neo4j` and the metric name is `vm.thread.count`.

By default, global metrics include:

- Page cache metrics
- GC metrics
- Thread metrics
- Memory pool metrics
- Memory buffers metrics
- File descriptor metrics
- Database operation metrics
- Bolt metrics
- Web Server metrics

12.2.2. Database metrics

Each database metric is reported for a particular database only. Database metrics are only available during the lifetime of the database. When a database becomes unavailable, all of its metrics become unavailable also.

Database metrics have the following name format: `<user-configured-prefix>.<database-name>.<metric-name>` if `metrics.namespaces.enabled` is `false`, or `<user-configured-prefix>.database.<database-name>.<metric-name>` if the setting is `true`.

For example, any transaction metric is a database metric. In particular, the `neo4j.mydb.transaction.started` metric has a default user-configured-prefix `neo4j` and it is a metric for the `mydb` database.

By default, database metrics include:

- Transaction metrics
- Checkpoint metrics
- Log rotation metrics
- Database data metrics
- Cypher metrics
- Causal clustering metrics

12.2.3. Expose metrics

Enable metrics logging

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

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

By default, metrics logging into CSV files is enabled. A subset of metrics are enabled once `metrics.enabled=true` is set, and you can use the `metrics.filter` setting to select the specific metrics you want to enable.

The `metrics.filter` should be specified as a comma separated list of globbing patterns. For example, `*check_point*,neo4j.dbms.page_cache.evictions` will enable all checkpoint metrics and the pagecache eviction metric. When specifying a complete metric name, you should take into account whether `metrics.namespaces.enabled` is set:

```
# Setting for enabling all supported metrics.
metrics.enabled=true

# Setting for enabling clear separation between global and database metrics.
metrics.namespaces.enabled=true

# Setting for exposing metrics. Should be specified as a comma separated list of globbing patterns.
metrics.filter=*causal_clustering*,*check_point*,neo4j.dbms.page_cache.evictions
```

Graphite

Send metrics to Graphite or any monitoring tool based on the Graphite protocol.

Add the following settings to *neo4j.conf* in order to enable integration with Graphite:

```
# Enable the Graphite integration. Default is 'false'.
metrics.graphite.enabled=true
# The IP and port of the Graphite server on the format <hostname or IP address>:<port number>.
# The default port number for Graphite is 2003.
metrics.graphite.server=localhost:2003
# How often to send data. Default is 30 seconds.
metrics.graphite.interval=30s
# Prefix for Neo4j metrics on Graphite server.
metrics.prefix=Neo4j_1
```

Start Neo4j and connect to Graphite via a web browser in order to monitor your Neo4j metrics.

If you configure the Graphite server to be a hostname or DNS entry you should be aware that the JVM resolves hostnames to IP addresses and by default caches the result indefinitely for security reasons. This is controlled by the value of `networkaddress.cache.ttl` in the JVM Security properties. See https://docs.oracle.com/javase/8/docs/technotes/guides/net/properties.html for more information.

Prometheus

Publish metrics for polling as Prometheus endpoint.

Add the following settings to *neo4j.conf* in order to enable the Prometheus endpoint.

```
# Enable the Prometheus endpoint. Default is 'false'.
metrics.prometheus.enabled=true
# The IP and port the endpoint will bind to in the format <hostname or IP address>:<port number>.
# The default is localhost:2004.
metrics.prometheus.endpoint=localhost:2004
```

When Neo4j is fully started, a Prometheus endpoint will be available at the configured address.

You should never expose the Prometheus endpoint directly to the Internet. If security is of paramount importance, you should set `metrics.prometheus.endpoint=localhost:2004` and configure a reverse HTTP proxy on the same machine that handles the authentication, SSL, caching, etc.

If you can afford to send unencrypted metrics within the internal network, such as `metrics.prometheus.endpoint=10.0.0.123:2004`, all servers within the same netmask will be able to access it.

If you specify anything more permissible, such as `metrics.prometheus.endpoint=0.0.0.0:2004`, you should have a firewall rule to prevent any unauthorized access. Data in transit will still not be encrypted, so it should never go other any insecure networks.

CSV files

Export metrics to CSV files.

Add the following settings to *neo4j.conf* in order to enable export of metrics into local .CSV files:

```
# Enable the CSV exporter. Default is 'true'.
metrics.csv.enabled=true
# Directory path for output files.
# Default is a "metrics" directory under NEO4J_HOME.
#dbms.directories.metrics='/local/file/system/path'
# How often to store data. Default is 30 seconds.
metrics.csv.interval=30s
# The maximum number of CSV files that will be saved. Default is 7.
metrics.csv.rotation.keep_number=7
# The file size at which the csv files will auto-rotate. Default is 10M.
metrics.csv.rotation.size=10M
# Compresses the metric archive files.
metrics.csv.rotation.compression=zip
```

metrics.csv.rotation.compression selects the compression scheme to use on the files after rotation. Since CSV files are highly compressible, it is recommended to enable compression of the files to save disk space.

JMX MBeans

Expose metrics over JMX MBeans.

In version 4.2.2 and later, metrics via JMX are exposed by default. In version 4.2.0 and 4.2.1, you can enable them by adding the following setting to *neo4j.conf*:

```
# Enable settings export via JMX.
metrics.jmx.enabled=true
```

For more information about accessing and adjusting the metrics, see The Java Reference Guide → JMX metrics.

12.2.4. Metrics reference

General-purpose metrics

Table 44. Database store size metrics

Name	Description
`<prefix>.store.size.total`	The total size of the database and transaction logs, in bytes. (gauge)
`<prefix>.store.size.database`	The size of the database, in bytes. (gauge)

Table 45. Database checkpointing metrics

Name	Description
`<prefix>.check_point.events`	The total number of check point events executed so far. (counter)
`<prefix>.check_point.total_time`	The total time, in milliseconds, spent in check pointing so far. (counter)
`<prefix>.check_point.duration`	The duration, in milliseconds, of the last check point event. (gauge)

Table 46. Database data metrics

Name	Description
`<prefix>.ids_in_use.relationship_type`	The total number of different relationship types stored in the database. (gauge)
`<prefix>.ids_in_use.property`	The total number of different property names used in the database. (gauge)
`<prefix>.ids_in_use.relationship`	The total number of relationships stored in the database. (gauge)
`<prefix>.ids_in_use.node`	The total number of nodes stored in the database. (gauge)

Table 47. Database page cache metrics

Name	Description
`<prefix>.page_cache.eviction_exceptions`	The total number of exceptions seen during the eviction process in the page cache. (counter)
`<prefix>.page_cache.flushes`	The total number of page flushes executed by the page cache. (counter)
`<prefix>.page_cache.merges`	The total number of page merges executed by the page cache. (counter)
`<prefix>.page_cache.unpins`	The total number of page unpins executed by the page cache. (counter)
`<prefix>.page_cache.pins`	The total number of page pins executed by the page cache. (counter)
`<prefix>.page_cache.evictions`	The total number of page evictions executed by the page cache. (counter)
`<prefix>.page_cache.page_faults`	The total number of page faults happened in the page cache. (counter)

Name	Description
<prefix>.page_cache.hits	The total number of page hits happened in the page cache. (counter)
<prefix>.page_cache.hit_ratio	The ratio of hits to the total number of lookups in the page cache. (gauge)
<prefix>.page_cache.usage_ratio	The ratio of number of used pages to total number of available pages. (gauge)
<prefix>.page_cache.bytes_read	The total number of bytes read by the page cache. (counter)
<prefix>.page_cache.bytes_written	The total number of bytes written by the page cache. (counter)

Table 48. Database transaction metrics

Name	Description
<prefix>.transaction.started	The total number of started transactions. (counter)
<prefix>.transaction.peak_concurrent	The highest peak of concurrent transactions. (counter)
<prefix>.transaction.active	The number of currently active transactions. (gauge)
<prefix>.transaction.active_read	The number of currently active read transactions. (gauge)
<prefix>.transaction.active_write	The number of currently active write transactions. (gauge)
<prefix>.transaction.committed	The total number of committed transactions. (counter)
<prefix>.transaction.committed_read	The total number of committed read transactions. (counter)
<prefix>.transaction.committed_write	The total number of committed write transactions. (counter)
<prefix>.transaction.rollbacks	The total number of rolled back transactions. (counter)
<prefix>.transaction.rollbacks_read	The total number of rolled back read transactions. (counter)
<prefix>.transaction.rollbacks_write	The total number of rolled back write transactions. (counter)
<prefix>.transaction.terminated	The total number of terminated transactions. (counter)
<prefix>.transaction.terminated_read	The total number of terminated read transactions. (counter)
<prefix>.transaction.terminated_write	The total number of terminated write transactions. (counter)
<prefix>.transaction.last_committed_tx_id	The ID of the last committed transaction. (counter)
<prefix>.transaction.last_closed_tx_id	The ID of the last closed transaction. (counter)
<prefix>.transaction.tx_size_heap	The transactions' size on heap in bytes. (histogram)

Name	Description
<prefix>.transaction.tx_size_native	The transactions' size in native memory in bytes. (histogram)

Table 49. Cypher metrics

Name	Description
<prefix>.cypher.replan_events	The total number of times Cypher has decided to re-plan a query. (counter)
<prefix>.cypher.replan_wait_time	The total number of seconds waited between query replans. (counter)

Table 50. Database transaction log metrics

Name	Description
<prefix>.log.rotation_events	The total number of transaction log rotations executed so far. (counter)
<prefix>.log.rotation_total_time	The total time, in milliseconds, spent in rotating transaction logs so far. (counter)
<prefix>.log.rotation_duration	The duration, in milliseconds, of the last log rotation event. (gauge)
<prefix>.log.appended_bytes	The total number of bytes appended to transaction log. (counter)

Table 51. Bolt metrics

Name	Description
<prefix>.bolt.sessions_started	The total number of Bolt sessions started since this instance started. This includes both succeeded and failed sessions (deprecated, use connections_opened instead). (counter)
<prefix>.bolt.connections_opened	The total number of Bolt connections opened since this instance started. This includes both succeeded and failed connections. (counter)
<prefix>.bolt.connections_closed	The total number of Bolt connections closed since this instance started. This includes both properly and abnormally ended connections. (counter)
<prefix>.bolt.connections_running	The total number of Bolt connections currently being executed. (gauge)
<prefix>.bolt.connections_idle	The total number of Bolt connections sitting idle. (gauge)
<prefix>.bolt.messages_received	The total number of messages received via Bolt since this instance started. (counter)
<prefix>.bolt.messages_started	The total number of messages that began processing since this instance started. This is different from messages received in that this counter tracks how many of the received messages havebeen taken on by a worker thread. (counter)
<prefix>.bolt.messages_done	The total number of messages that completed processing since this instance started. This includes successful, failed and ignored Bolt messages. (counter)
<prefix>.bolt.messages_failed	The total number of messages that failed processing since this instance started. (counter)
<prefix>.bolt.accumulated_queue_time	The accumulated time messages have spent waiting for a worker thread. (counter)

Name	Description
<prefix>.bolt.accumulated_processing_time	The accumulated time worker threads have spent processing messages. (counter)

Table 52. Database data count metrics

Name	Description
<prefix>.neo4j.count.relationship	The total number of relationships in the database. (gauge)
<prefix>.neo4j.count.node	The total number of nodes in the database. (gauge)

Table 53. Database operation count metrics

Name	Description
<prefix>.db.operation.count.create	Count of successful database create operations. (counter)
<prefix>.db.operation.count.start	Count of successful database start operations. (counter)
<prefix>.db.operation.count.stop	Count of successful database stop operations. (counter)
<prefix>.db.operation.count.drop	Count of successful database drop operations. (counter)
<prefix>.db.operation.count.failed	Count of failed database operations. (counter)
<prefix>.db.operation.count.recovered	Count of database operations which failed previously but have recovered. (counter)

Table 54. Server metrics

Name	Description
<prefix>.server.threads.jetty.idle	The total number of idle threads in the jetty pool. (gauge)
<prefix>.server.threads.jetty.all	The total number of threads (both idle and busy) in the jetty pool. (gauge)

Chapter 13. Metrics specific to Causal Clustering

Table 55. Raft core metrics

Name	Description
<prefix>.causal_clustering.core.append_index	Append index of the RAFT log. (gauge)
<prefix>.causal_clustering.core.commit_index	Commit index of the RAFT log. (gauge)
<prefix>.causal_clustering.core.applied_index	Applied index of the RAFT log. (gauge)
<prefix>.causal_clustering.core.term	RAFT Term of this server. (gauge)
<prefix>.causal_clustering.core.tx_retries	Transaction retries. (counter)
<prefix>.causal_clustering.core.is_leader	Is this server the leader? (gauge)
<prefix>.causal_clustering.core.in_flight_cache.total_bytes	In-flight cache total bytes. (gauge)
<prefix>.causal_clustering.core.in_flight_cache.max_bytes	In-flight cache max bytes. (gauge)
<prefix>.causal_clustering.core.in_flight_cache.element_count	In-flight cache element count. (gauge)
<prefix>.causal_clustering.core.in_flight_cache.max_elements	In-flight cache maximum elements. (gauge)
<prefix>.causal_clustering.core.in_flight_cache.hits	In-flight cache hits. (counter)
<prefix>.causal_clustering.core.in_flight_cache.misses	In-flight cache misses. (counter)
<prefix>.causal_clustering.core.message_processing_delay	Delay between RAFT message receive and process. (gauge)
<prefix>.causal_clustering.core.message_processing_timer	Timer for RAFT message processing. (counter, histogram)
<prefix>.causal_clustering.core.replication_new	Raft replication new request count. (counter)
<prefix>.causal_clustering.core.replication_attempt	Raft replication attempt count. (counter)

Name	Description
<prefix>.causal_clustering.core.replication_fail	Raft Replication fail count. (counter)
<prefix>.causal_clustering.core.replication_maybe	Raft Replication maybe count. (counter)
<prefix>.causal_clustering.core.replication_success	Raft Replication success count. (counter)
<prefix>.causal_clustering.core.last_leader_message	Time elapsed since last message from leader in milliseconds. (gauge)

Table 56. Read Replica Metrics

Name	Description
<prefix>.causal_clustering.read_replica.pull_updates	The total number of pull requests made by this instance. (counter)
<prefix>.causal_clustering.read_replica.pull_update_highest_tx_id_requested	The highest transaction id requested in a pull update by this instance. (counter)
<prefix>.causal_clustering.read_replica.pull_update_highest_tx_id_received	The highest transaction id that has been pulled in the last pull updates by this instance. (counter)

Table 57. Discovery core metrics

Name	Description
<prefix>.causal_clustering.core.discovery.replicated_data	Size of replicated data structures. (gauge)
<prefix>.causal_clustering.core.discovery.cluster.members	Discovery cluster member size. (gauge)
<prefix>.causal_clustering.core.discovery.cluster.unreachable	Discovery cluster unreachable size. (gauge)
<prefix>.causal_clustering.core.discovery.cluster.converged	Discovery cluster convergence. (gauge)

Chapter 14. Java Virtual Machine Metrics

These metrics are environment dependent and they may vary on different hardware and with JVM configurations. Typically these metrics will show information about garbage collections (for example the number of events and time spent collecting), memory pools and buffers, and finally the number of active threads running.

Table 58. GC metrics.

Name	Description
`<prefix>.vm.gc.time.%s`	Accumulated garbage collection time in milliseconds. (counter)
`<prefix>.vm.gc.count.%s`	Total number of garbage collections. (counter)

Table 59. JVM memory buffers metrics.

Name	Description
`<prefix>.vm.memory.buffer.%s.count`	Estimated number of buffers in the pool. (gauge)
`<prefix>.vm.memory.buffer.%s.used`	Estimated amount of memory used by the pool. (gauge)
`<prefix>.vm.memory.buffer.%s.capacity`	Estimated total capacity of buffers in the pool. (gauge)

Table 60. JVM memory pools metrics.

Name	Description
`<prefix>.vm.memory.pool.%s`	Estimated number of buffers in the pool. (gauge)

Table 61. JVM threads metrics.

Name	Description
`<prefix>.vm.thread.count`	Estimated number of active threads in the current thread group. (gauge)
`<prefix>.vm.thread.total`	The total number of live threads including daemon and non-daemon threads. (gauge)

14.1. Logging

14.1.1. Log files Enterprise edition

Neo4j provides logs for monitoring purposes. The root directory where the general log files are located is configured by `dbms.directories.logs`. For more information on where files are located, see File locations.

The following table describes the Neo4j general log files and the information they contain.

Table 62. Neo4j logs for monitoring

Filename	Description
neo4j.log	The *user log*, where general information about Neo4j is written. Not written for Debian and RPM packages.
debug.log	The *debug log*, log information useful when debugging problems with Neo4j.
http.log	The *HTTP log*, log for the HTTP API.
gc.log	The *garbage collection log*, logging provided by the JVM.
query.log	The *query log*, log of executed queries that takes longer than a specified threshold. Enterprise
security.log	The *security log*, log of security events. Enterprise
service-error.log	The *windows service log*, log of errors encountered when installing or running the Windows service. Windows

Table 63. Log paths

Configuration setting	Default value	Description
`dbms.directories.logs`	`logs`	Path of the logs directory.
`dbms.logs.user.path`	`neo4j.log`	Path to the *user log* file.
`dbms.logs.debug.path`	`debug.log`	Path to the *debug log* file.
`dbms.logs.http.path`	`http.log`	Path to *HTTP log* file.
`dbms.logs.query.path`	`query.log`	Path to the *query log* file.
`dbms.logs.security.path`	`security.log`	Path to the *security log* file.

14.1.2. Log format

Neo4j 4.2 does not have any configuration for the log format.

14.1.3. User log

Table 64. User log configurations

The *user log* configuration	Default value	Description
`dbms.logs.user.rotation.delay`	`5m`	The minimum time interval after last rotation of the *user log*, before it may be rotated again.
`dbms.logs.user.rotation.keep_number`	`7`	The maximum number of history files for the *user log*.

The *user log* configuration	Default value	Description
`dbms.logs.user.rotation.size`	`0B`	The threshold size for rotation of the *user log*. If set to `0` log rotation is disabled.
`dbms.logs.user.stdout_enabled`	`true`	Send user logs to the process stdout. If this is disabled then logs will instead be sent to the *user log* (*neo4j.log*).

14.1.4. Debug log

Table 65. Debug log configurations

The *debug log* configuration	Default value	Description
`dbms.logs.debug.level`	`INFO`	Log level threshold for the *debug log*.
`dbms.logs.debug.rotation.delay`	`5m`	The minimum time interval after last rotation of the *debug log*, before it may be rotated again.
`dbms.logs.debug.rotation.keep_number`	`7`	The maximum number of history files for the *debug log*.
`dbms.logs.debug.rotation.size`	`20M`	The threshold size for rotation of the *debug log*.

The following table lists all message types raised by Neo4j and their severity level:

Table 66. Message types

Message type	Severity level	Description
`DEBUG`	Low severity	Report details on the raised errors and possible solutions.
`INFO`	Low severity	Report status information and errors that are not severe.
`WARN`	Low severity	Report errors that need attention but are not severe.
`ERROR`	High severity	Report errors that prevent the Neo4j server from running and must be addressed immediately.
`FATAL`	High severity	An event occurs that brings the application to a halt. No work continues.

To set the log level threshold for the *debug log* use the configuration setting `dbms.logs.debug.level`.

14.1.5. Garbage collection log

Table 67. Garbage collection log configurations

The *garbage collection log* configuration	Default value	Description
`dbms.logs.gc.enabled`	`false`	Enable garbage collection logging.
`dbms.logs.gc.options`		Garbage collection logging options.
`dbms.logs.gc.rotation.keep_number`	`0`	The maximum number of history files for the garbage collection log.
`dbms.logs.gc.rotation.size`		The threshold size for rotation of the garbage collection log.

14.1.6. HTTP log

Table 68. HTTP log configurations

The *HTTP log* configuration	Default value	Description
`dbms.logs.http.enabled`	`false`	Enable HTTP logging.
`dbms.logs.http.rotation.keep_number`	`5`	The maximum number of history files for the *HTTP log*.
`dbms.logs.http.rotation.size`	`20M`	The threshold size for rotation of the *HTTP log*.

14.1.7. Security log Enterprise edition

Neo4j provides security event logging that records all security events.

For native user management, the following actions are recorded:

- Login attempts - per default both successful and unsuccessful logins are recorded.
- All administration commands run towards the system database.
- All security procedures run towards the system database.

Security log configuration

Rotation of the security events log can be configured in the *neo4j.conf* configuration file.

The following configuration settings are available for the *security log*:

Table 69. Security log configurations

The *security log* configuration	Default value	Description
`dbms.logs.security.level`	`INFO`	Security log level threshold.
`dbms.logs.security.path`	`security.log`	The name of the *security log* file.

The *security log* configuration	Default value	Description
`dbms.logs.security.rotation.size`	`20M`	Sets the file size at which the security event log will auto-rotate.
`dbms.logs.security.rotation.delay`	`300s`	The minimum time interval after the last *security log* rotation occurred, before the *security log* may be rotated again.
`dbms.logs.security.rotation.keep_number`	`7`	The number of historical log files kept.

If using LDAP as the authentication method, some cases of LDAP misconfiguration will also be logged, as well as LDAP server communication events and failures.

If many programmatic interactions are expected, it is advised to disable the logging of successful logins. Logging of successful logins is disabled by setting the `dbms.security.log_successful_authentication` parameter in the `neo4j.conf` file:

```
dbms.security.log_successful_authentication=false
```

Example output for a security log:

```
2019-12-09 13:45:00.796+0000 INFO  [AsyncLog @ 2019-12-09 ...]  [johnsmith]: logged in
2019-12-09 13:47:53.443+0000 ERROR [AsyncLog @ 2019-12-09 ...]  [johndoe]: failed to log in: invalid
principal or credentials
2019-12-09 13:48:28.566+0000 INFO  [AsyncLog @ 2019-12-09 ...]  [johnsmith]: CREATE USER janedoe SET
PASSWORD '******' CHANGE REQUIRED
2019-12-09 13:48:32.753+0000 INFO  [AsyncLog @ 2019-12-09 ...]  [johnsmith]: CREATE ROLE custom
2019-12-09 13:49:11.880+0000 INFO  [AsyncLog @ 2019-12-09 ...]  [johnsmith]: GRANT ROLE custom TO janedoe
2019-12-09 13:49:34.979+0000 INFO  [AsyncLog @ 2019-12-09 ...]  [johnsmith]: GRANT TRAVERSE ON GRAPH *
NODES A, B (*) TO custom
2019-12-09 13:49:37.053+0000 INFO  [AsyncLog @ 2019-12-09 ...]  [johnsmith]: DROP USER janedoe
```

14.1.8. Query log Enterprise edition

Neo4j can be configured to log queries executed in the database.

Query logging is enabled by default and is controlled by the setting `dbms.logs.query.enabled`.

Configuration options are:

Table 70. Query log enabled setting

Option	Description
`OFF`	Will completely disable logging.

Option	Description
`INFO`	Will log at the end of queries that have either succeeded or failed. The `dbms.logs.query.threshold` parameter is used to determine the threshold for logging a query. If the execution of a query takes a longer time than this threshold, it will be logged. Setting the threshold to `0s` will result in all queries being logged.
`VERBOSE`	Will log all queries at both start and finish, regardless of `dbms.logs.query.threshold`. Default

Query log configuration

The name of the query log file is `query.log` by default, (see `dbms.logs.query.path`).

Rotation of the query log can be configured in the *neo4j.conf* configuration file.

The following configuration settings are available for the query log file:

Table 71. Query log configurations

The *query log* configuration	Default value	Description
`dbms.logs.query.allocation_logging_enabled`	`true`	Log allocated bytes for the executed queries being logged. The logged number is cumulative over the duration of the query, 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`.
`dbms.logs.query.early_raw_logging_enabled`	`false`	Log query text and parameters without obfuscating passwords. This allows queries to be logged earlier before parsing starts.
`dbms.logs.query.enabled`	`VERBOSE`	Log executed queries.
`dbms.logs.query.page_logging_enabled`	`false`	Log page hits and page faults for the executed queries being logged.

The *query log* configuration	Default value	Description
`dbms.logs.query.parameter_full_entities`	`false`	Log complete parameter entities including ID, labels or relationship type, and properties. If `false`, only the entity ID will be logged. This only takes effect if `dbms.logs.query.parameter_logging_enabled=true`.
`dbms.logs.query.parameter_logging_enabled`	`true`	Log parameters for the executed queries being logged.
`dbms.logs.query.rotation.keep_number`	`7`	The maximum number of history files for the *query log*.
`dbms.logs.query.rotation.size`	`20M`	The file size in bytes at which the *query log* will auto-rotate.
`dbms.logs.query.runtime_logging_enabled`	`true`	Logs which runtime that was used to run the query.
`dbms.logs.query.threshold`	`0s`	If the execution of query takes a longer time than this threshold, the query is logged once completed (provided query logging is set to `INFO`). A threshold of 0 seconds, will log all queries.
`dbms.logs.query.time_logging_enabled`	`false`	Log detailed time information for the executed queries being logged. Requires `dbms.track_query_cpu_time=true`.

Example 70. Configure for simple query logging

In this example we set query logging to `INFO`, but leave all other query log parameters at their defaults.

```
dbms.logs.query.enabled=INFO
```

Below is an example of the query log with this basic configuration:

```
2017-11-22 14:31 ... INFO  9 ms: bolt-session   bolt    johndoe neo4j-javascript/1.4.1
client/127.0.0.1:59167  ...
2017-11-22 14:31 ... INFO  0 ms: bolt-session   bolt    johndoe neo4j-javascript/1.4.1
client/127.0.0.1:59167  ...
2017-11-22 14:32 ... INFO  3 ms: server-session http    127.0.0.1   /db/data/cypher neo4j - CALL
dbms.procedures() - {}
2017-11-22 14:32 ... INFO  1 ms: server-session http    127.0.0.1   /db/data/cypher neo4j - CALL
dbms.showCurrentUs...
2017-11-22 14:32 ... INFO  0 ms: bolt-session   bolt    johndoe neo4j-javascript/1.4.1
client/127.0.0.1:59167  ...
2017-11-22 14:32 ... INFO  0 ms: bolt-session   bolt    johndoe neo4j-javascript/1.4.1
client/127.0.0.1:59167  ...
2017-11-22 14:32 ... INFO  2 ms: bolt-session   bolt    johndoe neo4j-javascript/1.4.1
client/127.0.0.1:59261  ...
```

Example 71. 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 transaction

You can attach metadata to a transaction and have it printed in the query log, using the built-in procedure `tx.setMetaData`.

Neo4j Drivers also support attaching metadata to a transaction. For more information, see the respective Driver's manual.

Every graph-app should follow a convention for passing metadata with the queries that it sends to Neo4j:

```
{
  app: "neo4j-browser_v4.2.2", ①
  type: "system" ②
}
```

① `app` could be a user-agent styled name plus version.

② `type` could be one of:

- `system` — a query automatically run by the app.
- `user-direct` — a query the user directly submitted to/through the app.
- `user-action` — a query resulting from an action the user performed.
- `user-transpiled` — a query that has been derived from the user input.

This is typically done programmatically but can also be used with the Neo4j dev tools.
In general, you start a transaction on a user database and attach a list of metadata to it by calling `tx.setMetaData`. You can also use the procedure `CALL tx.getMetaData()` to show the metadata of the current transaction. These examples use the MovieGraph dataset from the Neo4j Browser guide.

Example 72. Using `cypher-shell`, attach metadata to a transaction

```
neo4j@neo4j> :begin
neo4j@neo4j# CALL tx.setMetaData({app: 'neo4j-cypher-shell_v.4.2.2', type: 'user-direct', user:
'jsmith'});
0 rows
ready to start consuming query after 2 ms, results consumed after another 0 ms
neo4j@neo4j# CALL tx.getMetaData();
+--------------------------------------------------------------------------+
| metadata                                                                 |
+--------------------------------------------------------------------------+
| {app: "neo4j-cypher-shell_v.4.2.2", type: "user-direct", user: "jsmith"} |
+--------------------------------------------------------------------------+

1 row
ready to start consuming query after 37 ms, results consumed after another 2 ms
neo4j@neo4j# MATCH (n:Person) RETURN n  LIMIT 5;
+-----------------------------------------------------+
| n                                                   |
+-----------------------------------------------------+
| (:Person {name: "Keanu Reeves", born: 1964})        |
| (:Person {name: "Carrie-Anne Moss", born: 1967})    |
| (:Person {name: "Laurence Fishburne", born: 1961}) |
| (:Person {name: "Hugo Weaving", born: 1960})        |
| (:Person {name: "Lilly Wachowski", born: 1967})     |
+-----------------------------------------------------+

5 rows
ready to start consuming query after 2 ms, results consumed after another 1 ms
neo4j@neo4j# :commit
```

Example result in the query.log file

```
2021-07-30 14:43:17.176+0000 INFO  id:225 - 2 ms: 136 B - bolt-session  bolt    neo4j-cypher-
shell/v4.2.2       client/127.0.0.1:54026  server/127.0.0.1:7687>  neo4j - neo4j -
MATCH (n:Person) RETURN n  LIMIT 5; - {} - runtime=pipelined - {app: 'neo4j-cypher-shell_v.4.2.2',
type: 'user-direct', user: 'jsmith'}
```

Example 73. Using Neo4j Browser, attach metadata to a transaction

```
CALL tx.setMetaData({app: 'neo4j-browser_v.4.2.2', type: 'user-direct', user: 'jsmith'});
MATCH (n:Person) RETURN n LIMIT 5
```

Example result in the query.log file

```
2021-07-30 14:51:39.457+0000 INFO  Query started: id:328 - 0 ms: 0 B - bolt-session bolt    neo4j-
browser/v4.2.2        client/127.0.0.1:53666  server/127.0.0.1:7687>  neo4j - neo4j - MATCH
(n:Person) RETURN n  LIMIT 5 - {} - runtime=null - {type: 'system', app: 'neo4j-browser_v4.2.2'}
```

Example 74. Using Neo4j Bloom, attach metadata to a transaction

```
CALL tx.setMetaData({app: 'neo4j-browser_v.1.7.0', type: 'user-direct', user: 'jsmith'})
MATCH (n:Person) RETURN n LIMIT 5
```

Example result in the query.log file

```
2021-07-30 15:09:54.048+0000 INFO  id:95 - 1 ms: 72 B - bolt-session    bolt    neo4j-bloom/v1.7.0
client/127.0.0.1:54693  server/127.0.0.1:11003> neo4j - neo4j - RETURN TRUE - {} - runtime=pipelined
- {app: 'neo4j-bloom_v1.7.0', type: 'system'}
```

In Neo4j Browser and Bloom, the user-provided metadata is always replaced by the system metadata.

14.2. Query management

14.2.1. List all running queries

An administrator is able to view all queries that are currently executing within the instance. Alternatively, the current user may view all of their own currently-executing queries.

Syntax:

```
CALL dbms.listQueries()
```

Returns:

Name	Type	Description
`queryId`	String	This is the ID of the query.
`username`	String	This is the username of the user who is executing the query.
`metaData`	Map	This is any metadata associated with the transaction.
`query`	String	This is the query itself.

Name	Type	Description
`parameters`	Map	This is a map containing all the parameters used by the query.
`planner`	String	Planner used by the query. [11]
`runtime`	String	Runtime used by the query. [12]
`indexes`	List	Indexes used by the query.
`startTime`	String	This is the time at which the query was started.
`elapsedTime`	String	Deprecated: Use `elapsedTimeMillis` instead. This is the time that has elapsed since the query was started.
`connectionDetails`	String	Deprecated: Use `requestScheme`, `clientAddress`,`requestUri` These are the connection details pertaining to the query.
`protocol`	String	The protocol used by connection issuing the query.
`connectionId`	String	The ID of the connection issuing the query. This field will be null if the query was issued using embedded API.
`clientAddress`	String	The client address of the connection issuing the query.
`requestUri`	String	The request URI used by the client connection issuing the query.
`status`	String	Status of the executing query. Possible values: `Parsing`, `Planning`, `Planned`, `Running`, and `Waiting`.
`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 `dbms.track_query_cpu_time` is set to `true`.
`waitTimeMillis`	Integer	Wait time in milliseconds that has been spent waiting to acquire locks.
`idleTimeMillis`	Integer	Idle time in milliseconds. This field will be null unless the config parameter `dbms.track_query_cpu_time` is set to `true`.

Name	Type	Description
allocatedBytes	Integer	Estimated bytes allocated for the executing query. For memory-intense or long-running queries the value may be larger than the current memory usage. This field will be null unless the config parameter dbms.track_query_allocation is set to true.
pageHits	Integer	Page hits occurred during the execution.
pageFaults	Integer	Page faults occurred during the execution.
database	String	This is the name of the database the query is executing against.

Example 75. Viewing queries that are currently executing

The following example shows that the user alwood is currently running dbms.listQueries() yielding specific variables, namely queryId, username, query, elapsedTimeMillis, requestUri, status, and database.

```
CALL dbms.listQueries() YIELD queryId, username, query, elapsedTimeMillis, requestUri, status,
database
```

```
╒═══════════╤══════════╤═══════════════════════╤═══════════════╤═══════════╤═══════
╕
│"queryId"  │"username"│"query"                │"elapsedTimeMi
llis"│"requestUri"│"status"  │"database"│
╞═══════════╪══════════╪═══════════════════════╪═══════════════
╪═══════════╪═══════
╕
│"query-33" │"alwood"  │"CALL dbms.listQueries() YIELD queryId
, username, query, ela│1          │"127.0.0.1:7687"│"running" │"myDb"    │psedTime, requestUri, status,
database"             │           │           │
└───────────┴──────────┴───────────────────────┴───────────────
┴───────────┴───────
┘
1 row
```

14.2.2. List all active locks for a query

An administrator is able to view all active locks held by the transaction executing the query with the queryId.

Syntax:

```
CALL dbms.listActiveLocks(queryId)
```

Returns:

Name	Type	Description
mode	String	Lock mode corresponding to the transaction.
resourceType	String	Resource type of the locked resource
resourceId	Integer	Resource id of the locked resource .

Example 76. Viewing active locks for a query

The following example shows the active locks held by transaction executing query with id `query-614`

```
CALL dbms.listActiveLocks( "query-614" )
```

"mode"	"resourceType"	"resourceId"
"SHARED"	"SCHEMA"	0

1 row

The following example shows the active locks for all currently executing queries by yielding the `queryId` from `dbms.listQueries` procedure

```
CALL dbms.listQueries() YIELD queryId, query, database
CALL dbms.listActiveLocks( queryId ) YIELD resourceType, resourceId, mode
RETURN queryId, query, resourceType, resourceId, mode, database
```

"queryId"	"query"	"resourceType"	"resourceId"	"mode"	"database"
"query-614"	"match (n), (m), (o), (p), (q) return count(*)"	"SCHEMA"	0	"SHARED"	"myDb"
"query-684"	"CALL dbms.listQueries() YIELD .."	"SCHEMA"	0	"SHARED"	"myOtherDb"

2 rows

14.2.3. 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
ids	List<String>	This is a list of the IDs of all the queries 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 77. Terminating multiple queries

The following example shows that the administrator has terminated the queries with IDs `query-378` and `query-765`, started by the users `joesmith` and `annebrown`, respectively.

This command can target queries from multiple databases at the same time. In this example, `joesmith` ran his query against `joeDb` and `annebrown` ran hers against `anneDb`.

```
CALL dbms.killQueries(['query-378','query-765'])
```

```
+-----------------------------------------+
| queryId     | username    | message     |
+-----------------------------------------+
| "query-378" | "joesmith"  | "Query found" |
| "query-765" | "annebrown" | "Query found" |
+-----------------------------------------+
2 rows
```

14.2.4. 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 78. 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
```

14.3. Transaction management

14.3.1. 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 79. 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 (e.g. via the Java API or Neo4j Drivers), as a custom timeout will override the value set for `dbms.transaction.timeout`. Please note that the timeout value can only be overridden to a value that is smaller than that configured by `dbms.transaction.timeout`.

The *transaction timeout* feature is also known as the *transaction guard*.

14.3.2. 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 80. Configure lock acquisition timeout

Set the timeout to ten seconds.

```
dbms.lock.acquisition.timeout=10s
```

14.3.3. 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
`transactionId`	String	This is the ID of the transaction.
`username`	String	This is the username of the user who is executing the transaction.

Name	Type	Description
`metaData`	Map	This is any metadata associated with the transaction.
`startTime`	String	This is the time at which the transaction was started.
`protocol`	String	The protocol used by connection issuing the transaction.
`connectionId`	String	The ID of the connection issuing the transaction. This field will be null if the transaction was issued using embedded API.
`clientAddress`	String	The client address of the connection issuing the transaction.
`requestUri`	String	The request URI used by the client connection issuing the transaction.
`currentQueryId`	String	This is the ID of the current query executed by transaction.
`currentQuery`	String	This is the current query executed by transaction.
`activeLockCount`	Integer	Count of active locks held by transaction.
`status`	String	Status of the executing transaction. Possible values: `Running`, `Closing`, `Blocked by: <additional info>`, and `Terminated with reason: <additional info>`.
`resourceInformation`	Map	Information about what transaction is waiting for when it is blocked.
`elapsedTimeMillis`	Integer	This is the time in milliseconds that has elapsed since the transaction was started.
`cpuTimeMillis`	Integer	CPU time in milliseconds that has been actively spent executing the transaction.
`waitTimeMillis`	Integer	Wait time in milliseconds that has been spent waiting to acquire locks.
`idleTimeMillis`	Integer	Idle time in milliseconds.
`allocatedBytes`	Integer	Number of bytes allocated so far by the transaction. This column is deprecated in favor of `estimatedUsedHeapMemory`.
`allocatedDirectBytes`	Integer	Direct bytes used by the executing transaction.
`pageHits`	Integer	Page hits occurred during the execution.

Name	Type	Description
`pageFaults`	Integer	Page faults occurred during the execution.
`database`	String	This is the name of the database the transaction is executing against.
`estimatedUsedHeapMemory`	Integer	This is the current estimated heap usage of the transaction, in bytes.

Example 81. Viewing transactions that are currently executing

The following example shows that the user '**alwood**' is currently running `dbms.listTransactions()`. The procedure call yields specific information about the running transaction, namely `transactionId`, `username`, `currentQuery`, `elapsedTimeMillis`, `requestUri`, and `status`.

```
CALL dbms.listTransactions() YIELD transactionId, username, currentQuery, elapsedTimeMillis,
requestUri, status
```

```
"transactionId"        "username"  "currentQuery"                        "elapsedTimeMillis"
"requestUri"     "status"
"myDb-transaction-22" "alwood"    "CALL dbms.listTransactions() YIELD   "1"
"127.0.0.1:7687" "Running"          transactionId, username, currentQuery
                                   elapsedTime, requestUri, status"

1 row
```

14.4. Connection management

14.4.1. 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
`connectionId`	String	This is the ID of the network connection.
`connectTime`	String	This is the time at which the connection was started.
`connector`	String	Name of the connector that accepted the connection.
`username`	String	This is the username of the user who initiated the connection. This field will be null if the transaction was issued using embedded API. It can also be null if connection did not perform authentication.
`userAgent`	String	Name of the software that is connected. 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.
`serverAddress`	String	The server address this connection is connected to.
`clientAddress`	String	The client address of the connection.

Example 82. List all network connections

The following example shows that the user '**alwood**' is connected using Java driver and a Firefox web browser. The procedure call yields specific information about the connection, namely `connectionId`, `connectTime`, `connector`, `username`, `userAgent`, and `clientAddress`.

```
CALL dbms.listConnections() YIELD connectionId, connectTime, connector, username, userAgent,
clientAddress
```

```
│"connectionId"│"connectTime"             │"connector"│"username"│"userAgent"
"clientAddress"│"status" │
│"bolt-21"     │"2018-10-10T12:11:42.276Z"│"bolt"     │"alwood"  │"neo4j-java/1.6.3"
"127.0.0.1:53929"│"Running"│
│"http-11"     │"2018-10-10T12:37:19.014Z"│"http"     │null      │"Mozilla/5.0 (Macintosh; Intel
macOS 10.13; rv:62.0) Gecko/20100101 Firefox/62.0"│"127.0.0.1:54118"│"Running"│

2 rows
```

14.4.2. 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 83. 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

14.4.3. 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 84. 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

14.5. Background job management

There are many types of background jobs performed in the DBMS, many of which are triggered as system jobs by the DBMS itself without any user action. For example, important background jobs include checkpoint or index population. The former is triggered by the DBMS, and the latter can be a result of a user creating or modifying an index definition.

Background jobs are of the following types:

- `IMMEDIATE` - a one-time action, triggered and run in the background.
- `DELAYED` - a one-time action, run in the background at a given point in the future.
- `PERIODIC` - a recurring action, run in the background at a given time interval.

The DBMS provides a way to show active and failed background jobs. Active jobs are those that are currently running, or are scheduled to be delayed or periodic jobs. If a background job fails, or fails to start, the details of the failure are stored in the failed jobs list. Please note that only the last 100 jobs are stored in the failed jobs list, and that this list is not persistent, so it is cleared with a DBMS restart.

Additionally, it should be noted that a single periodic job can contribute multiple times to the failed jobs list.

14.5.1. Listing active background jobs

An administrator can list background jobs active on an instance:

Syntax:

```
CALL dbms.scheduler.jobs()
```

Returns:

Name	Type
`jobId` ID of the job. Can be used to keep track of an active job, and to link a job to a failed job run.	String
`group` A job is a member of a job group. For example, `INDEX_POPULATION`, `LOG_ROTATION` or `RAFT_SERVER`.	String
`submitted` A timestamp for when the job was submitted, in ISO-8601 format.	String
`database` Jobs can have either a database or a DBMS scope: • For database, this column will display the name of the database. • For DBMS, this column will be blank.	String
`submitter` Jobs are either triggered as a result of user action, or as a system job by the DBMS itself. This column will contain a username for jobs triggered by users, or is otherwise blank.	String
`description` A short description of a job that, unlike `currentStateDescription`, does not change during the running of the job.	String
`type` Type of the job. The values can be `IMMEDIATE`, `DELAYED` or `PERIODIC`.	String

Name	Type
`scheduledAt` A timestamp for when a `DELAYED` or `PERIODIC` job will be run, in ISO-8601 format. This column is not applicable to `IMMEDIATE` jobs, and will be blank for that job type.	String
`period` A period of a `PERIODIC` job, in format `hh:mm:ss.sss`.	String
`state` A state of the job. Since this procedure lists only active jobs, they can be either in `SCHEDULED` or `EXECUTING` state. `SCHEDULED` state is applicable only to `DELAYED` or `PERIODIC` jobs, and means that the job is scheduled for a given time in the future.	String
`currentStateDescription` If a job supports reposting its progress, the progress will be reported in this column in a free-form format, specific for each job.	String

14.5.2. Listing failed job executions

An administrator can list job executions failed on an instance:

Syntax:

```
CALL dbms.scheduler.failedJobs()
```

Returns:

Name	Type
`jobId` ID of the failed job.	String
`group` A job is a member of a job group. For example, `INDEX_POPULATION`, `LOG_ROTATION` or `RAFT_SERVER`.	String

Name	Type
`database` Jobs can have either a database or a DBMS scope: • For database, this column will display the name of the database. • For DBMS, this column will be blank.	String
`submitter` Jobs are either triggered as a result of user action, or as a system job by the DBMS itself. This column will contain a username for jobs triggered by users, or is otherwise blank.	String
`description` A short description of a job that, unlike `currentStateDescription`, does not change during the running of the job.	String
`type` Type of the job. The values can be `IMMEDIATE`, `DELAYED` or `PERIODIC`.	String
`submitted` A timestamp for when the job was submitted, in ISO-8601 format.	String
`executionStart` A timestamp for when the failed execution started, in ISO-8601 format.	String
`failureTime` A timestamp for when the execution failed, in ISO-8601 format.	String
`failureDescription` A short description of the failure. If the failure description is insufficient, more information can be found in logs.	String

14.6. Monitoring a Causal Cluster

In addition to specific metrics as described in previous sections, Neo4j Causal Clusters provide an infrastructure that operators will wish to monitor. The procedures can be used to inspect the cluster state and to understand its current condition and topology. Additionally, there are HTTP endpoints for checking health and status.

This section describes the following:

14.6.1. Procedures for monitoring a Causal Cluster

A number of procedures are available that provide information about a cluster.

Find out the role of a cluster member

The procedure `dbms.cluster.role(databaseName)` can be called on every instance in a Causal Cluster to return the role of the instance. Each instance holds multiple databases and participates in multiple independent Raft groups. The role returned by the procedure is for the database denoted by the `databaseName` parameter.

Syntax:

```
CALL dbms.cluster.role(databaseName)
```

Arguments:

Name	Type	Description
`databaseName`	String	The name of the database to get the cluster role for.

Returns:

Name	Type	Description
`role`	String	This is the role of the current instance, which can be `LEADER`, `FOLLOWER`, or `READ_REPLICA`.

Considerations:

- While this procedure is useful in and of itself, it serves as basis for more powerful monitoring procedures.

Example 85. Check the role of this instance

The following example shows how to find out the role of the current instance for database `neo4j`, which in this case is `FOLLOWER`.

```
CALL dbms.cluster.role("neo4j")
```

role
FOLLOWER

Gain an overview over the instances in the cluster

The procedure `dbms.cluster.overview()` provides an overview of cluster topology by returning details on all the instances in the cluster.

Syntax:

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

Returns:

Name	Type	Description
`id`	String	This is id of the instance.
`addresses`	List	This is a list of all the addresses for the instance.
`groups`	List	This is a list of all the server groups which an instance is part of.
`databases`	Map	This is a map of all databases with corresponding roles which the instance is hosting. The keys in the map are database names. The values are roles of this instance in the corresponding Raft groups, which can be `LEADER`, `FOLLOWER`, or `READ_REPLICA`.

Example 86. Get an overview of the cluster

The following example shows how to explore the cluster topology.

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

id	addresses	groups	databases
08eb9305-53b9-4394-9237-0f0d63bb05d5	[bolt://neo20:7687, http://neo20:7474, https://neo20:7473]	[]	{system: LEADER, neo4j: FOLLOWER}
cb0c729d-233c-452f-8f06-f2553e08f149	[bolt://neo21:7687, http://neo21:7474, https://neo21:7473]	[]	{system: FOLLOWER, neo4j: FOLLOWER}
ded9eed2-dd3a-4574-bc08-6a569f91ec5c	[bolt://neo22:7687, http://neo22:7474, https://neo22:7473]	[]	{system: FOLLOWER, neo4j: LEADER}
00000000-0000-0000-0000-000000000000	[bolt://neo34:7687, http://neo34:7474, https://neo34:7473]	[]	{system: READ_REPLICA, neo4j: READ_REPLICA}
00000000-0000-0000-0000-000000000000	[bolt://neo28:7687, http://neo28:7474, https://neo28:7473]	[]	{system: READ_REPLICA, neo4j: READ_REPLICA}
00000000-0000-0000-0000-000000000000	[bolt://neo31:7687, http://neo31:7474, https://neo31:7473]	[]	{system: READ_REPLICA, neo4j: READ_REPLICA}

Get routing recommendations

From the application point of view it is not interesting to know about the role a member plays in the cluster. Instead, the application needs to know which instance can provide the wanted service. The procedure `dbms.routing.getRoutingTable(routingContext, databaseName)` provides this information.

Syntax:

```
CALL dbms.routing.getRoutingTable(routingContext, databaseName)
```

Arguments:

Name	Type	Description
`routingContext`	Map	The routing context used for multi-data center deployments. It should be used in combination with multi-data center load balancing.
`databaseName`	String	The name of the database to get the routing table for.

Example 87. Get routing recommendations

The following example shows how discover which instances in the cluster can provide which services for database `neo4j`.

```
CALL dbms.routing.getRoutingTable({}, "neo4j")
```

The procedure returns a map between a particular service, `READ`, `WRITE` and `ROUTE`, and the addresses of instances that provide this service. It also returns a Time To Live (TTL) in seconds as a suggestion on how long the client could cache the response.

The result is not primarily intended for human consumption. Expanding it this is what it looks like.

```
{
    "ttl": 300,
    "servers": [
        {
            "addresses": ["neo20:7687"],
            "role": "WRITE"
        },
        {
            "addresses": ["neo21:7687", "neo22:7687", "neo34:7687", "neo28:7687", "neo31:7687"],
            "role": "READ"
        },
        {
            "addresses": ["neo20:7687", "neo21:7687", "neo22:7687"],
            "role": "ROUTE"
        }
    ]
}
```

14.6.2. Endpoints for status information

A Causal Cluster exposes some HTTP endpoints which can be used to monitor the health of the cluster. In this section we will describe these endpoints and explain their semantics.

Adjusting security settings for Causal Clustering endpoints

If authentication and authorization is enabled in Neo4j, the Causal Clustering status endpoints will also require authentication credentials. The setting `dbms.security.auth_enabled` controls whether the native auth provider is enabled. For some load balancers and proxy servers, providing authentication credentials with the request is not an option. For those situations, consider disabling authentication of the Causal Clustering status endpoints by setting `dbms.security.causal_clustering_status_auth_enabled=false` in *neo4j.conf*.

Unified endpoints

A unified set of endpoints exist, both on Core Servers and on Read Replicas, with the following behavior:

- `/db/<databasename>/cluster/writable` — Used to direct `write` traffic to specific instances.
- `/db/<databasename>/cluster/read-only` — Used to direct `read` traffic to specific instances.
- `/db/<databasename>/cluster/available` — Available for the general case of directing arbitrary request

types to instances that are available for processing read transactions.

- `/db/<databasename>/cluster/status` — Gives a detailed description of this instance's view of its status within the cluster, for the given database.
- `/dbms/cluster/status` — Gives a detailed description of this instance's view of its status within the cluster, for all databases. See Status endpoints for further details.

Every `/db/<databasename>/*` endpoint targets a specific database. The `databaseName` path parameter represents the name of the database. By default, a fresh Neo4j installation with two databases `system` and `neo4j` will have the following cluster endpoints:

```
http://localhost:7474/dbms/cluster/status

http://localhost:7474/db/system/cluster/writable
http://localhost:7474/db/system/cluster/read-only
http://localhost:7474/db/system/cluster/available
http://localhost:7474/db/system/cluster/status

http://localhost:7474/db/neo4j/cluster/writable
http://localhost:7474/db/neo4j/cluster/read-only
http://localhost:7474/db/neo4j/cluster/available
http://localhost:7474/db/neo4j/cluster/status
```

Table 72. Unified HTTP endpoint responses

Endpoint	Instance state	Returned code	Body text
`/db/<databasename>/cluster/writable`	Leader	`200 OK`	`true`
	Follower	`404 Not Found`	`false`
	Read Replica	`404 Not Found`	`false`
`/db/<databasename>/cluster/read-only`	Leader	`404 Not Found`	`false`
	Follower	`200 OK`	`true`
	Read Replica	`200 OK`	`true`
`/db/<databasename>/cluster/available`	Leader	`200 OK`	`true`
	Follower	`200 OK`	`true`
	Read Replica	`200 OK`	`true`

Endpoint	Instance state	Returned code	Body text
/db/<databasename>/cluster/status	Leader	200 OK	JSON - See Status endpoint for details.
	Follower	200 OK	JSON - See Status endpoint for details.
	Read Replica	200 OK	JSON - See Status endpoint for details.
/dbms/cluster/status	Leader	200 OK	JSON - See Status endpoint for details.
	Follower	200 OK	JSON - See Status endpoint for details.
	Read Replica	200 OK	JSON - See Status endpoint for details.

Example 88. Use a Causal Clustering monitoring endpoint

From the command line, a common way to ask those endpoints is to use `curl`. With no arguments, `curl` will do an HTTP `GET` on the URI provided and will output the body text, if any. If you also want to get the response code, just add the `-v` flag for verbose output. Here are some examples:

- Requesting `writable` endpoint on a Core Server that is currently elected leader with verbose output:

```
#> curl -v localhost:7474/db/neo4j/cluster/writable
* About to connect() to localhost port 7474 (#0)
*   Trying ::1...
* connected
* Connected to localhost (::1) port 7474 (#0)
> GET /db/neo4j/cluster/writable HTTP/1.1
> User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
> Host: localhost:7474
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/plain
< Access-Control-Allow-Origin: *
< Transfer-Encoding: chunked
< Server: Jetty(9.4.17)
<
* Connection #0 to host localhost left intact
true* Closing connection #0
```

Status endpoints

The status endpoint, available at `/db/<databasename>/cluster/status`, is to be used to assist with rolling upgrades. For more information, see Upgrade and Migration Guide → Upgrade a Causal Cluster.

Typically, you will want to have some guarantee that a Core is safe to shutdown for each database before removing it from a cluster. Counter intuitively, a core being safe to shutdown means that a majority of the **other** cores are healthy, caught up, and have recently heard from that database's leader. The status endpoints provide the following information in order to help resolve such issues.

Several of the fields in status endpoint responses refer to details of Raft, the algorithm used in Neo4j Causal Clusters to provide highly available transactions. When using multiple databases, each database implements Raft independently. Therefore, details such as `leader` and `raftCommandsPerSecond` are database specific.

Example 89. Example status response

```
{
  "lastAppliedRaftIndex":0,
  "votingMembers":["30edc1c4-519c-4030-8348-7cb7af44f591","80a7fb7b-c966-4ee7-88a9-35db8b4d68fe"
,"f9301218-1fd4-4938-b9bb-a03453e1f779"],
  "memberId":"80a7fb7b-c966-4ee7-88a9-35db8b4d68fe",
  "leader":"30edc1c4-519c-4030-8348-7cb7af44f591",
  "millisSinceLastLeaderMessage":84545,
  "participatingInRaftGroup":true,
  "core":true,
  "isHealthy":true,
  "raftCommandsPerSecond":124
}
```

Table 73. Status endpoint descriptions

Field	Type	Optional	Example	Description
`core`	boolean	no	`true`	Used to distinguish between Core Servers and Read Replicas.
`lastAppliedRaftIndex`	number	no	`4321`	Every transaction in a cluster is associated with a raft index. Gives an indication of what the latest applied raft log index is.
`participatingInRaftGroup`	boolean	no	`false`	A participating member is able to vote. A Core is considered participating when it is part of the voter membership and has kept track of the leader.
`votingMembers`	string[]	no	`[]`	A member is considered a voting member when the leader has been receiving communication with it. List of member's `memberId` that are considered part of the voting set by this Core.

Field	Type	Optional	Example	Description
`isHealthy`	boolean	no	`true`	Reflects that the local database of this member has not encountered a critical error preventing it from writing locally.
`memberId`	string	no	`30edc1c4-519c-4030-8348-7cb7af44f591`	Every member in a cluster has it's own unique member id to identify it. Use `memberId` to distinguish between Core and Read Replica.
`leader`	string	yes	`80a7fb7b-c966-4ee7-88a9-35db8b4d68fe`	Follows the same format as `memberId`, but if it is null or missing, then the leader is unknown.
`millisSinceLastLeaderMessage`	number	yes	`1234`	The number of milliseconds since the last heartbeat-like leader message. Not relevant to Read Replicas, and hence is not included.
`raftCommandsPerSecond`	number	yes	`124`	An estimate of the average Raft state machine throughput over a sampling windown configurable via `causal_clustering.status_throughput_window` setting.

After an instance has been switched on, you can access the status endpoint in order to make sure all the guarantees listed in the table below are met.

To get the most accurate view of a cluster it is strongly recommended to access the *status endpoint* on all core members and compare the result. The following table explains how results can be compared.

Table 74. Measured values, accessed via the status endpoint

Name of check	Method of calculation	Description
`allServersAreHealthy`	Every Core's status endpoint indicates `isHealthy==true`.	We want to make sure the data across the entire cluster is healthy. Whenever any Cores are false that indicates a larger problem.
`allVotingSetsAreEqual`	For any 2 Cores (A and B), status endpoint A's `votingMembers`== status endpoint B's `votingMembers`.	When the voting begins, all the Cores are equal to each other, and you know all members agree on membership.
`allVotingSetsContainAtLeastTargetCluster`	For all Cores (S), excluding Core Z (to be switched off), every member in S contains S in their voting set. Membership is determined by using the `memberId` and `votingMembers` from the status endpoint.	Sometimes network conditions will not be perfect and it may make sense to switch off a different Core to the one we originally wanted to switch off. If you run this check for all Cores, the ones that match this condition can be switched off (providing other conditions are also met).
`hasOneLeader`	For any 2 Cores (A and B), `A.leader == B.leader && leader!=null`.	If the leader is different then there may be a partition (alternatively, this could also occur due to bad timing). If the leader is unknown, that means the leader messages have actually timed out.

Name of check	Method of calculation	Description
noMembersLagging	For Core A with lastAppliedRaftIndex = min, and Core B with lastAppliedRaftIndex = max, B.lastAppliedRaftIndex-A.lastAppliedRaftIndex<raftIndexLagThreshold.	If there is a large difference in the applied indexes between Cores, then it could be dangerous to switch off a Core.

Combined status endpoints

When using the status endpoints to support a rolling upgrade, you need to assess whether a Core is safe to shutdown for **all** databases. To avoid having to issue a separate request to each `/db/<databasename>/cluster/status` endpoint, you can use the `/dbms/cluster/status` instead.

This endpoint returns a json array, the elements of which contain the same fields as the single database version, along with fields for for `databaseName` and `databaseUuid`.

Example 90. Example combined status response


```
[
  {
    "databaseName": "neo4j",
    "databaseUuid": "f4dacc01-f88a-4512-b3bf-68f7539c941e",
    "databaseStatus": {
      "lastAppliedRaftIndex": -1,
      "votingMembers": [
        "0cff51ad-7cee-44cc-9102-538fc4544b95",
        "90ff5df1-f5f8-4b4c-8289-a0e3deb2235c",
        "99ca7cd0-6072-4387-bd41-7566a98c6afc"
      ],
      "memberId": "90ff5df1-f5f8-4b4c-8289-a0e3deb2235c",
      "leader": "90ff5df1-f5f8-4b4c-8289-a0e3deb2235c",
      "millisSinceLastLeaderMessage": 0,
      "raftCommandsPerSecond": 0.0,
      "core": true,
      "participatingInRaftGroup": true,
      "healthy": true
    }
  },
  {
    "databaseName": "system",
    "databaseUuid": "00000000-0000-0000-0000-000000000001",
    "databaseStatus": {
      "lastAppliedRaftIndex": 7,
      "votingMembers": [
        "0cff51ad-7cee-44cc-9102-538fc4544b95",
        "90ff5df1-f5f8-4b4c-8289-a0e3deb2235c",
        "99ca7cd0-6072-4387-bd41-7566a98c6afc"
      ],
      "memberId": "90ff5df1-f5f8-4b4c-8289-a0e3deb2235c",
      "leader": "90ff5df1-f5f8-4b4c-8289-a0e3deb2235c",
      "millisSinceLastLeaderMessage": 0,
      "raftCommandsPerSecond": 0.0,
      "core": true,
      "participatingInRaftGroup": true,
      "healthy": true
    }
  }
]
```

14.7. Monitoring individual database states

In addition to the system-wide metrics and logs described in previous sections, operators may wish to monitor the state of individual databases being hosted within a Neo4j instance. The `SHOW DATABASES` command may be used for this purpose.

14.7.1. Listing Databases

First ensure that you are executing queries against the `system` database, either by running the command `:use system` (if using the Cypher shell or Neo4j Browser) or by creating a session against the `system` database using a Neo4j driver. Subsequently, run the `SHOW DATABASES` command.

Syntax:

```
SHOW DATABASES
```

Returns:

Name	Type	Description
`name`	String	The human-readable name of the database.
`address`	String	The bolt address of the Neo4j instance hosting the database.
`role`	String	The cluster role which the Neo4j instance fulfils for this database.
`requestedStatus`	String	The state that an operator has requested the database to be in.
`currentStatus`	String	The state the database is actually in on this Neo4j instance.
`error`	String	Error encountered by the Neo4j instance when transitioning the database to `requestedStatus`, if any.
`default`	String	Whether this database is the default for this DBMS.

Example 91. Listing databases in standalone Neo4j

When executing `SHOW DATABASES` against a standalone instance of Neo4j, you should see output like the following:

name	address	role	requestedStatus	currentStatus	error	default
"neo4j"	"localhost:7687"	"standalone"	"online"	"online"	""	true
"system"	"localhost:7687"	"standalone"	"online"	"online"	""	false

Note that the `role` and `address` columns are primarily intended to distinguish between the states of a given database, across multiple Neo4j instances deployed in a Causal Cluster. In a standalone deployment where you have a single Neo4j instance, your `address` field should be the same for every database, and your `role` field should always be "standalone".

If an error occurred whilst creating (or stopping, dropping etc.) a database, you should see output like the following:

name	address	role	requestedStatus	currentStatus	error	default
"neo4j"	"localhost:7687"	"standalone"	"online"	"online"	""	true
"system"	"localhost:7687"	"standalone"	"online"	"online"	""	false
"foo"	"localhost:7687"	"standalone"	"online"	"offline"	"An error occurred! Unable to start database ..."	false

Note that for failed databases, the `currentStatus` and `requestedStatus` are different. This can imply an error. For example:

- a database may take a while to transition from "offline" to "online", due to performing recovery.
- during normal operation, the `currentStatus` of a database may be transiently different from its `requestedStatus`, due to a necessary automatic process, such as one Neo4j instance copying store files from another.

The possible statuses are "initial", "online", "offline", "store copying", and "unknown".

Example 92. Listing databases in a Neo4j Causal Cluster

When running `SHOW DATABASES` against a Neo4j Causal Cluster you might see output like the following:

name	address	role	requestedStatus	currentStatus	error	default
"neo4j"	"localhost:20031"	"follower"	"online"	"online"	""	true
"neo4j"	"localhost:20010"	"follower"	"online"	"online"	""	true
"neo4j"	"localhost:20005"	"leader"	"online"	"online"	""	true
"neo4j"	"localhost:20034"	"read_replica"	"online"	"online"	""	true
"system"	"localhost:20031"	"follower"	"online"	"online"	""	false
"system"	"localhost:20010"	"follower"	"online"	"online"	""	false
"system"	"localhost:20005"	"leader"	"online"	"online"	""	false
"system"	"localhost:20034"	"read_replica"	"online"	"online"	""	false
"foo"	"localhost:20031"	"leader"	"online"	"online"	""	false
"foo"	"localhost:20010"	"follower"	"online"	"online"	""	false
"foo"	"localhost:20005"	"follower"	"online"	"online"	""	false
"foo"	"localhost:20034"	"read_replica"	"online"	"online"	""	false

Note that `SHOW DATABASES` does **not** return 1 row per database. Instead, it returns 1 row per database, per Neo4j instance in the cluster. Therefore, if you have a 4-instance cluster, hosting 3 databases, you will have 12 rows.

If an error occurred whilst creating (or stopping, dropping etc.) a database, you should see output like the following:

name	address	role	requestedStatus	currentStatus	error	default
"neo4j"	"localhost:20031"	"follower"	"online"	"online"	""	true
"neo4j"	"localhost:20010"	"follower"	"online"	"online"	""	true

name	address	role	requestedStatus	currentStatus	error	default
"neo4j"	"localhost:20005"	"leader"	"online"	"online"	""	true
"neo4j"	"localhost:20034"	"read_replica"	"online"	"online"	""	true
"system"	"localhost:20031"	"follower"	"online"	"online"	""	false
"system"	"localhost:20010"	"follower"	"online"	"online"	""	false
"system"	"localhost:20005"	"leader"	"online"	"online"	""	false
"system"	"localhost:20034"	"read_replica"	"online"	"online"	""	false
"foo"	"localhost:20031"	"unknown"	"online"	"initial"	"An error occurred! Unable to start database ..."	false
"foo"	"localhost:20010"	"leader"	"online"	"online"	""	false
"foo"	"localhost:20005"	"follower"	"online"	"online"	""	false
"foo"	"localhost:20034"	"unknown"	"online"	"initial"	"An error occurred! Unable to start database ..."	false

Note that different instances may have different roles for each database.

If a database is offline on a particular Neo4j instance, either because it was stopped by an operator or an error has occurred, its cluster `role` is "unknown". This is because the cluster role of a given instance/database combination cannot be assumed in advance. This differs from standalone Neo4j instances, where the role of that instance for each database can always be assumed to be "standalone".

The possible roles are "standalone", "leader", "follower", "read_replica", and "unknown".

14.7.2. Listing a single database

The number of rows returned by `SHOW DATABASES` can be quite large, especially when run in a cluster. You can filter the rows returned by database name (e.g. "foo") by using the command `SHOW DATABASE foo`.

Syntax:

```
SHOW DATABASE databaseName
```

Arguments:

Name	Type	Description
`databaseName`	String	The name of the database whose status to report

Returns:

Name	Type	Description
`name`	String	The human-readable name of the database.
`address`	String	The bolt address of the Neo4j instance hosting the database.
`role`	String	The cluster role which the Neo4j instance fulfils for this database.
`requestedStatus`	String	The state that an operator has requested the database to be in.
`currentStatus`	String	The state the database is actually in on this Neo4j instance.
`error`	String	Error encountered by Neo4j instance when transitioning the database to `requestedStatus`, if any.
`default`	String	Whether this database is the default for this DBMS.

Example 93. Listing statuses for database foo

When running `SHOW DATABASE foo` in a Neo4j Causal Cluster, you should see output like the following:

name	address	role	requestedStatus	currentStatus	error	default
"foo"	"localhost:20031"	"unknown"	"online"	"initial"	"An error occurred! Unable to start database ..."	false
"foo"	"localhost:20010"	"leader"	"online"	"online"	""	false
"foo"	"localhost:20005"	"follower"	"online"	"online"	""	false
"foo"	"localhost:20034"	"unknown"	"online"	"initial"	"An error occurred! Unable to start database ..."	false

[11] For details, see Cypher Manual → Cypher planner
[12] For details, see Cypher Manual → Cypher runtime

Chapter 15. Performance

The topics described in this chapter are:

- Memory configuration — How to configure memory settings for efficient operations.
- Index configuration — How to configure indexes.
- Garbage collector — How to configure the Java Virtual Machine's garbage collector.
- Bolt thread pool configuration — How to configure the Bolt thread pool.
- Linux file system tuning — How to configure the Linux file system.
- Disks, RAM and other tips — Disks, RAM and other tips.
- Statistics and execution plans — How schema statistics and execution plans affect Cypher query performance.
- Space reuse — Data deletion and storage space reuse.

15.1. Memory configuration

15.1.1. Overview

The RAM of the Neo4j server has a number of usage areas, with some sub-areas:

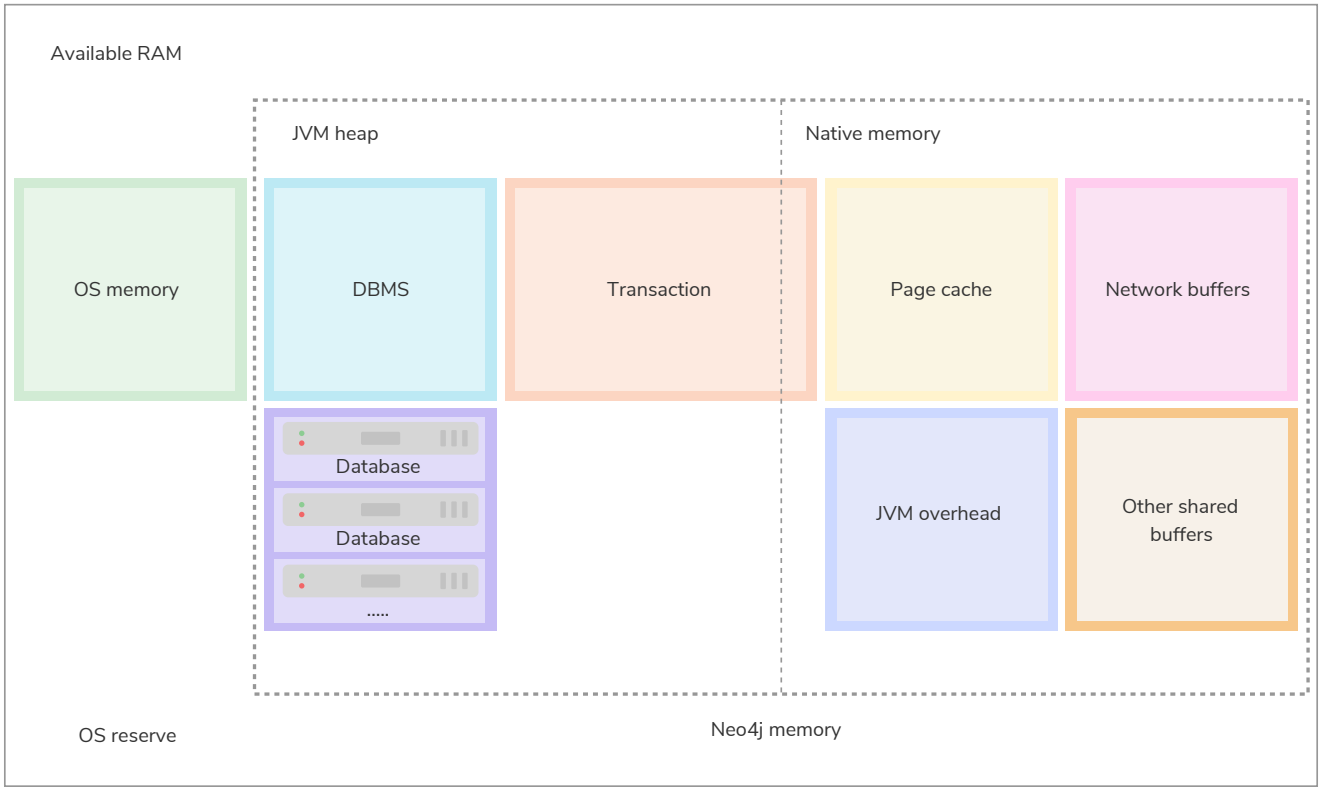


Figure 13. Neo4j memory management

OS memory

Some memory must be reserved for running the processes of the operating system itself. It is not possible to explicitly configure the amount of RAM that should be reserved for the operating system, as

this is what RAM remains available after configuring Neo4j. If you do not leave enough space for the OS, it will start to swap memory to disk, which will heavily affect performance.

1GB is a good starting point for a server that is dedicated to running Neo4j. However, there are cases where the amount reserved for the OS is significantly larger than 1GB, such as servers with exceptionally large RAM.

JVM Heap

The JVM heap is a separate dynamic memory allocation that Neoj4 uses to store instantiated Java objects. The memory for the Java objects are managed automatically by a garbage collector. Particularly important is that a garbage collector automatically handles the deletion of unused objects. For more information on how the garbage collector works and how to tune it, see Tuning of the garbage collector.

The heap memory size is determined by the parameters `dbms.memory.heap.initial_size` and `dbms.memory.heap.max_size`. It is recommended to set these two parameters to the same value to avoid unwanted full garbage collection pauses.

Generally, to aid performance, you should configure a large enough heap to sustain concurrent operations.

Native memory

Native memory, sometimes referred to as off-heap memory, is memory directly allocated by Neo4j from the OS. This memory will grow dynamically as needed and is not subject to the garbage collector.

DBMS

The database management system, or DBMS, contains the global components of the Neo4j instance. For example, the bolt server, logging service, monitoring service, etc.

Database

Each database in the system comes with an overhead. In deployments with multiple databases, this overhead needs to be accounted for.

Transaction

When executing a transaction, Neo4j holds not yet committed data, the result, and intermediate states of the queries in memory. The size needed for this is very dependent on the nature of the usage of Neo4j. For example, long-running queries, or very complicated queries, are likely to require more memory. Some parts of the transactions can optionally be placed off-heap, but for the best performance, it is recommended to keep the default with everything on-heap.

This memory group can be limited with the setting `dbms.memory.transaction.global_max_size`.

Page cache

The page cache is used to cache the Neo4j data stored on disk. The caching of graph data and indexes into memory helps avoid costly disk access and result in optimal performance.

The parameter for specifying how much memory Neo4j is allowed to use for the page cache is: `dbms.memory.pagecache.size`.

Network buffers

Direct buffers are used by Neo4j to send and receive data. Direct byte buffers are important for improving performance because they allow native code and Java code to share data without copying it. However, they are expensive to create, which means byte buffers are usually reused once they are created.

Other shared buffers

This includes unspecified shared direct buffers.

JVM overhead

The JVM will require some memory to function correctly. For example, this can be:

- **Thread stacks** – Each thread has its own call stack. The stack stores primitive local variables and object references along with the call stack (list of method invocations) itself. The stack is cleaned up as stack frames move out of context, so there is no GC performed here.
- **Metaspace** – Metaspace stores the java class definitions and some other metadata.
- **Code cache** – The JIT compiler stores the native code it generates in the code cache to improve performance by reusing it.

For more details and means of limiting the memory used by the JVM please consult your JVM documentation.

15.1.2. Considerations

Always use explicit configuration

To have good control of the system behavior, it is recommended to always define the page cache and heap size parameters explicitly in neo4j.conf. Otherwise, Neo4j computes some heuristic values at startup based on the available system resources.

Initial memory recommendation

Use the `neo4j-admin memrec` command to get an initial recommendation for how to distribute a certain amount of memory. The values may need to be adjusted to cater for each specific use case.

Inspect the memory settings of all databases in a DBMS

The `neo4j-admin memrec` command is useful for inspecting the current distribution of data and indexes.

Example 94. Use `neo4j-admin memrec` to inspect the memory settings of all your databases

Estimate the total size of the database files.

```
$neo4j-home> bin/neo4j-admin memrec
...
...
...
# Total size of lucene indexes in all databases: 6690m
# Total size of data and native indexes in all databases: 17050m
```

You can see that the Lucene indexes take up approximately 6.7GB of data, and that the data volume and native indexes combined take up approximately 17GB.

Using this information, you can do a sanity check of your memory configuration:

- Compare the value for data volume and native indexes to the value of `dbms.memory.pagecache.size`.
- For cases when *off-heap* transaction state is used, estimate transactional workload and how much memory is left to the value of `dbms.tx_state.max_off_heap_memory`.
- Compare the value for Lucene indexes to how much memory is left after assigning `dbms.memory.pagecache.size` and `dbms.memory.heap.initial_size`.

In some production systems the access to memory is limited and must be negotiated between different areas. Therefore, it is recommended to perform a certain amount of testing and tuning of these settings to figure out the optimal division of the available memory.

Limit transaction memory usage recommendation

The measured heap usage of all transactions is only an estimate and the actual heap utilization may be slightly larger or slightly smaller than the estimated value. In some cases, limitations of the estimation algorithm to detect shared objects at a deeper level of the memory graph could lead to overestimations. This is because a conservative estimate is given based on aggregated estimations of memory usage, where the identities of all contributing objects are not known, and cannot be assumed to be shared. For example, when you use `UNWIND` on a very large list, or expand a variable length or shortest path pattern, where many relationships are shared between the computed result paths.

In these cases, if you experience problems with a query that gets terminated, you can execute the same query with the transaction memory limit disabled. If the actual heap usage is not too large, it might succeed without triggering an out-of-memory error.

15.1.3. Capacity planning

In many use cases, it is advantageous to try to cache as much of the data and indexes as possible. The following examples illustrate methods for estimating the page cache size, depending on whether you are already running in production or planning for a future deployment:

Example 95. Estimate page cache for the existing Neo4j databases

First, estimate the total size of data and indexes, and then multiply with some factor, for example 20%, to allow for growth.

```
$neo4j-home> bin/neo4j-admin memrec
...
...
...
# Total size of lucene indexes in all databases: 6690m
# Total size of data and native indexes in all databases: 35050m
```

You can see that the data volume and native indexes combined take up approximately 35GB. In your specific use case, you estimate that 20% will provide sufficient head room for growth.

`dbms.memory.pagecache.size` = 1.2 * (35GB) = 42GB

You configure the page cache by adding the following to *neo4j.conf*:

```
dbms.memory.pagecache.size=42GB
```

Example 96. Estimate page cache for a new Neo4j database

When planning for a future database, it is useful to run an import with a fraction of the data, and then multiply the resulting store size delta by that fraction plus some percentage for growth.

1. Run the `memrec` command to see the total size of the data and indexes in all current databases.

```
$neo4j-home> bin/neo4j-admin memrec
...
...
...
# Total size of lucene indexes in all databases: 6690m
# Total size of data and native indexes in all databases: 35050m
```

2. Import 1/100th of the data and again measure the data volume and native indexes of all databases.

```
$neo4j-home> bin/neo4j-admin memrec
...
...
...
# Total size of lucene indexes in all databases: 6690m
# Total size of data and native indexes in all databases: 35400m
```

   You can see that the data volume and native indexes combined take up approximately 35.4GB.

3. Multiply the resulting store size delta by that fraction.

   35.4GB - 35GB = 0.4GB * 100 = 40GB

4. Multiply that number by 1.2 to size up the result, and allow for 20% growth.

   `dbms.memory.pagecache.size` = 1.2 * (40GB) = 48GB

5. Configure the page cache by adding the following to *neo4j.conf*:

```
dbms.memory.pagecache.size=48G
```

15.1.4. Limit transaction memory usage

By using the `dbms.memory.transaction.global_max_size` setting you can configure a global maximum memory usage for all of the transactions running on the server. This setting must be configured low enough so that you do not run out of memory. If you are experiencing `OutOfMemory` messages during high transaction load, try to lower this limit.

Neo4j also offers the following settings to provide fairness, which can help improve stability in multi-tenant deployments.

- The setting `dbms.memory.transaction.database_max_size` limits the transaction memory usage per database.
- The setting `dbms.memory.transaction.max_size` constrains each transaction.

When any of the limits are reached, the transaction is terminated without affecting the overall health of the database.

To help configure these settings you can use the following commands to list the current usage:

```
CALL dbms.listPools()
CALL dbms.listTransactions()
CALL dbms.listQueries()
```

Or alternatively, you can enable `dbms.logs.query.allocation_logging_enabled` and monitor the memory usage of each query in the *query.log*.

By default, transaction sizes are unconstrained. However, in a Causal Cluster deployment, where an instance is configured as either a Core or a Read Replica, the maximum amount of memory each transaction is permitted to use is 2GB. Consequently, if your configuration contains `dbms.mode=CORE` or `dbms.mode=READ_REPLICA`, the largest value permitted for the `dbms.memory.transaction.max_size` setting is also 2GB.

15.2. Index configuration

15.2.1. Introduction

In Neo4j there are two different index types: b-tree and full-text.

B-tree indexes can be created and dropped using Cypher. Users typically do not have to know about the index in order to use it, since Cypher's query planner decides which index to use in which situation. B-tree indexes are good at exact look-ups on all types of values, and range scans, full scans, and prefix searches.

For details on the configuration aspects of b-tree indexes, see B-tree indexes.

Full-text indexes differ from b-tree indexes, in that they are optimised for indexing and searching text. They are used for queries that demand an understanding of language, and they only index string data. They must also be queried explicitly via procedures, as Cypher will not make plans that rely on them.

An example of a use case for full-text indexes is parsing a book for a certain term, and taking advantage of the knowledge that the book is written in a certain language. The use of an *analyzer* for that language will, among other things, enable you to exclude words that are not relevant for the search (for example "*if*" and "*and*"), and include conjugations of words that are.

Another use case example is indexing the various address fields and text data in a corpus of emails. Indexing this data using the `email` analyzer would enable someone to find all emails that are sent from, or to, or mentions, an email account.

In contrast to b-tree indexes, full-text indexes are created, queried, and dropped using built-in procedures. The use of full-text indexes do require a familiarity with how the indexes operate.

For details on the configuration aspects of full-text indexes, see Full-text indexes.

For details on creating, querying and dropping full-text indexes, see Cypher Manual → Indexes to support

full-text search.

The type of an index can be identified according to the table below:

Index type	Cypher command	Core API
B-tree index	`SHOW INDEXES#BTREE`	`org.neo4j.graphdb.schema.IndexType#BTREE`
Full-text index	`SHOW INDEXES#FULLTEXT`	`org.neo4j.graphdb.schema.IndexType#FULLTEXT`

15.2.2. B-tree indexes

B-tree indexes can be backed by two different index providers, `native-btree-1.0` and `lucene+native-3.0`. If not explicitly set, `native-btree-1.0` will be used.

For more information on the different index types, refer to Cypher Manual → Indexes to support full-text search.

Limitations

In this section a few limitations for b-tree indexes are described, together with suggested workarounds.

Limitations for queries using `CONTAINS` and `ENDS WITH`

The index provider `native-btree-1.0` has limited support for `ENDS WITH` and `CONTAINS` queries. These queries will not be able to do an optimized search as per queries that use `STARTS WITH`, `=`, and `<>`. Instead, the index result will be a stream of an index scan with filtering.

In the future, `ENDS WITH` and `CONTAINS` queries will be supported with full-text indexes, but for now the index provider `lucene+native-3.0` can be used instead. Please note that `lucene+native-3.0` only has support for `ENDS WITH` and `CONTAINS` for single property strings.

- For details about execution plans, refer to Cypher Manual → Execution plans.
- For details about string operators, refer to Cypher Manual → Operators.

Limitations on key size

The index provider `native-btree-1.0` has a key size limit of around 8kB.

If a transaction reaches the key size limit for one or more of its changes, that transaction will fail before committing any changes. If the limit is reached during index population, the resulting index will be in a failed state, and as such will not be usable for any queries.

If this is an issue, you can use the index provider `lucene+native-3.0` instead. This provider has a key size limit for single property strings of around 32kB.

Workarounds to address limitations

To workaround problems with key size, or performance issues related to `ENDS WITH` or `CONTAINS`, you can use the index provider `lucene+native-3.0`. This only works for single-property string indexes.

This can be done using either of the following methods:

Option 1. Use `OPTIONS` clause in create command (recommended)

The Cypher commands for index creation, unique property constraint creation, and node key creation contains an optional `OPTIONS` clause. This clause can be used to specify index provider.

For details on indexes, see Cypher Manual → Indexes for search performance. For details on constraints, see Cypher manual → Constraints.

Option 2. Use a built-in procedure Deprecated

Please note that this option uses built-in procedures that have been deprecated, and will be removed in a future release. These have been replaced with the Cypher commands in Option 1.

The built-in procedures `db.createIndex`, `db.createUniquePropertyConstraint`, and `db.createNodeKey` can be used to specify index provider on index creation, unique property constraint creation, and node key creation.

For details on constraints, see Procedures.

Option 3. Change the config Deprecated

Please note that this option uses the index setting `dbms.index.default_schema_provider`, which has been deprecated and will be removed in a future release. It will be a fully internal concern which index provider an index is using.

1. Configure the setting `dbms.index.default_schema_provider` to the one required.
2. Restart Neo4j.
3. Drop and recreate the relevant index.
4. Change `dbms.index.default_schema_provider` back to the original value.
5. Restart Neo4j.

   The recommended way to set index provider for an index is to use the `OPTIONS` clause for index creation, unique property constraint creation, and node key creation.

   For more information, see Cypher manual → Constraints.

Index migration

When upgrading a 3.5 store to 4.2.19, all indexes will be upgraded to the latest index version, and rebuilt automatically, with the exception for the indexes that were previously using Lucene for single-property strings. They will be upgraded to a fallback version which still use Lucene for those properties. Please note that they will still need to be rebuilt. For more information, see Upgrade and Migration Guide → Neo4j indexes.

Procedures to create index and index backed constraint Deprecated

Indexes and constraints are best created through Cypher, but can still be created through the deprecated procedures described in the example below. Index provider and index settings can both be specified using the optional `OPTIONS` clause for the Cypher commands.

Example 97. Example of procedures to create index and index backed constraint Deprecated

The following procedures provide the option to specify both index provider and index settings (optional). Note that settings keys need to be escaped with back-ticks if they contain dots.

Use `db.createIndex` procedure to create an index:

```
CALL db.createIndex("MyIndex", ["Person"], ["name"], "native-btree-1.0", {`spatial.cartesian.max`:
[100.0,100.0], `spatial.cartesian.min`: [-100.0,-100.0]})
```

If a settings map is not provided, the settings will be picked up from the Neo4j config file, the same way as when creating an index or constraint through Cypher.

```
CALL db.createIndex("MyIndex", ["Person"], ["name"], "native-btree-1.0")
```

Use `db.createUniquePropertyConstraint` to create a node property uniqueness constraint (the example is without settings map, left out for abbreviation):

```
CALL db.createUniquePropertyConstraint("MyIndex", ["Person"], ["name"], "native-btree-1.0")
```

Use `db.createNodeKey` to create node key constraint (the example is without settings map, left out for abbreviation):

```
CALL db.createNodeKey("MyIndex", ["Person"], ["name"], "native-btree-1.0")
```

15.2.3. Full-text indexes

Full-text indexes are powered by the Apache Lucene indexing and search library. A full description on how to create and use full-text indexes is provided in the Cypher Manual → Indexes to support full-text search.

Configuration

The following options are available for configuring full-text indexes:

`dbms.index.fulltext.default_analyzer`

The name of the analyzer that the full-text indexes should use by default. This setting only has effect when a full-text index is created, and will be remembered as an index-specific setting from then on.

The list of possible analyzers is available through the `db.index.fulltext.listAvailableAnalyzers()` Cypher procedure.

Unless otherwise specified, the default analyzer is `standard-no-stop-words`, which is the same as the `StandardAnalyzer` from Lucene, except no stop-words are filtered out.

`dbms.index.fulltext.eventually_consistent`

Used to declare whether full-text indexes should be eventually consistent, or not. This setting only has effect when a full-text index is created, and will be remembered as an index-specific setting from then on.

Indexes are normally fully consistent, and the committing of a transaction does not return until both the store and the indexes have been updated. Eventually consistent full-text indexes, on the other hand, are not updated as part of commit, but instead have their updates queued up and applied in a background thread. This means that there can be a short delay between committing a change, and that change becoming visible via any eventually consistent full-text indexes. This delay is just an artifact of the queueing, and will usually be quite small since eventually consistent indexes are updated "as soon as possible".

By default, this is turned off, and full-text indexes are fully consistent.

`dbms.index.fulltext.eventually_consistent_index_update_queue_max_length`

Eventually consistent full-text indexes have their updates queued up and applied in a background thread, and this setting determines the maximum size of that update queue. If the maximum queue size is reached, then committing transactions will block and wait until there is more room in the queue, before adding more updates to it.

This setting applies to all eventually consistent full-text indexes, and they all use the same queue. The maximum queue length must be at least 1 index update, and must be no more than 50 million due to heap space usage considerations.

The default maximum queue length is 10.000 index updates.

15.3. Tuning of the garbage collector

The heap is separated into an *old generation* and a *young generation*. New objects are allocated in the young generation, and then later moved to the old generation, if they stay live (in use) for long enough. When a generation fills up, the garbage collector performs a collection, during which all other threads in the process are paused. The young generation is quick to collect since the pause time correlates with the *live* set of objects. In the old generation, pause times roughly correlates with the size of the heap. For this reason, the heap should ideally be sized and tuned such that transaction and query state never makes it to the old generation.

The heap size is configured with the `dbms.memory.heap.max_size` (in MBs) setting in the *neo4j.conf* file. The initial size of the heap is specified by the `dbms.memory.heap.initial_size` setting, or with the `-Xms???m` flag, or chosen heuristically by the JVM itself if left unspecified. The JVM will automatically grow the heap as needed, up to the maximum size. The growing of the heap requires a full garbage collection cycle. It is recommended to set the initial heap size and the maximum heap size to the same value. This way the pause that happens when the garbage collector grows the heap can be avoided.

If the new generation is too small, short-lived objects may be moved to the old generation too soon. This is called premature promotion and will slow the database down by increasing the frequency of old

generation garbage collection cycles. If the new generation is too big, the garbage collector may decide that the old generation does not have enough space to fit all the objects it expects to promote from the new to the old generation. This turns new generation garbage collection cycles into old generation garbage collection cycles, again slowing the database down. Running more concurrent threads means that more allocations can take place in a given span of time, in turn increasing the pressure on the new generation in particular.

The *Compressed OOPs* feature in the JVM allows object references to be compressed to use only 32 bits. The feature saves a lot of memory but is only available for heaps up to 32 GB. The maximum applicable size varies from platform and JVM version. The `-XX:+UseCompressedOops` option can be used to verify whether the system can use the *Compressed OOPs* feature. If it cannot, this will will be logged in the default process output stream.

How to tune the specific garbage collection algorithm depends on both the JVM version and the workload. It is recommended to test the garbage collection settings under realistic load for days or weeks. Problems like heap fragmentation can take a long time to surface.

To gain good performance, these are the things to look into first:

- Make sure the JVM is not spending too much time performing garbage collection. The goal is to have a large enough heap to make sure that heavy/peak load will not result in so called GC-trashing. Performance can drop as much as two orders of magnitude when GC-trashing happens. Having too large heap may also hurt performance so you may have to try some different heap sizes.
- Neo4j needs enough heap memory for the transaction state and query processing, plus some head-room for the garbage collector. As heap memory requirements are so workload-dependent, it is common to see heap memory configurations from 1 GB, up to 32 GB.

Edit the following properties:

Table 75. neo4j.conf JVM tuning properties

Property Name	Meaning
`dbms.memory.heap.initial_size`	initial heap size (in MB)
`dbms.memory.heap.max_size`	maximum heap size (in MB)
`dbms.jvm.additional`	additional literal JVM parameter

15.4. Bolt thread pool configuration

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.

15.4.1. How thread pooling works

The Bolt thread pool has a minimum and a maximum capacity. It starts with a minimum number of threads available, and grows up to the maximum count depending on the workload. Threads that sit idle for longer than a specified time period are stopped and removed from the pool in order to free up resources.

However, the size of the pool will never go below the minimum.

Each connection being established is assigned to the connector's thread pool. Idle connections do not consume any resources on the server side, and they are monitored against messages arriving from the client. Each message arriving on a connection triggers the scheduling of a connection on an available thread in the thread pool. If all the available threads are busy, and there is still space to grow, a new thread is created and the connection is handed over to it for processing. If the pool capacity is filled up, and no threads are available to process, the job submission is rejected and a failure message is generated to notify the client of the problem.

The default values assigned to the Bolt thread pool will fit most workloads, so it is generally not necessary to configure the connection pool explicitly. If the maximum pool size is set too low, an exception will be thrown with an error message indicating that there are no available threads to serve. The message will also be written to *neo4j.log*.

Any connection with an active explicit, or implicit, transaction will stick to the thread that starts the transaction, and will not return that thread to the pool until the transaction is closed. Therefore, in applications that are making use of explicit transactions, it is important to close the transactions appropriately. To learn more about transactions, refer to the Neo4j Driver manuals.

15.4.2. Configuration options

The following configuration options are available for configuring the Bolt connector:

Table 76. Thread pool options

Option name	Default	Description
`dbms.connector.bolt.thread_pool_min_size`	`5`	The minimum number of threads that will always be up even if they are idle.
`dbms.connector.bolt.thread_pool_max_size`	`400`	The maximum number of threads that will be created by the thread pool.
`dbms.connector.bolt.thread_pool_keep_alive`	`5m`	The duration that the thread pool will wait before killing an idle thread from the pool. However, the number of threads will never go below `dbms.connector.bolt.thread_pool_min_size`.

15.4.3. How to size your Bolt thread pool

Select values for thread pool sizing based on your workload. Since each active transaction will borrow a thread from the pool until the transaction is closed, it is basically the minimum and maximum active transaction at any given time that determine the values for pool configuration options. You can use the monitoring capabilities (see Monitoring) of the database to discover more about your workload.

Configure `dbms.connector.bolt.thread_pool_min_size` based on your minimum or average workload. Since there will always be this many amount of threads in the thread pool, sticking with lower values may

be more resource-friendly than having too many idle threads waiting for job submissions.

Configure `dbms.connector.bolt.thread_pool_max_size` based on your maximum workload. This should basically be set after the maximum number of active transactions that is expected on the server. You should also account for non-transaction operations that will take place on the thread pool, such as connection and disconnection of clients.

Example 98. Configure the thread pool for a Bolt connector

In this example we configure the Bolt thread pool to be of minimum size `5`, maximum size `100`, and have a keep-alive time of `10 minutes`.

```
dbms.connector.bolt.thread_pool_min_size=10
dbms.connector.bolt.thread_pool_max_size=100
dbms.connector.bolt.thread_pool_keep_alive=10m
```

15.5. Linux file system tuning

It is recommended to disable file and directory access time updates by setting the `noatime,nodiratime` mount options in *fstab*, or when issuing the disk mount command. This way, the file system will not have to issue writes that update this meta-data, thus improving write performance.

Since databases can put a high and consistent load on a storage system for a long time, it is recommended to use a file system that has good aging characteristics. The EXT4 and XFS file systems are both supported.

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.

Neo4j does not recommend and support the usage of NFS or NAS as database storage.

15.6. Disks, RAM and other tips

As with any persistence solution, performance depends a lot on the persistence media used. In general, the faster storage you have, and the more of your data you can fit in RAM, the better performance you will get.

15.6.1. Storage

There are many performance characteristics to consider for your storage solutions. The performance can vary hugely in orders of magnitude. Generally, having all your data in RAM achieves maximum performance.

If you have multiple disks or persistence media available, it may be a good idea to divide the store files and transaction logs across those disks. Keeping the store files on disks with low seek time can do wonders for read operations.

Use tools like `dstat` or `vmstat` to gather information when your application is running. If the swap or paging numbers are high, that is a sign that the database does not quite fit in memory. In this case, database access can have high latencies.

To achieve maximum performance, it is recommended to provide Neo4j with as much RAM as possible to avoid hitting the disk.

15.6.2. Page cache

When Neo4j starts up, its page cache is empty and needs to warm up. The pages, and their graph data contents, are loaded into memory on demand as queries need them. This can take a while, especially for large stores. It is not uncommon to see a long period with many blocks being read from the drive, and high IO wait times. This will show up in the page cache metrics as an initial spike in page faults. The page fault spike is then followed by a gradual decline of page fault activity, as the probability of queries needing a page that is not yet in memory drops.

15.6.3. Active page cache warmup Enterprise edition

Neo4j Enterprise Edition has a feature called *active page cache warmup*, which is enabled by default via the `dbms.memory.pagecache.warmup.enable` configuration setting.

How it works

It shortens the page fault spike and makes the page cache warm up faster. This is done by periodically recording *cache profiles* of the store files while the database is running. These profiles contain information about what data is and is not in memory and are stored in the *data/databases/mydatabase/profiles* directory. When Neo4j is restarted next time, it looks for these cache profiles and loads the same data that was in memory when the profile was created. The profiles are also copied as part of the online backup and cluster store-copy operations and help warm up new databases that join a cluster.

The setting should remain enabled for most scenarios. However, when the workload changes after the database restarts, the setting can be disabled to avoid spending time fetching data that will be directly evicted.

Configuration options

Load the entire database into memory

It is also possible to configure `dbms.memory.pagecache.warmup.preload` to load the entire database data into memory. This is useful when the size of the database store is smaller than the available memory for the page cache. When enabled, it disables warmup by profile and prefetches data into the page cache as part of the startup.

Load specified files into memory

The files that you want to prefetched can be filtered using the `dbms.memory.pagecache.warmup.preload.allowlist` setting. It takes a regular expression as a value to match the files.

Example 99. Load only the nodes and relationships

For example, if you want to load only the nodes and relationships, you can use the regex `.*(node|relationship).*` to match the name of the store files. The *active page cache warmup* will prefetch the content of the following files:

```
neostore.nodestore.db
neostore.nodestore.db.id
neostore.nodestore.db.labels
neostore.nodestore.db.labels.id
neostore.relationshipgroupstore.db
neostore.relationshipgroupstore.db.id
neostore.relationshipstore.db
neostore.relationshipstore.db.id
neostore.relationshiptypestore.db
neostore.relationshiptypestore.db.id
neostore.relationshiptypestore.db.names
Neostore.relationshiptypestore.db.names.id
```

And can be verified using unix `grep`:

```
ls neo4j/ | grep -E '.*(node|relationship).*'
```

Configure the profile frequency for the page cache

The profile frequency is the rate at which the profiles are re-generated. More frequent means more accurate. A profile contains information about those parts of the files that are currently loaded into memory. By default, it is set to `dbms.memory.pagecache.warmup.profile.interval=1m`. It takes some time to generate these profiles, and therefore `1m` is a good interval. If the workload is very stable, then the profile will not change much. Accordingly, if the workload changes often, the profile will thus often become outdated.

15.6.4. Checkpoint IOPS limit Enterprise edition

Neo4j flushes its page cache in the background as part of its checkpoint process. This will show up as a period of elevated write IO activity. If the database is serving a write-heavy workload, the checkpoint can slow the database down by reducing the IO bandwidth that is available to query processing. Running the database on a fast SSD, which can service a lot of random IOs, significantly reduces this problem. If a fast SSD is not available in your environment, or if it is insufficient, then an artificial IOPS limit can be placed on the checkpoint process. The `dbms.checkpoint.iops.limit` restricts the IO bandwidth that the checkpoint process is allowed to use. Each IO is, in the case of the checkpoint process, an 8 KiB write. An IOPS limit of 600, for instance, would thus only allow the checkpoint process to write at a rate of roughly 5 MiB per second. This will, on the other hand, make checkpoints take longer to complete. A longer time between checkpoints can cause more transaction log data to accumulate, and can lengthen recovery times. See the transaction logs section for more details on the relationship between checkpoints and log pruning. The IOPS limit can be changed at runtime, making it possible to tune it until you have the right balance between IO usage and checkpoint time.

15.7. Statistics and execution plans

When a Cypher query is issued, it gets compiled to an execution plan that can run and answer the query. The Cypher query engine uses the available information about the database, such as schema information about which indexes and constraints exist in the database. Neo4j also uses statistical information about the database to optimize the execution plan. For more information, see Cypher Manual → Execution plans.

15.7.1. Configure statistics collection

The Cypher query planner depends on accurate statistics to create efficient plans. Therefore, these statistics are kept up-to-date as the database evolves.

For each database in the DBMS, Neo4j collects the following statistical information and keeps it up-to-date:

For graph entities

- The number of nodes with a certain label.
- The number of relationships by type.
- The number of relationships by type between nodes with a specific label.

These numbers are updated whenever you set or remove a label from a node.

For database schema

- Selectivity per index.

To produce a selectivity number, Neo4j runs a full index scan in the background. Because this could potentially be a very time-consuming operation, a full index scan is triggered only when the changed data reaches a specified threshold.

Automatic statistics collection

You can control whether and how often statistics are collected automatically by configuring the following settings:

Parameter name	Default value	Description
`dbms.index_sampling.background_enabled`	`true`	Enable the automatic (background) index sampling.
`dbms.index_sampling.update_percentage`	`5`	Percentage of index updates of total index size required before sampling of a given index is triggered.

Manual statistics collection

You can manually trigger index resampling by using the built-in procedures `db.resampleIndex()` and `db.resampleOutdatedIndexes()`.

`db.resampleIndex()`

Trigger resampling of a specified index.

```
CALL db.resampleIndex("indexName")
```

`db.resampleOutdatedIndexes()`

Trigger resampling of all outdated indexes.

```
CALL db.resampleOutdatedIndexes()
```

15.7.2. Configure the replanning of execution plans

Execution plans are cached and are not replanned until the statistical information used to produce the plan changes.

Automatic replanning

You can control how sensitive the replanning should be to database updates by configuring the following settings:

Parameter name	Default value	Description
`cypher.statistics_divergence_threshold`	`0.75`	The threshold for statistics above which a plan is considered stale. When the changes to the underlying statistics of an execution plan meet the specified threshold, the plan is considered stale and is replanned. Change is calculated as `abs(a-b)/max(a,b)`. This means that a value of `0.75` requires the database to approximately quadruple in size before replanning occurs. A value of `0` means that the query is replanned as soon as there is a change in the statistics and the replan interval elapses.
`cypher.min_replan_interval`	`10s`	The minimum amount of time between two query replanning executions. After this time, the graph statistics are evaluated, and if they have changed more than the value set in `cypher.statistics_divergence_threshold`, the query is replanned. Each time the statistics are evaluated, the divergence threshold is reduced until it reaches 10% after about 7h. This ensures that even moderately changing databases see query replanning after a sufficiently long time interval.

Manual replanning

You can manually force the database to replan the execution plans that are already in the cache by using the following built-in procedures:

`db.clearQueryCaches()`

Clear all query caches. Does not change the database statistics.

```
CALL db.clearQueryCaches()
```

`db.prepareForReplanning()`

Completely recalculates all database statistics to be used for any subsequent query planning.

The procedure triggers an index resampling, waits for it to complete, and clears all query caches. Afterwards, queries are planned based on the latest database statistics.

```
CALL db.prepareForReplanning()
```

You can use Cypher replanning to specify whether you want to force a replan, even if the plan is valid according to the planning rules, or skip replanning entirely should you wish to use a valid plan that already exists.

For more information, see:

- Cypher manual → Cypher replanning
- Cypher manual → Execution plans
- Procedures

15.8. Space reuse

Neo4j uses logical deletes to remove data from the database to achieve maximum performance and scalability. A logical delete means that all relevant records are marked as deleted, but the space they occupy is not immediately returned to the operating system. Instead, it is subsequently reused by the transactions *creating* data.

Marking a record as deleted requires writing a record update command to the transaction log, as when something is created or updated. Therefore, when deleting large amounts of data, this leads to a storage usage growth of that particular database, because Neo4j writes records for all deleted nodes, their properties, and relationships to the transaction log.

Keep in mind that when doing `DETACH DELETE` on many nodes, those deletes can take up more space in the in-memory transaction state and the transaction log than you might expect.

Transactions are eventually pruned out of the transaction log, bringing the storage usage of the log back down to the expected level. The store files, on the other hand, do not shrink when data is deleted. The space that the deleted records take up is kept in the store files. Until the space is reused, the store files are sparse and fragmented, but the performance impact of this is usually minimal.

15.8.1. ID files

Neo4j uses *.id* files for managing the space that can be reused. These files contain the set of IDs for all the deleted records in their respective files. The ID of the record uniquely identifies it within the store file. For instance, the `neostore.nodestore.db.id` contains the IDs of all deleted nodes.

These *.id* files are maintained as part of the write transactions that interact with them. When a write transaction commits a deletion, the record's ID is buffered in memory. The buffer keeps track of all overlapping unfinished transactions. When they complete, the ID becomes available for reuse.

The buffered IDs are flushed to the *.id* files as part of the checkpointing. Concurrently, the *.id* file changes (the ID additions and removals) are inferred from the transaction commands. This way, the recovery process ensures that the *.id* files are always in-sync with their store files. The same process also ensures that clustered databases have precise and transactional space reuse.

If you want to shrink the size of your database, do not delete the *.id* files. The store files must *only* be modified by the Neo4j database and the `neo4j-admin` tools.

15.8.2. Reclaim unused space

You can use the `neo4j-admin copy` command to create a defragmented copy of your database. The `copy` command creates and entirely new and independent database. If you want to run that database in a cluster, you have to re-seed the existing cluster, or seed a new cluster from that copy.

Example 100. Example of database compaction using `neo4j-admin copy`

The following is a detailed example on how to check your database store usage and how to reclaim space.

Let's use the Cypher Shell command-line tool to add 100k nodes and then see how much store they occupy.

1. In a running Neo4j standalone instance, log in to the Cypher Shell command-line tool with your credentials.

```
$neo4j-home/bin$> ./cypher-shell -u neo4j -p <password>
```

```
Connected to Neo4j at neo4j://localhost:7687 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
```

2. Add 100k nodes to the `neo4j` database using the following command:

```
neo4j@neo4j> foreach (x in range (1,100000) | create (n:testnode1 {id:x}));
```

```
0 rows available after 1071 ms, consumed after another 0 ms
Added 100000 nodes, Set 100000 properties, Added 100000 labels
```

3. Check the allocated ID range:

```
neo4j@neo4j> MATCH (n:testnode1) RETURN ID(n) as ID order by ID limit 5;
```

```
+----+
| ID |
+----+
| 0  |
| 1  |
| 2  |
| 3  |
| 4  |
+----+

5 rows available after 171 ms, consumed after another 84 ms
```

4. Run `call db.checkpoint()` procedure to force a checkpoint.

```
neo4j@neo4j> call db.checkpoint();
```

```
+-----------------------------------+
| success | message                 |
+-----------------------------------+
| TRUE    | "Checkpoint completed." |
+-----------------------------------+

1 row available after 18 ms, consumed after another 407 ms
```

5. In Neo4j Browser, run `:sysinfo` to check the total store size of `neo4j`.

   The reported output for the store size is 791.92 KiB, ID Allocation: Node ID 100000, Property ID 100000.

6. Delete the above created nodes.

```
neo4j@neo4j> Match (n) detach delete n;
```

7. Run `call db.checkpoint()` procedure again.

```
neo4j@neo4j> call db.checkpoint();
```

```
+-----------------------------------+
| success | message                 |
+-----------------------------------+
| TRUE    | "Checkpoint completed." |
+-----------------------------------+

1 row available after 18 ms, consumed after another 407 ms
```

8. In Neo4j Browser, run `:sysinfo` to check the total store size of `neo4j`.

   The reported output for the store size is 31.01 MiB, ID Allocation: Node ID 100000, Property ID 100000.

By default, a checkpoint flushes any cached updates in pagecache to store files. Thus, the allocated IDs remain unchanged, and the store size increases or does not alter (if the instance restarts) despite the deletion. In a production database, where numerous load/deletes are frequently performed, the result is a significant unused space occupied by store files.

To reclaim that unused space, you can use the neo4j-admin copy command to create a defragmented copy of your database. Use the `system` database and stop the `neo4j` database before running the command.

1. Invoke the `neo4j-admin copy` command to create a copy of your `neo4j` database.

```
$neo4j-home/bin$> ./neo4j-admin copy --to-database=neo4jcopy1 --from-database=neo4j --force
--verbose
```

```
Starting to copy store, output will be saved to: $neo4j_home/logs/neo4j-admin-copy-2020-11-
04.11.30.57.log
2020-10-23 11:40:00.749+0000 INFO [StoreCopy] ### Copy Data ###
2020-10-23 11:40:00.750+0000 INFO [StoreCopy] Source: $neo4j_home/data/databases/neo4j (page
cache 8m) (page cache 8m)
2020-10-23 11:40:00.750+0000 INFO [StoreCopy] Target: $neo4j_home/data/databases/neo4jcopy1 (page
cache 8m)
2020-10-23 11:40:00.750+0000 INFO [StoreCopy] Empty database created, will start importing
readable data from the source.
2020-10-23 11:40:02.397+0000 INFO [o.n.i.b.ImportLogic] Import starting
Nodes, started 2020-11-04 11:31:00.088+0000
[*Nodes:?? 7.969MiB--------------------------------------------------------------------------------]
100K Δ 100K
Done in 632ms
Prepare node index, started 2020-11-04 11:31:00.735+0000
[*DETECT:7.969MiB----------------------------------------------------------------------------------]
0 Δ    0
Done in 79ms
Relationships, started 2020-11-04 11:31:00.819+0000
[*Relationships:?? 7.969MiB------------------------------------------------------------------------]
0 Δ    0
Done in 37ms
Node Degrees, started 2020-11-04 11:31:01.162+0000
[*>:??---------------------------------------------------------------------------------------------]
0 Δ    0
Done in 12ms
Relationship --> Relationship 1/1, started 2020-11-04 11:31:01.207+0000
[*>:??---------------------------------------------------------------------------------------------]
0 Δ    0
Done in 0ms
RelationshipGroup 1/1, started 2020-11-04 11:31:01.232+0000
[*>:??---------------------------------------------------------------------------------------------]
0 Δ    0
Done in 10ms
Node --> Relationship, started 2020-11-04 11:31:01.245+0000
[*>:??---------------------------------------------------------------------------------------------]
0 Δ    0
Done in 10ms
Relationship <-- Relationship 1/1, started 2020-11-04 11:31:01.287+0000
[*>:??---------------------------------------------------------------------------------------------]
0 Δ    0
Done in 0ms
Count groups, started 2020-11-04 11:31:01.549+0000
[*>:??---------------------------------------------------------------------------------------------]
0 Δ    0
Done in 0ms
Node --> Group, started 2020-11-04 11:31:01.579+0000
[*>:??---------------------------------------------------------------------------------------------]
0 Δ    0
Done in 1ms
Node counts and label index build, started 2020-11-04 11:31:01.986+0000
[*>:??---------------------------------------------------------------------------------------------]
0 Δ    0
Done in 11ms
Relationship counts, started 2020-11-04 11:31:02.034+0000
[*>:??---------------------------------------------------------------------------------------------]
0 Δ    0
Done in 0ms

IMPORT DONE in 3s 345ms.
Imported:
  0 nodes
  0 relationships
  0 properties
Peak memory usage: 7.969MiB
2020-11-04 11:31:02.835+0000 INFO [o.n.i.b.ImportLogic] Import completed successfully, took 3s
345ms. Imported:
  0 nodes
  0 relationships
  0 properties
2020-11-04 11:31:03.330+0000 INFO [StoreCopy] Import summary: Copying of 100704 records took 5
seconds (20140 rec/s). Unused Records 100704 (100%) Removed Records 0 (0%)
2020-11-04 11:31:03.330+0000 INFO [StoreCopy] ### Extracting schema ###
2020-11-04 11:31:03.330+0000 INFO [StoreCopy] Trying to extract schema...
2020-11-04 11:31:03.338+0000 INFO [StoreCopy] ... found 0 schema definitions.
```

The example resulted in a compact and consistent store (any inconsistent nodes, properties, relationships are not copied over to the newly created store).

2. Use the `system` database and create the `neo4jcopy1` database.

```
neo4j@system> create database neo4jcopy1;
```

```
0 rows available after 60 ms, consumed after another 0 ms
```

3. Verify that the `neo4jcopy1` database is online.

```
neo4j@system> show databases;
+-----------------------------------------------------------------------------------------------------
----+
| name         | address          | role         | requestedStatus | currentStatus | error |
default |
+-----------------------------------------------------------------------------------------------------
----+
| "neo4j"      | "localhost:7687" | "standalone" | "offline"       | "offline"     | ""    | TRUE
|
| "neo4jcopy1" | "localhost:7687" | "standalone" | "online"        | "online"      | ""    |
FALSE   |
| "system"     | "localhost:7687" | "standalone" | "online"        | "online"      | ""    |
FALSE   |
+-----------------------------------------------------------------------------------------------------
----+

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

4. In Neo4j Browser, run `:sysinfo` to check the total store size of `neo4jcopy1`.

   The reported output for the store size after the compaction is 800.68 KiB, ID Allocation: Node ID 0, Property ID 0.

Chapter 16. Tools

This chapter comprises the following topics:

- Neo4j Admin — A description of the Neo4j Admin tool.
- Consistency checker — How to check the consistency of a Neo4j database using Neo4j Admin.
- Neo4j Admin report — How to collect the most common information needed for remote assessments.
- Display store information — How to display information about a database store.
- Memory recommendations — How to get an initial recommendation for Neo4j memory settings.
- Import — How to import data into Neo4j using the command `neo4j-admin import`.
- Unbind a Core Server — How to remove cluster state data from a Neo4j server.
- Push to Neo4j AuraDB — How to push an existing Neo4j graph to Neo4j AuraDB.
- Cypher Shell — How to use the Cypher Shell.

16.1. Neo4j Admin

16.1.1. Introduction

Neo4j Admin is the primary tool for managing a Neo4j DBMS. It is a command-line tool that is installed as part of the product and can be executed with a number of commands. Some of the commands are described in more detail in separate sections.

The Neo4j Admin commands must be invoked with the same user as Neo4j runs as. This guarantees that Neo4j will have full rights to start and work with the database files you use.

16.1.2. 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	`help <command>`	Display help text for `<command>`
	`check-consistency`	Check the consistency of a database. For details, see Consistency checker.
	`report`	Collect the most common information needed for remote assessments. For details, see Neo4j Admin report.
	`store-info`	Print information about a Neo4j database store. For details, see Display store information.
	`memrec`	Print Neo4j heap and pagecache memory settings recommendations. For details, see Memory recommendations.
	`import`	Import from a collection of CSV files or a pre-3.0 database. For details, see Import.
	`copy`	Copy data from an existing database to a new database. For details, see Copy a database store.
	`push-to-cloud`	Dumps a local database, and imports into a specified Neo4j AuraDB instance. For details, see Push to Neo4j AuraDB.
Authentication	`set-default-admin`	Sets the default admin user when no roles are present.
	`set-initial-password`	Sets the initial password of the initial admin user (`neo4j`). For details, see Set an initial password.

Functionality area	Command	Description
Offline backup For details see Back up an offline database and Restore a database dump.	`dump`	Dump a database into a single-file archive.
	`load`	Load a database from an archive created with the dump command.
Online backup For details see Back up an online database and Backup and restore.	`backup`	Perform an online backup from a running Neo4j server.
	`restore`	Restore a backed-up database.
Clustering	`unbind`	Removes cluster state data from a stopped Neo4j server. For details, see Unbind a Core Server.

16.1.3. 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.

16.1.4. Exit codes

When `neo4j-admin` finishes as expected, it returns an exit code of `0`. A non-zero exit code means something undesired happened during command execution. The non-zero exit code can contain further information about the error, such as the `backup` command's exit codes.

16.2. Consistency checker

The consistency of a database or a backup can be checked using the `check-consistency` argument to the `neo4j-admin` tool. The `neo4j-admin` tool is located in the `bin` directory. If checking the consistency of a database, note that it has to be stopped first or else the consistency check will result in an error.

It is not recommended to use an NFS to check the consistency of a database or a backup as this slows the process down significantly.

Syntax

```
neo4j-admin check-consistency ([--database=<database>] | [--backup=<path>])
                              [--verbose] [--additional-config=<path>]
                              [--check-graph=<true/false>]
                              [--check-indexes=<true/false>]
                              [--check-index-structure=<true/false>]
                              [--check-label-scan-store=<true/false>]
                              [--check-property-owners=<true/false>]
                              [--report-dir=<path>]`
```

Options

Option	Default	Description
--database	neo4j	Name of database.
--backup		Path to backup to check consistency of. Cannot be used together with --database.
--additional-config		Configuration file to supply additional configuration in.
--verbose	false	Enable verbose output.
--report-dir	.	Directory to write report file in.
--check-graph	true	Perform checks between nodes, relationships, properties, types and tokens.
--check-indexes	true	Perform checks on indexes by comparing content with the store.
--check-index-structure	true	Perform physical structure check on indexes. No comparison with the store takes place.
--check-label-scan-store	true	Perform checks on the label scan store.
--check-property-owners	false	Perform additional checks on property ownership. This check is **very** expensive in time and memory.

Output

If the consistency checker does not find errors, it will exit cleanly and not produce a report. If the consistency checker finds errors, it will exit with an exit code of `1` and write a report file with a name on the format `inconsistencies-YYYY-MM-DD.HH24.MI.SS.report`. The location of the report file is the current working directory, or as specified by the parameter `report-dir`.

Example 101. Run the consistency checker

Run with the `--database` option to check the consistency of a database. Note that the database must be stopped first.

```
$neo4j-home> bin/neo4j-admin check-consistency --database=neo4j

2019-11-13 12:42:14.479+0000 INFO [o.n.k.i.s.f.RecordFormatSelector] Selected
RecordFormat:StandardV4_0[SF4.0.b] record format from store /data/databases/neo4j
2019-11-13 12:42:14.481+0000 INFO [o.n.k.i.s.f.RecordFormatSelector] Format not configured for store
/data/databases/neo4j. Selected format from the store files: RecordFormat:StandardV4_0[SF4.0.b]
Index structure consistency check
..................  10%
..................  20%
..................  30%
..................  40%
..................  50%
..................  60%
..................  70%
..................  80%
..................  90%
.................. 100%
Full Consistency Check
..................  10%
..................  20%
..................  30%
..................  40%
..................  50%
..................  60%
..................  70%
..................  80%
..................  90%
.Checking node and relationship counts
..................  10%
..................  20%
..................  30%
..................  40%
..................  50%
..................  60%
..................  70%
..................  80%
..................  90%
.................. 100%
```

Run with the `--backup` option to check the consistency of a backup.

```
bin/neo4j-admin check-consistency --backup backup/neo4j-backup
```

`neo4j-admin check-consistency` cannot be applied to the Fabric virtual database. It must be run directly on the databases that are part of the Fabric setup.

16.3. Neo4j Admin report

Use the `report` command of `neo4j-admin` to gather information about a Neo4j installation and save it to an archive.

```
neo4j-admin report [--force] [--list] [--verbose] [--pid=<pid>] [--to=<path>] [<classifier>…]
```

The intended usage of the report tool is to simplify the support process by collecting the relevant information in a standard way. This tool does not send any information automatically. To share this

information with the Neo4j Support organization, you will have to send it manually.

Options

Option	Default	Description
--to	reports/	Specifies to target directory where the report should be written to.
--list		Will list available classifiers.
--verbose		Will instruct the tool to print more verbose output.
--force		Will disable the available disk space check.
--pid		Specify process id of running Neo4j instance. Only applicable when used together with classifiers indicated as *Online* in the table below.

By default, the tool tries to estimate the final size of the report and use that to assert that there is enough disk space available for it. If there is not enough available space the tool will abort. However, this estimation is pessimistic and does not take the compression into consideration, so if you are confident that you do have enough disk space, you can disable this check with the option `--force`.

Classifiers

Classifier	Online	Description
`all`		Include all of the available classifiers listed below.
`ccstate`		Include the current cluster state.
`config`		Include the `neo4j.conf` file.
`heap`	X	Include a heap dump.
`logs`		Include log files, e.g `debug.log`, `neo4j.log` etc.
`metrics`		Include the collected metrics.
`plugins`		Include a text view of the plugin directory (no files are collected).
`ps`		include a list of running processes.
`raft`		Include the raft log.
`sysprop`	X	Include a list of Java system properties.
`threads`	X	Include a thread dump of the running instance.

Classifier	Online	Description
`tree`		Include a text view of the folder structure of the data directory (no files are collected).
`tx`		Include transaction logs.

The classifiers indicated as *Online* only work when you have a running Neo4j instance that the tool can find.

If no classifiers are specified, the following classifiers will be used: `logs`, `config`, `plugins`, `tree`, `metrics`, `threads`, `sysprop` and `ps`.

The report tool does not read any data from your database. However, the heap, the raft logs, and the transaction logs may contain data. Additionally while the standard `neo4j.conf` file does not contain password information, for certain configurations it may have this type of information. Therefore, be aware of your organization's data security rules before using the classifiers `heap`, `tx`, `raft` and `config`.

This tool uses the Java Attach API to gather data from a running Neo4j instance. Therefore, it requires the Java JDK in order to execute properly.

16.4. Display store information

The `neo4j-admin store-info` command outputs information about the *store format* for a given *database store*.

1. The store format version.
2. When the store format version was introduced.
3. Whether the store format needs to be migrated to a newer version.

The store format can be set with the `dbms.record_format` configuration setting.

The store formats are:

- `aligned`
- `standard`
- `high_limit` Enterprise edition

16.4.1. Syntax

The `neo4j-admin store-info` command is located in the *bin* directory. It is invoked against an **offline** database store or a backup as follows:

```
neo4j-admin store-info [--all] [--structured] [--verbose] <path>
```

`<path>` — Path to database store, or databases directory if `--all` option is used.

16.4.2. Options

Table 77. Options

Name	Description
`--verbose`	Enable verbose output.
`--structured`	Return result structured as JSON.
`--all`	Return store format info for all databases at provided path.

16.4.3. Examples

Example 102. Invoke `neo4j-admin store-info` against a database store

```
bin/neo4j-admin store-info data/databases/mygraph.db
```

Output:

```
Store format version:         SF4.0.0
Store format introduced in:   4.0.0
```

Example 103. Invoke `neo4j-admin store-info` against a database backup

You can run the `store-info` command to see if the store format of the backup that you want to restore, is compatible with your running Neo4j instance. For example, if you want to restore the database backup */tmp/3518/mygraph.db* into a 4.x Neo4j instance:

```
bin/neo4j-admin store-info /tmp/3518/mygraph.db
```

Output:

```
Store format version:         vE.H.4    ①
Store format introduced in:   3.4.0     ②
Store format superseded in:   4.0.0     ③
```

① The store format version reveals that the database is configured to use the `high_limit`, see `dbms.record_format`.

② The store format version was introduced in `Neo4j 3.4.0`.

③ The store format of the current instance is 4.0.0, which means that a format migration must be performed if you want to restore this backup into the current instance.

For more information on how to migrate a single database, see Upgrade and Migration Guide → Tutorial: Back up and copy a database in a standalone instance.

Example 104. Invoke `neo4j-admin store-info` against a root containing several databases

The command can also be invoked against a root directory containing several databases, as follows:

```
neo4j-admin store-info <path> --all
```

```
bin/neo4j-admin store-info data/databases --all
```

Output:

```
Database name:                foo
Database in use:              false
Store format version:         SF4.0.0
Store format introduced in:   4.0.0
Last committed transaction id:2
Store needs recovery:         true

Database name:                bar
Database in use:              true
```

When the command is invoked against several databases, if some are **online** they will simply report as `in use` and exclude all other information.

Example 105. Invoke `neo4j-admin store-info` against a database and output JSON

If you are parsing the results of this command you may use the `--structured` option to receive the output as JSON. All the same fields are included and all values are strings.

```
bin/neo4j-admin store-info data/databases/foo --structured
```

Output:

```
{"databaseName":"foo",
"inUse":"false",
"storeFormat”:"SF4.0.0",
"storeFormatIntroduced”:"4.0.0",
"lastCommittedTransaction":"2",
"recoveryRequired":"true"}
```

16.4.4. Store format — `aligned`

Table 78. Store versions — `aligned`

Store Format Name	Store Format Version	Neo4j Version
`ALIGNED_V4_1`	`AF4.1.a`	`4.1.0`

Table 79. Store limits — `aligned`

Name	Limit
Property keys	`2^24` (16 777 216)

Name	Limit
Nodes	2^35 (34 359 738 368)
Relationships	2^35 (34 359 738 368)
Properties	2^36 (68 719 476 736)
Labels	2^32 (4 294 967 296)
Relationship types	2^16 (65 536)
Relationship groups	2^35 (34 359 738 368)

16.4.5. Store format — standard

Table 80. Store versions — standard

Store Format Name	Store Format Version	Neo4j Version
STANDARD_V4_0	SF4.0.0	4.0.0
STANDARD_V3_4	v0.A.9	3.4.0

Table 81. Store limits — standard

Name	Limit
Property keys	2^24 (16 777 216)
Nodes	2^35 (34 359 738 368)
Relationships	2^35 (34 359 738 368)
Properties	2^36 (68 719 476 736)
Labels	2^32 (4 294 967 296)
Relationship types	2^16 (65 536)
Relationship groups	2^35 (34 359 738 368)

16.4.6. Store format — high_limit Enterprise edition

Table 82. Store versions — high_limit

Store Format Name	Store Format Version	Neo4j Version
HIGH_LIMIT_V4_0_0	HL4.0.0	4.0.0
HIGH_LIMIT_V3_4_0	vE.H.4	3.4.0
HIGH_LIMIT_V3_2_0	vE.H.3	3.2.0
HIGH_LIMIT_V3_1_0	vE.H.2	3.1.0
HIGH_LIMIT_V3_0_6	vE.H.0b	3.0.6
HIGH_LIMIT_V3_0_0	vE.H.0	3.0.0

Table 83. Store limits — high_limit

Name	Limit
Property keys	`2^24` (16 777 216)
Nodes	`2^50` (1 Quadrillion)
Relationships	`2^50` (1 Quadrillion)
Properties	`2^50` (1 Quadrillion)
Labels	`2^32` (4 294 967 296)
Relationship types	`2^24` (16 777 216)
Relationship groups	`2^50` (1 Quadrillion)

16.5. Memory recommendations

Use the `memrec` command of `neo4j-admin` to get an initial recommendation on how to configure memory parameters for Neo4j:

```
neo4j-admin memrec --memory=<memory dedicated to Neo4j>, --verbose, --docker
```

The command gives heuristic memory setting recommendations for the Neo4j JVM heap and pagecache. The heuristic is based on the total memory of the system the command is running on, or on the amount of memory specified with the `--memory` argument. The heuristic assumes that the system is dedicated to running Neo4j. If this is not the case, then use the `--memory` argument to specify how much memory can be expected to be dedicated to Neo4j. The default output is formatted as such that it can be copy-pasted into *neo4j.conf*. The argument `--docker` outputs environmental variables that can be passed to a Neo4j docker container. For a detailed example, see Use Neo4j Admin for memory recommendations.

Options

Option	Default	Description
`--memory=<size>`	The memory capacity of the machine.	The amount of memory to allocate to Neo4j. Valid units are: `k`, `K`, `m`, `M`, `g`, `G`.
`--verbose`		Enable verbose output.
`--docker`		Enable output formatted as environmental variables that can be passed to a Neo4j docker container.

Considerations

The `neo4j-admin memrec` command calculates a valid starting point for Neo4j memory settings, based on the provided memory. The specific conditions for your use case may warrant adjustment of these values. See Memory configuration for a description of the memory settings in Neo4j.

Example 106. Use the `memrec` command of `neo4j-admin`

The following example illustrates how `neo4j-admin memrec` provides a recommendation on how to use 16g of memory:

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

...
...
...
# Based on the above, the following memory settings are recommended:
dbms.memory.heap.initial_size=5g
dbms.memory.heap.max_size=5g
dbms.memory.pagecache.size=7g
```

For an example on how to use the `neo4j-admin memrec` command, see Inspect the memory settings of all databases in a DBMS.

16.6. Import

There are two ways to import data from CSV files into Neo4j: via `neo4j-admin import` or `LOAD CSV`.

If you want to do batch imports of large amounts of data into a Neo4j database from CSV files, use the `import` command of `neo4j-admin`. This command can **only** be used to load data into a *previously unused* database and can only be performed **once** per database. By default, this database is set to `graph.db` but can be configured to other names and locations.

If you wish to import small to medium-sized CSV files into an existing database, use `LOAD CSV`. `LOAD CSV` can be run as many times as needed and does not require an empty database.

However, using the `import` command of `neo4j-admin` is generally faster since it is run against a stopped and empty database. This section describes the `neo4j-admin import` option.

For information on `LOAD CSV`, see the Cypher Manual → `LOAD CSV`.
For in-depth examples of using the command `neo4j-admin import`, refer to the Tutorials → Neo4j Admin import.
To create a cluster based on imported data, see Seed a cluster using the import tool.

These are some things you will need to keep in mind when creating your input files:

- Fields are comma-separated by default but a different delimiter can be specified.
- All files must use the same delimiter.
- Multiple data sources can be used for both nodes and relationships.
- A data source can optionally be provided using multiple files.
- A separate file with a header that provides information on the data fields, must be the first specified file of each data source.
- Fields without corresponding information in the header will not be read.

- UTF-8 encoding is used.
- By default, the importer will trim extra whitespace at the beginning and end of strings. Quote your data to preserve leading and trailing whitespaces.

Indexes and constraints

Indexes and constraints are not created during the import. Instead, you will need to add these afterwards (see Cypher Manual → Indexes).

16.6.1. Syntax

The syntax for importing a set of CSV files is:

```
neo4j-admin import [--expand-commands]
                   [--verbose]
                   [--cache-on-heap[=<true/false>]]
                   [--high-io[=<true/false>]]
                   [--ignore-empty-strings[=<true/false>]]
                   [--ignore-extra-columns[=<true/false>]]
                   [--legacy-style-quoting[=<true/false>]]
                   [--multiline-fields[=<true/false>]]
                   [--normalize-types[=<true/false>]]
                   [--skip-bad-entries-logging[=<true/false>]]
                   [--skip-bad-relationships[=<true/false>]]
                   [--skip-duplicate-nodes[=<true/false>]]
                   [--trim-strings[=<true/false>]]
                   [--additional-config=<path>]
                   [--array-delimiter=<char>]
                   [--bad-tolerance=<num>]
                   [--database=<database>]
                   [--delimiter=<char>]
                   [--id-type=<STRING|INTEGER|ACTUAL>]
                   [--input-encoding=<character-set>]
                   [--max-memory=<size>]
                   [--processors=<num>]
                   [--quote=<char>]
                   [--read-buffer-size=<size>]
                   [--report-file=<path>]
                   --nodes=[<label>[:<label>]...=]<files>...
                   [--nodes=[<label>[:<label>]...=]<files>...]...
                   [--relationships=[<type>=]<files>...]...
```

Example 107. Import data from CSV files

Assume that we have formatted our data as per CSV header format so that we have it in six different files:

1. `movies_header.csv`
2. `movies.csv`
3. `actors_header.csv`
4. `actors.csv`
5. `roles_header.csv`
6. `roles.csv`

The following command 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
```

Example 108. Import data from CSV files using regular expression

Assume that we want to include a header and then multiple files that matches a pattern, e.g. containing numbers. In this case a regular expression can be used. It is guaranteed that groups of digits will be sorted in numerical order, as opposed to lexicograghic order.

For example:

```
neo4j_home$ bin/neo4j-admin import --nodes import/node_header.csv,import/node_data_\d+\.csv
```

Example 109. Import data from CSV files using a more complex regular expression

For regular expression patterns containing commas, which is also the delimiter between files in a group, the pattern can be quoted to preserve the pattern.

For example:

```
neo4j_home$ bin/neo4j-admin import --nodes import/node_header.csv,'import/node_data_\d{1,5}.csv'
```

If importing to a database that has not explicitly been created prior to the import, it must be created subsequently in order to be used.

16.6.2. Options

Table 84. `neo4j-admin import` options

name
--expand-commands
--verbose
--cache-on-heap
--high-io
--ignore-empty-strings
--ignore-extra-columns
--legacy-style-quoting
--multiline-fields
--normalize-types
--skip-bad-entries-logging
--skip-bad-relationships
--skip-duplicate-nodes
--trim-strings
--additional-config
--array-delimiter
--bad-tolerance
--database
--delimiter
--id-type
--input-encoding
--max-memory
--processors
--quote
--read-buffer-size
--report-file
--nodes
--relationships

Some of the options below are marked as **Advanced**. These options should not be used for experimentation.

For more information, please contact Neo4j Professional Services.

`--expand-commands`

Allow command expansion in config value evaluation.

`--verbose`

Enable verbose output.

`--cache-on-heap[=<true/false>]` *Advanced*

Determines whether or not to allow allocating memory for the cache on heap.

If `false`, then caches will still be allocated off-heap, but the additional free memory inside the JVM will not be allocated for the caches.

Use this to have better control over the heap memory.

Default: `false`

`--high-io[=<true/false>]`

Ignore environment-based heuristics, and specify whether the target storage subsystem can support parallel IO with high throughput.

Typically this is `true` for SSDs, large raid arrays and network-attached storage.

Default: `false`

`--ignore-empty-strings[=<true/false>]`

Determines whether or not empty string fields, such as `""`, from input source are ignored (treated as null).

Default: `false`

`--ignore-extra-columns[=<true/false>]`

If unspecified columns should be ignored during the import.

Default: `false`

`--legacy-style-quoting[=<true/false>]`

Determines whether or not backslash-escaped quote e.g. `\"` is interpreted as inner quote.

Default: `false`

`--multiline-fields[=<true/false>]`

Determines whether or not fields from input source can span multiple lines, i.e. contain newline characters.

Setting `--multiline-fields=true` can severely degrade performance of the importer. Therefore, use it with care, especially with large imports.

Default: `false`

`--normalize-types[=<true/false>]`

Determines whether or not to normalize property types to Cypher types, e.g. `int` becomes `long` and `float` becomes `double`.

Default: `true`

`--skip-bad-entries-logging[=<true/false>]`

Determines whether or not to skip logging bad entries detected during import.

Default: `false`

`--skip-bad-relationships[=<true/false>]`

Determines whether or not to skip importing relationships that refer to missing node IDs, i.e. either start or end node ID/group referring to node that was not specified by the node input data.

Skipped relationships will be logged, containing at most the number of entities specified by `--bad-tolerance`, unless otherwise specified by the `--skip-bad-entries-logging` option.

Default: `false`

`--skip-duplicate-nodes[=<true/false>]`

Determines whether or not to skip importing nodes that have the same ID/group.

In the event of multiple nodes within the same group having the same ID, the first encountered will be imported, whereas consecutive such nodes will be skipped.

Skipped nodes will be logged, containing at most the number of entities specified by `--bad-tolerance`, unless otherwise specified by the `--skip-bad-entries-logging` option.

Default: `false`

`--trim-strings[=<true/false>]`

Determines whether or not strings should be trimmed for whitespaces.

Default: `false`

`--additional-config=<config-file-path>`

Path to a configuration file that contain additional configuration options.

`--array-delimiter=<char>`

Determines the array delimiter within a value in CSV data.

- ASCII character — e.g. `--array-delimiter=";"`.
- `\ID` — unicode character with ID, e.g. `--array-delimiter="\59"`.
- `U+XXXX` — unicode character specified with 4 HEX characters, e.g. `--array-delimiter="U+20AC"`.
- `\t` — horizontal tabulation (HT), e.g. `--array-delimiter="\t"`.

For horizontal tabulation (HT), use `\t` or the unicode character ID `\9`.

Unicode character ID can be used if prepended by `\`.

Default: `;`

`--bad-tolerance=<num>`

Number of bad entries before the import is considered failed.

This tolerance threshold is about relationships referring to missing nodes. Format errors in input data are still treated as errors.

Default: `1000`

`--database=<name>`

Name of the database to import.

Default: `neo4j`

`--delimiter=<char>`

Determines the delimiter between values in CSV data.

- ASCII character — e.g. `--delimiter=","`.
- `\ID` — unicode character with ID, e.g. `--delimiter="\44"`.
- `U+XXXX` — unicode character specified with 4 HEX characters, e.g. `--delimiter="U+20AC"`.
- `\t` — horizontal tabulation (HT), e.g. `--delimiter="\t"`.

For horizontal tabulation (HT), use `\t` or the unicode character ID `\9`.

Unicode character ID can be used if prepended by `\`.

Default: `,`

`--id-type=<STRING|INTEGER|ACTUAL>`

Each node must provide a unique ID in order to be used for creating relationships during the import.

Possible values are:

- `STRING` — arbitrary strings for identifying nodes.
- `INTEGER` — arbitrary integer values for identifying nodes.
- `ACTUAL` — actual node IDs. (**Advanced**)

Default: `STRING`

`--input-encoding=<character-set>`

Character set that input data is encoded in.

Default: `UTF-8`

`--max-memory=<size>`

Maximum memory that `neo4j-admin` can use for various data structures and caching to improve performance.

Values can be plain numbers such as `10000000`, or `20G` for 20 gigabyte. It can also be specified as a percentage of the available memory, for example `70%`.

Default: `90%`

`--processors=<num>` *Advanced*

Max number of processors used by the importer.

Defaults to the number of available processors reported by the JVM. There is a certain amount of minimum threads needed, so for that reason there is no lower bound for this value.

For optimal performance, this value shouldn't be greater than the number of available processors.

`--quote=<char>`

Character to treat as quotation character for values in CSV data.

Quotes can be escaped as per RFC 4180 by doubling them, for example `""` would be interpreted as a literal `"`.

You cannot escape using `\`.

Default: `"`

`--read-buffer-size=<size>`

Size of each buffer for reading input data.

It has to at least be large enough to hold the biggest single value in the input data. Value can be a plain number or byte units string, e.g. `128k`, `1m`.

Default: `4m`

`--report-file=<filename>`

File in which to store the report of the csv-import.

Default: `import.report`

The location of the import log file can be controlled using the `--report-file` option. If you run large imports of CSV files that have low data quality, the import log file can grow very large. For example, CSV files that contain duplicate node IDs, or that attempt to create relationships between non-existent nodes, could be classed as having low data quality. In these cases, you may wish to direct the output to a location that can handle the large log file.

If you are running on a UNIX-like system and you are not interested in the output, you can get rid of it altogether by directing the report file to `/dev/null`.

If you need to debug the import, it might be useful to collect the stack trace. This is done by using `--verbose` option.

`--nodes=[<label>[:<label>]…=]<files>…`

Node CSV header and data.

- Multiple files will be logically seen as one big file from the perspective of the importer.
- The first line must contain the header.
- Multiple data sources like these can be specified in one import, where each data source has its own header.
- Files can also be specified using regular expressions.

For an example, see Import data from CSV files using regular expression.

`--relationships=[<type>=]<files>…`

Relationship CSV header and data.

- Multiple files will be logically seen as one big file from the perspective of the importer.
- The first line must contain the header.
- Multiple data sources like these can be specified in one import, where each data source has its own header.
- Files can also be specified using regular expressions.

For an example, see Import data from CSV files using regular expression.

`@<arguments-file-path>`

File containing all arguments, used as an alternative to supplying all arguments on the command line directly.

Each argument can be on a separate line, or multiple arguments per line and separated by space.

Arguments containing spaces must be quoted.

Heap size for the import

You want to set the maximum heap size to a relevant value for the import. This is done by defining the `HEAP_SIZE` environment parameter before starting the import. For example, 2G is an appropriate value for smaller imports.

If doing imports in the order of magnitude of 100 billion entities, 20G will be an appropriate value.

Record format

If your import data will result in a graph that is larger than 34 billion nodes, 34 billion relationships, or 68 billion properties you will need to configure the importer to use the high limit record format. This is achieved by setting the parameters `dbms.record_format=high_limit` and `dbms.allow_upgrade=true` in a configuration file, and supplying that file to the importer with `--additional-config`. The format is printed in the *debug.log* file.

The `high_limit` format is available for Enterprise Edition only.

16.6.3. CSV header format

The header file of each data source specifies how the data fields should be interpreted. You must use the same delimiter for the header file and for the data files.

The header contains information for each field, with the format `<name>:<field_type>`. The `<name>` is used for properties and node IDs. In all other cases, the `<name>` part of the field is ignored.

16.6.4. Node files

Files containing node data can have an `ID` field, a `LABEL` field as well as properties.

ID

Each node must have a unique ID if it is to be connected by any relationships created in the import. The IDs are used to find the correct nodes when creating relationships. Note that the ID has to be unique across all nodes in the import; even for nodes with different labels. The unique ID can be persisted in a property whose name is defined by the `<name>` part of the field definition `<name>:ID`. If no such property name is defined, the unique ID will be used for the purpose of the import but not be available for reference later. If no ID is specified, the node will be imported but it will not be able to be connected by any relationships during the import.

LABEL

Read one or more labels from this field. Like array values, multiple labels are separated by `;`, or by the character specified with `--array-delimiter`.

Example 110. 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
```

16.6.5. Relationship files

Files containing relationship data have three mandatory fields and can also have properties. The mandatory fields are:

TYPE

The relationship type to use for this relationship.

START_ID

The ID of the start node for this relationship.

END_ID

The ID of the end node for this relationship.

The `START_ID` and `END_ID` refer to the unique node ID defined in one of the node data sources, as explained in the previous section. None of these takes a name, e.g. if `<name>:START_ID` or `<name>:END_ID` is defined, the `<name>` part will be ignored.

Example 111. 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
```

16.6.6. Properties

For properties, the `<name>` part of the field designates the property key, while the `<field_type>` part assigns a data type (see below). You can have properties in both node data files and relationship data files.

Data types

Use one of `int`, `long`, `float`, `double`, `boolean`, `byte`, `short`, `char`, `string`, `point`, `date`, `localtime`, `time`, `localdatetime`, `datetime`, and `duration` to designate the data type for properties. If no data type is given, this defaults to `string`. To define an array type, append `[]` to the type. By default, array values

are separated by `;`. A different delimiter can be specified with `--array-delimiter`. Boolean values are *true* if they match exactly the text `true`. All other values are *false*. Values that contain the delimiter character need to be escaped by enclosing in double quotation marks, or by using a different delimiter character with the `--delimiter` option.

Example 112. Header format with data types

This example illustrates several different data types specified in the CSV header.

```
:ID,name,joined:date,active:boolean,points:int
user01,Joe Soap,2017-05-05,true,10
user02,Jane Doe,2017-08-21,true,15
user03,Moe Know,2018-02-17,false,7
```

Special considerations for the `point` data type

A point is specified using the Cypher syntax for maps. The map allows the same keys as the input to the Cypher Manual → Point function. The point data type in the header can be amended with a map of default values used for all values of that column, e.g. `point{crs: 'WGS-84'}`. Specifying the header this way allows you to have an incomplete map in the value position in the data file. Optionally, a value in a data file may override default values from the header.

Example 113. Property format for `point` data type

This example illustrates various ways of using the `point` data type in the import header and the data files.

We are going to import the name and location coordinates for cities. First, we define the header as:

```
:ID,name,location:point{crs:WGS-84}
```

We then define cities in the data file.

- The first city's location is defined using `latitude` and `longitude`, as expected when using the coordinate system defined in the header.
- The second city uses `x` and `y` instead. This would normally lead to a point using the coordinate reference system `cartesian`. Since the header defines `crs:WGS-84`, that coordinate reference system will be used.
- The third city overrides the coordinate reference system defined in the header, and sets it explicitly to `WGS-84-3D`.

```
:ID,name,location:point{crs:WGS-84}
city01,"Malmö","{latitude:55.6121514, longitude:12.9950357}"
city02,"London","{y:51.507222, x:-0.1275}"
city03,"San Mateo","{latitude:37.554167, longitude:-122.313056, height: 100, crs:'WGS-84-3D'}"
```

Note that all point maps are within double quotation marks `"` in order to prevent the enclosed `,` character from being interpreted as a column separator. An alternative approach would be to use `--delimiter='\t'` and reformat the file with tab separators, in which case the `"` characters are not required.

```
:ID name    location:point{crs:WGS-84}
city01  Malmö   {latitude:55.6121514, longitude:12.9950357}
city02  London  {y:51.507222, x:-0.1275}
city03  San Mateo   {latitude:37.554167, longitude:-122.313056, height: 100, crs:'WGS-84-3D'}
```

Special considerations for temporal data types

The format for all temporal data types must be defined as described in Cypher Manual → Durations syntax. Two of the temporal types, *Time* and *DateTime*, take a time zone parameter which might be common between all or many of the values in the data file. It is therefor possible to specify a default time zone for *Time* and *DateTime* values in the header, for example: `time{timezone:+02:00}` and: `datetime{timezone:Europe/Stockholm}`. If no default time zone is specified, the default timezone is determined by the `db.temporal.timezone` configuration setting. The default time zone can be explicitly overridden in the values in the data file.

Example 114. 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]
```

16.6.7. 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 115. 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
```

16.6.8. 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 116. 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`.

16.6.9. Import compressed files

The import tool can handle files compressed with `zip` or `gzip`. Each compressed file must contain a single file.

Example 117. 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
```

16.6.10. Resuming a stopped or cancelled import Enterprise edition

An import that is stopped or fails before completing can be resumed from a point closer to where it was stopped. An import can be resumed from the following points:

- Linking of relationships
- Post-processing

16.7. Unbind a Core Server

The cluster state of a Causal Cluster member can be removed by using the following command:

`neo4j-admin unbind`

Limitations

The Neo4j server process should be shutdown before running the `unbind` command.

Examples of usage

- When transforming a Causal Cluster member to a standalone server:

  The `unbind` command can be used to turn a Causal Cluster server into a standalone server. To start the database in single (standalone) mode after unbinding it from the cluster, first set `dbms.mode=SINGLE` in *neo4j.conf*.

- When seeding a Causal Cluster with existing store files:

  If you wish to seed a new Causal Cluster using the store files of a previous cluster, you must first run `neo4j-admin unbind` on each server. For more information about seeding Causal Clusters, see Seed a cluster.

If a cluster holds a previous version of any of the databases being seeded, you must `DROP` those databases before seeding. Alternatively, you can stop every instance, unbind them from the cluster using neo4j-admin unbind and then forcefully restore the correct seeds (backups) for the databases in question. If you do not `DROP` or `unbind` before seeding, either with `neo4j-admin restore` or `neo4j-admin load`, the database's store files and cluster state will be out of sync, potentially leading to logical corruptions.

- When recovering a Causal Cluster:

  In the event of serious failures you may need to recover an entire Causal Cluster from backups. Before restoring those backups, you must first run `neo4j-admin unbind` on each server. For more information about recovering databases from online backups, see Restore a database backup.

Unlike versions of Neo4j prior to v4.0.0, you **must** run the `unbind` command on both Read Replicas and Core members.

16.8. Push to Neo4j AuraDB

You can use the `neo4j-admin database upload` command to upload a local Neo4j database dump into a Neo4j Aura instance.

This operation is secured and TLS encrypted end to end.

16.8.1. Import database to Neo4j AuraDB Aura

The `neo4j-admin` tool is located in the `bin` directory. Run it with the `push-to-cloud` argument in order to push your local database to a Neo4j AuraDB instance.

Syntax

```
neo4j-admin push-to-cloud [--overwrite] [--verbose] --bolt-uri=<boltURI>
                          [--database=<database>] [--dump=<dump>]
                          [--dump-to=<tmpDumpFile>] [--password=<password>]
                          [--username=<username>]
```

Options

Option	Default	Description
`--database`	`neo4j`	Name of the database to push. This argument cannot be used together with `--dump`.
`--dump`		Path to an existing database dump for upload, in the format */path/to/my-neo4j-database-dump-file*. This argument cannot be used together with `--database`.

Option	Default	Description
`--dump-to`		Optional. Target path for temporary database dump file to be uploaded, in the format */path/to/temp-file*. Used in combination with the `--database` argument.
`--bolt-uri`		Bolt URI of the target database. For example, `neo4j://mydatabaseid.databases.neo4j.io`.
`--username`		Optional. Username of the target database to push this database to. If you do not provide a username, you will be prompted to provide one. Alternatively, the `NEO4J_USERNAME` environment variable can be used.
`--password`		Optional. Password of the target database to push this database to. If you do not provide a password, you will be prompted to provide one. Alternatively, the `NEO4J_PASSWORD` environment variable can be used.
`--overwrite`		Optional. Overwrite the data in the target database.
`--verbose`		Optional. Enable verbose output.

Limitations

- A Neo4j AuraDB database must already be provisioned and running.
- Your local database must be stopped before you run the `push-to-cloud` command with the `--database` argument. The `push-to-cloud` function cannot be used with a source database which is currently in use.
- If used with a running source database, it will exit and print an error.

Output

If the `push-to-cloud` function completes successfully, it will exit with the following log line:

```
“Your data was successfully pushed to cloud and is now running”.
```

If the `push-to-cloud` function encounters an error at any point, you will be provided with instructions on how to try again, or to contact Neo4j Aura support.

Example 118. Run the `push-to-cloud` command

Run with the `--database` option to dump a specific database name. Note that the local database must be stopped first.

```
$neo4j-home> bin/neo4j stop
$neo4j-home> bin/neo4j-admin push-to-cloud --database=graph.db --bolt-uri=<neo4j-cloud-bolt-uri>
Neo4j cloud database user name: neo4j
Neo4j cloud database password:
Dumped contents of database 'graph.db' into '$neo4j-home/dump-of-graph.db-1569931327069'
Upload
.....  10%
.....  20%
.....  30%
.....  40%
.....  50%
.....  60%
.....  70%
.....  80%
.....  90%
..... 100%
We have received your export and it is currently being loaded into your cloud instance.
You can wait here, or abort this command and head over to the console to be notified of when your
database is running.
Import status
.....  10%
.....  20%
.....  30%
.....  40%
.....  50%
.....  60%
.....  70%
.....  80%
.....  90%
..... 100%
Your data was successfully pushed to cloud and is now running.
```

16.9. Cypher Shell

16.9.1. About Cypher Shell CLI

Cypher Shell is a command-line tool that comes with the Neo4j installation. It can also be downloaded from Neo4j Download Center and installed separately.

Cypher Shell CLI is used to run queries and perform administrative tasks against a Neo4j instance. By default, the shell is interactive, but you can also use it for scripting, by passing cypher directly on the command line or by piping a file with cypher statements (requires PowerShell on Windows). It communicates via the Bolt protocol.

16.9.2. Syntax

The Cypher Shell CLI is located in the `bin` directory if installed as part of the product.

The syntax is:

```
cypher-shell [-u USERNAME, --username USERNAME]
             [cypher]
             [-h, --help]
             [--fail-fast]
             [--fail-at-end]
             [--format]
             [--debug]
             [--non-interactive]
             [-v, --version]
             [-a ADDRESS, --address ADDRESS]
             [-p PASSWORD, --password PASSWORD]
             [--encryption]
             [-d DATABASE, --database DATABASE]
             [--P PARAM, --param PARAM]
             [--sample-rows SAMPLE-ROWS]
             [--wrap]
             [--driver-version]
             [-f FILE, --file FILE]
```

Arguments

Argument	Type	Description	Default value
`-u USERNAME`, `--username USERNAME`	Connection argument	Username to connect as. It can also be specified by the environment variable `NEO4J_USERNAME`.	
`cypher`	Positional argument	An optional string of cypher to execute and then exit.	
`-h`, `--help`	Optional argument	Show help message and exit.	
`--fail-fast`	Optional argument	Exit and report failure on first error when reading from file.	This is the default behavior.
`--fail-at-end`	Optional argument	Exit and report failures at end of input when reading from file.	
`--format {auto,verbose,plain}`	Optional argument	Desired output format.	`auto` (default) displays results in tabular format if you use the shell interactively and with minimal formatting if you use it for scripting. `verbose` displays results in tabular format and prints statistics. `plain` displays data with minimal formatting.
`--debug`	Optional argument	Print additional debug information.	`false`
`--non-interactive`	Optional argument	Force non-interactive mode; only useful if auto-detection fails (e.g. Windows).	`false`

Argument	Type	Description	Default value
`-v`, `--version`	Optional argument	Print version of cypher-shell and exit.	`false`
`-a ADDRESS`, `--address ADDRESS`	Connection argument	Address and port to connect to.	`neo4j://localhost:7687`
`-p PASSWORD`, `--password PASSWORD`	Connection argument	Password to connect with. It can also be specified by the environment variable `NEO4J_PASSWORD`.	
`--encryption {true,false,default}`	Connection argument	Whether the connection to Neo4j should be encrypted; must be consistent with Neo4j's configuration.	`default` - the encryption setting is deduced from the specified address. For example, the `neo4j+ssc` protocol would use encryption.
`-d DATABASE`, `--database DATABASE`	Connection argument	Database to connect to. It can also be specified by the environment variable `NEO4J_DATABASE`.	
`--P PARAM`, `--param PARAM`	Optional argument	Add a parameter to this session. For example, `-P "number ⇒ 3"` or `-P "country ⇒ 'Spain'"`. This argument can be specified multiple times.	
`--sample-rows SAMPLE-ROWS`	Optional argument	Number of rows sampled to compute table widths (only for format=`VERBOSE`).	`1000`
`--wrap {true,false}`	Optional argument	Wrap table column values if column is too narrow (only for format=`VERBOSE`).	`true`
`--driver-version`	Optional argument	Print version of the Neo4j Driver used and exit.	`false`
`-f FILE`, `--file FILE`	Optional argument	Pass a file with cypher statements to be executed. After the statements have been executed cypher-shell shuts down.	

Example 119. Invoke `cypher-shell` with username and password

```
$neo4j-home> bin/cypher-shell -u neo4j -p <password>
```

```
Connected to Neo4j at neo4j://localhost:7687 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
neo4j>
```

Example 120. Invoke `help`

```
neo4j> :help
```

```
Available commands:
  :begin    Open a transaction
  :commit   Commit the currently open transaction
  :exit     Exit the logger
  :help     Show this help message
  :history  Print a list of the last commands executed
  :param    Set the value of a query parameter
  :params   Prints all currently set query parameters and their values
  :rollback Rollback the currently open transaction
  :source   Interactively executes Cypher statements from a file
  :use      Set the active database

For help on a specific command type:
  :help command
```

Example 121. Execute a query

```
neo4j> MATCH (n) RETURN n;
```

```
+-------------------------------------------------------------------+
| n                                                                 |
+-------------------------------------------------------------------+
| (:Person {name: "Bruce Wayne", alias: "Batman"})                  |
| (:Person {name: "Selina Kyle", alias: ["Catwoman", "The Cat"]}) |
+-------------------------------------------------------------------+
```

Example 122. Invoke `cypher-shell` with a Cypher script

The contents of a file called `examples.cypher`:

```
MATCH (n) RETURN n;

MATCH (batman:Person {name: 'Bruce Wayne'}) RETURN batman;
```

Invoke the `examples.cypher` script from the command-line. All the examples in the remainder of this section use the `--format plain` flag for a simple output.

Using `cat` (UNIX)

```
$neo4j-home> cat examples.cypher | bin/cypher-shell -u neo4j -p <password> --format plain
```

Using `type` (Windows)

```
$neo4j-home> type examples.cypher | bin/cypher-shell.bat -u neo4j -p <password> --format plain
```

Result

```
n
(:Person {name: "Bruce Wayne", alias: "Batman"})
(:Person {name: "Selina Kyle", alias: ["Catwoman", "The Cat"]})
batman
(:Person {name: "Bruce Wayne", alias: "Batman"})
```

16.9.3. Query parameters

Cypher Shell CLI supports querying based on parameters. This is often used while scripting.

Example 123. 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
```

16.9.4. Transactions

Cypher Shell supports explicit transactions. Transaction states are controlled using the keywords `:begin`, `:commit`, and `:rollback`.

Example 124. Use fine-grained transaction control

Start a transaction in your first Cypher Shell session:

```
neo4j> MATCH (n) RETURN n;
+--------------------------------------------------------------------+
| n                                                                  |
+--------------------------------------------------------------------+
| (:Person {name: "Bruce Wayne", alias: "Batman"})                   |
| (:Person {name: "Selina Kyle", alias: ["Catwoman", "The Cat"]}) |
| (:Person {name: "Dick Grayson", alias: "Robin"})                   |
+--------------------------------------------------------------------+
3 rows available after 2 ms, consumed after another 2 ms
neo4j> :begin
neo4j# CREATE (:Person {name : 'Edward Mygma', alias : 'The Riddler' });
```

If you open a second Cypher Shell session, you will notice no changes from the latest `CREATE` statement.

```
neo4j> MATCH (n) RETURN n;
+--------------------------------------------------------------------+
| n                                                                  |
+--------------------------------------------------------------------+
| (:Person {name: "Bruce Wayne", alias: "Batman"})                   |
| (:Person {name: "Selina Kyle", alias: ["Catwoman", "The Cat"]}) |
| (:Person {name: "Dick Grayson", alias: "Robin"})                   |
+--------------------------------------------------------------------+
3 rows available after 2 ms, consumed after another 2 ms
```

Go back to the first session and commit the transaction.

```
neo4j# :commit
0 rows available after 1 ms, consumed after another 0 ms
Added 1 nodes, Set 2 properties, Added 1 labels
neo4j> MATCH (n) RETURN n;
+--------------------------------------------------------------------+
| n                                                                  |
+--------------------------------------------------------------------+
| (:Person {name: "Bruce Wayne", alias: "Batman"})                   |
| (:Person {name: "Selina Kyle", alias: ["Catwoman", "The Cat"]}) |
| (:Person {name: "Dick Grayson", alias: "Robin"})                   |
| (:Person {name: "Edward Mygma", alias: "The Riddler"})             |
+--------------------------------------------------------------------+
4 rows available after 1 ms, consumed after another 1 ms

neo4j>
```

16.9.5. Procedures

Cypher Shell supports running any procedures for which the current user is authorized.

Example 125. Call the `dbms.showCurrentUser` procedure

```
neo4j> CALL dbms.showCurrentUser();
+------------------------------+
| username | roles     | flags |
+------------------------------+
| "neo4j"  | ["admin"] | []    |
+------------------------------+

1 row available after 66 ms, consumed after another 2 ms
neo4j> :exit
```

16.9.6. Supported operating systems

You can use the Cypher Shell CLI via `cmd` on Windows systems, and `bash` on Unix systems.

Other shells may work as intended, but there is no test coverage to guarantee compatibility.

Appendix A: Reference

This appendix contains the following:

- Configuration settings
- Procedures

16.A.1. Configuration settings

Configuration settings can be set in *neo4j.conf*. Refer to The neo4j.conf file for details on how to use configuration settings.

All settings

- 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.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.routing.getRoutingTable()` procedure should include followers as read endpoints or return only read replicas.
- causal_clustering.cluster_allow_reads_on_leader: Configure if the `dbms.routing.getRoutingTable()`

procedure should include the leader as read endpoint or return only read replicas/followers.

- causal_clustering.cluster_binding_timeout: The time allowed for a database on a Neo4j server to either join a cluster or form a new cluster with the other Neo4j Core Servers provided by `causal_clustering.initial_discovery_members`.
- causal_clustering.cluster_topology_refresh: Time between scanning the cluster to refresh current server's view of topology.
- causal_clustering.command_applier_parallelism: Limits amount of global threads for applying commands.
- 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.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.election_failure_detection_window: The rate at which leader elections happen.
- 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.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.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_max_lag: Maximum amount of lag accepted for a new follower to join the Raft group.
- 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.cluster_domain: Kubernetes cluster domain.
- 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.last_applied_state_size: The maximum file size before the storage file is rotated (in unit of entries).

- causal_clustering.leader_election_timeout: This setting is moved and enhanced into causal_clustering.leader_failure_detection_window and causal_clustering.election_failure_detection_window.
- causal_clustering.leader_failure_detection_window: The time window within which the loss of the leader is detected and the first re-election attempt is held.The window should be significantly larger than typical communication delays to make conflicts unlikely.
- causal_clustering.leadership_balancing: Which strategy to use when transferring database leaderships around a cluster.
- causal_clustering.leadership_priority_group: The name of a server_group whose members should be prioritized as leaders.
- 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.log_shipping_retry_timeout: Retry time for log shipping to followers after a stall.
- causal_clustering.middleware.logging.level: The level of middleware logging.
- causal_clustering.minimum_core_cluster_size_at_formation: Minimum number of Core machines initially required to form a cluster.
- causal_clustering.minimum_core_cluster_size_at_runtime: The minimum size of the dynamically adjusted voting set (which only core members may be a part of).
- causal_clustering.multi_dc_license: Enable multi-data center features.
- 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_handler_parallelism: Limits amount of global threads shared by raft groups for handling bathing of messages and timeout events.
- 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_log_implementation: 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.refuse_to_be_leader: Prevents the current instance from volunteering to become Raft leader.
- causal_clustering.replicated_lease_state_size: The maximum file size before the replicated lease state file is rotated (in unit of entries).
- causal_clustering.replication_leader_await_timeout: The duration for which the replicator will await a new leader.
- 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.server_groups: A list of group names for the server used when configuring load balancing and replication policies.
- 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.status_throughput_window: Sampling window for throughput estimate reported in the status endpoint.
- causal_clustering.store_copy_chunk_size: Store copy chunk size.
- causal_clustering.store_copy_max_retry_time_per_request: Maximum retry time per request during store copy.
- 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 for statistics above which a plan is considered stale. If any of the underlying statistics used to create the plan have changed more than this value, the plan will be considered stale and will be replanned.
- db.temporal.timezone: Database timezone for temporal functions.
- dbms.allow_single_automatic_upgrade: Whether to allow a system graph upgrade to happen automatically in single instance mode (dbms.mode=SINGLE).
- dbms.allow_upgrade: Whether to allow a store upgrade in case the current version of the database starts against an older version of the store.
- dbms.backup.enabled: Enable support for running online backups.
- dbms.backup.listen_address: Network interface and port for the backup server to listen on.
- 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., cypher., etc).
- dbms.connector.bolt.advertised_address: Advertised address for this connector.
- dbms.connector.bolt.enabled: Enable the bolt connector.
- dbms.connector.bolt.listen_address: Address the connector should bind to.
- dbms.connector.bolt.ocsp_stapling_enabled: Enable server OCSP stapling for bolt and http connectors.
- dbms.connector.bolt.thread_pool_keep_alive: The maximum time an idle thread in the thread pool bound to this connector will wait for new tasks.
- dbms.connector.bolt.thread_pool_max_size: The maximum number of threads allowed in the thread pool bound to this connector.
- dbms.connector.bolt.thread_pool_min_size: The number of threads to keep in the thread pool bound to this connector, even if they are idle.
- dbms.connector.bolt.tls_level: Encryption level to require this connector to use.

- dbms.connector.bolt.unsupported_thread_pool_shutdown_wait_time: The maximum time to wait for the thread pool to finish processing its pending jobs and shutdown.
- dbms.connector.http.advertised_address: Advertised address for this connector.
- dbms.connector.http.enabled: Enable the http connector.
- dbms.connector.http.listen_address: Address the connector should bind to.
- dbms.connector.https.advertised_address: Advertised address for this connector.
- dbms.connector.https.enabled: Enable the https connector.
- dbms.connector.https.listen_address: Address the connector should bind to.
- dbms.db.timezone: Database timezone.
- dbms.default_advertised_address: Default hostname or IP address the server uses to advertise itself.
- dbms.default_database: Name of the default database.
- dbms.default_listen_address: Default network interface to listen for incoming connections.
- dbms.directories.cluster_state: Directory to hold cluster state including Raft log.
- dbms.directories.data: Path of the data directory.
- dbms.directories.dumps.root: Root location where Neo4j will store database dumps optionally produced when dropping said databases.
- 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.neo4j_home: Root relative to which directory settings are resolved.
- dbms.directories.plugins: Location of the database plugin directory.
- dbms.directories.run: Path of the run directory.
- dbms.directories.script.root: Root location where Neo4j will store scripts for configured databases.
- dbms.directories.transaction.logs.root: Root location where Neo4j will store transaction logs for configured databases.
- dbms.dynamic.setting.allowlist: A list of setting name patterns (comma separated) that are allowed to be dynamically changed.
- dbms.dynamic.setting.whitelist: A list of setting name patterns (comma separated) that are allowed to be dynamically changed.
- dbms.filewatcher.enabled: Allows the enabling or disabling of the file watcher service.
- dbms.http_enabled_modules: Defines the set of modules loaded into the Neo4j web server.
- dbms.import.csv.buffer_size: The size of the internal buffer in bytes used by `LOAD CSV`.
- dbms.import.csv.legacy_quote_escaping: 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.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.early_raw_logging_enabled: Log query text and parameters without obfuscating passwords.
- dbms.logs.query.enabled: Log executed queries.
- dbms.logs.query.page_logging_enabled: Log page hits and page faults for the executed queries being logged.
- dbms.logs.query.parameter_full_entities: Log complete parameter entities including id, labels or

relationship type, and properties.

- dbms.logs.query.parameter_logging_enabled: Log parameters for the executed queries being logged.
- dbms.logs.query.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 once completed - provided query logging is set to INFO.
- dbms.logs.query.time_logging_enabled: Log detailed time information for the executed queries being logged, such as `(planning: 92, waiting: 0)`.
- 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.user.path: Path to the user log file.
- dbms.logs.user.rotation.delay: Minimum time interval after last rotation of the user log (*neo4j.log*) before it may be rotated again.
- dbms.logs.user.rotation.keep_number: Maximum number of history files for the user log (*neo4j.log*).
- dbms.logs.user.rotation.size: Threshold for rotation of the user log (*neo4j.log*).
- dbms.logs.user.stdout_enabled: Send user logs to the process stdout.
- dbms.max_databases: The maximum number of databases.
- dbms.memory.heap.initial_size: Initial heap size.
- dbms.memory.heap.max_size: Maximum heap size.
- dbms.memory.off_heap.block_cache_size: Defines the size of the off-heap memory blocks cache.
- dbms.memory.off_heap.max_cacheable_block_size: Defines the maximum size of an off-heap memory block that can be cached to speed up allocations.
- dbms.memory.off_heap.max_size: 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.memory.pagecache.directio: Use direct I/O for page cache.
- dbms.memory.pagecache.flush.buffer.enabled: Page cache can be configured to use a temporal buffer for flushing purposes.
- dbms.memory.pagecache.flush.buffer.size_in_pages: Page cache can be configured to use a temporal buffer for flushing purposes.
- dbms.memory.pagecache.scan.prefetchers: The maximum number of worker threads to use for pre-fetching data when doing sequential scans.

- 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: This setting is not used anymore.
- dbms.memory.pagecache.warmup.enable: Page cache can be configured to perform usage sampling of loaded pages that can be used to construct active load profile.
- dbms.memory.pagecache.warmup.preload: Page cache warmup can be configured to prefetch files, preferably when cache size is bigger than store size.
- dbms.memory.pagecache.warmup.preload.allowlist: Page cache warmup prefetch file allowlist regex.
- dbms.memory.pagecache.warmup.preload.whitelist: Page cache warmup prefetch file whitelist regex.
- dbms.memory.pagecache.warmup.profile.interval: The profiling frequency for the page cache.
- dbms.memory.tracking.enable: Enable off heap and on heap memory tracking.
- dbms.memory.transaction.database_max_size: Limit the amount of memory that all transactions in one database can consume, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g').
- dbms.memory.transaction.global_max_size: Limit the amount of memory that all of the running transactions can consume, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g').
- dbms.memory.transaction.max_size: Limit the amount of memory that a single transaction can consume, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g').
- dbms.mode: Configure the operating mode of the database — 'SINGLE' for stand-alone operation, 'CORE' for operating as a core member of a Causal Cluster, or 'READ_REPLICA' for operating as a read replica member of a Causal Cluster.
- dbms.netty.ssl.provider: Netty SSL provider.
- 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.reconciler.max_backoff: Defines the maximum amount of time to wait before retrying after the dbms fails to reconcile a database to its desired state.
- dbms.reconciler.max_parallelism: Defines the level of parallelism employed by the reconciler.
- dbms.reconciler.may_retry: Defines whether the dbms may retry reconciling a database to its desired state.
- dbms.reconciler.min_backoff: Defines the minimum amount of time to wait before retrying after the dbms fails to reconcile a database to its desired state.
- dbms.record_format: Database record format.
- dbms.recovery.fail_on_missing_files: If `true`, Neo4j will abort recovery if transaction log files are missing.
- 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.routing.advertised_address: The advertised address for the intra-cluster routing connector.

- dbms.routing.driver.api: Determines which driver API will be used.
- dbms.routing.driver.connection.connect_timeout: Socket connection timeout. A timeout of zero is treated as an infinite timeout and will be bound by the timeout configured on the operating system level.
- dbms.routing.driver.connection.max_lifetime: Pooled connections older than this threshold will be closed and removed from the pool. Setting this option to a low value will cause a high connection churn and might result in a performance hit. It is recommended to set maximum lifetime to a slightly smaller value than the one configured in network equipment (load balancer, proxy, firewall, etc.
- dbms.routing.driver.connection.pool.acquisition_timeout: Maximum amount of time spent attempting to acquire a connection from the connection pool. This timeout only kicks in when all existing connections are being used and no new connections can be created because maximum connection pool size has been reached. Error is raised when connection can't be acquired within configured time. Negative values are allowed and result in unlimited acquisition timeout.
- dbms.routing.driver.connection.pool.idle_test: Pooled connections that have been idle in the pool for longer than this timeout will be tested before they are used again, to ensure they are still alive. If this option is set too low, an additional network call will be incurred when acquiring a connection, which causes a performance hit. If this is set high, no longer live connections might be used which might lead to errors. Hence, this parameter tunes a balance between the likelihood of experiencing connection problems and performance Normally, this parameter should not need tuning. Value 0 means connections will always be tested for validity.
- dbms.routing.driver.connection.pool.max_size: Maximum total number of connections to be managed by a connection pool. The limit is enforced for a combination of a host and user.
- dbms.routing.driver.logging.level: Sets level for driver internal logging.
- dbms.routing.enabled: Enable intra-cluster routing using an additional bolt connector.
- dbms.routing.listen_address: The address the routing connector should bind to.
- dbms.routing_ttl: How long callers should cache the response of the routing procedure `dbms.routing.getRoutingTable()`.
- 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, as defined by `dbms.security.auth_lock_time`.The locked out user will not be able to log in until the lock period

expires, even if correct credentials are provided.

- dbms.security.authentication_providers: A list of security authentication providers containing the users and roles.
- dbms.security.authorization_providers: A list of security authorization providers containing the users and roles.
- dbms.security.causal_clustering_status_auth_enabled: Require authorization for access to the Causal Clustering 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_auth_allowlist: Defines an allowlist of http paths where Neo4j authentication is not required.
- dbms.security.http_auth_whitelist: Defines a whitelist of http paths where Neo4j authentication is not required.
- 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.allowlist: A list of procedures (comma separated) that are to be loaded.
- 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.
- 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.threads.worker_count: Number of Neo4j worker threads.
- 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.concurrent.maximum: The maximum number of concurrently running transactions.
- dbms.transaction.monitor.check.interval: Configures the time interval between transaction monitor checks.
- dbms.transaction.sampling.percentage: Transaction sampling percentage.
- dbms.transaction.timeout: The maximum time interval of a transaction within which it should be completed.
- dbms.transaction.tracing.level: Transaction creation tracing level.
- dbms.tx_log.preallocate: Specify if Neo4j should try to preallocate the logical log file in advance. It optimizes the filesystem by ensuring there is room to accommodate newly generated files and avoid file-level fragmentation.
- dbms.tx_log.rotation.retention_policy: Tell Neo4j how long logical transaction logs should be kept to backup the database.For example, "10 days" will prune logical logs that only contain transactions older than 10 days.Alternatively, "100k txs" will keep the 100k latest transactions from each database and prune any older transactions.

- dbms.tx_log.rotation.size: Specifies at which file size the logical log will auto-rotate.
- dbms.tx_state.memory_allocation: Defines whether memory for transaction state should be allocated on- or off-heap.
- dbms.unmanaged_extension_classes: Comma-separated list of <classname>=<mount point> for unmanaged extensions.
- dbms.upgrade_max_processors: Max number of processors used when upgrading the store.
- dbms.windows_service_name: Name of the Windows Service.
- fabric.database.name: Name of the Fabric database.
- fabric.driver.api: Determines which driver API will be used.
- fabric.driver.connection.connect_timeout: Socket connection timeout. A timeout of zero is treated as an infinite timeout and will be bound by the timeout configured on the operating system level.
- fabric.driver.connection.max_lifetime: Pooled connections older than this threshold will be closed and removed from the pool. Setting this option to a low value will cause a high connection churn and might result in a performance hit. It is recommended to set maximum lifetime to a slightly smaller value than the one configured in network equipment (load balancer, proxy, firewall, etc.
- fabric.driver.connection.pool.acquisition_timeout: Maximum amount of time spent attempting to acquire a connection from the connection pool. This timeout only kicks in when all existing connections are being used and no new connections can be created because maximum connection pool size has been reached. Error is raised when connection can't be acquired within configured time. Negative values are allowed and result in unlimited acquisition timeout.
- fabric.driver.connection.pool.idle_test: Pooled connections that have been idle in the pool for longer than this timeout will be tested before they are used again, to ensure they are still alive. If this option is set too low, an additional network call will be incurred when acquiring a connection, which causes a performance hit. If this is set high, no longer live connections might be used which might lead to errors. Hence, this parameter tunes a balance between the likelihood of experiencing connection problems and performance Normally, this parameter should not need tuning. Value 0 means connections will always be tested for validity.
- fabric.driver.connection.pool.max_size: Maximum total number of connections to be managed by a connection pool. The limit is enforced for a combination of a host and user.
- fabric.driver.logging.level: Sets level for driver internal logging.
- fabric.routing.servers: A comma-separated list of Fabric instances that form a routing group.
- fabric.routing.ttl: The time to live (TTL) of a routing table for fabric routing group.
- fabric.stream.buffer.low_watermark: Number of records in prefetching buffer that will trigger prefetching again.
- fabric.stream.buffer.size: Maximal size of a buffer used for pre-fetching result records of remote queries. To compensate for latency to remote databases, the Fabric execution engine pre-fetches records needed for local executions. This limit is enforced per fabric query.
- fabric.stream.concurrency: Maximal concurrency within Fabric queries. Limits the number of iterations of each subquery that are executed concurrently.
- 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.compression: Decides what compression to use for the csv history 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: Enable metrics.
- metrics.filter: Specifies which metrics should be enabled by using a comma separated list of globbing patterns.
- 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.jmx.enabled: Set to `true` to enable the JMX metrics endpoint.
- metrics.jvm.buffers.enabled: Enable reporting metrics about the buffer pools.
- metrics.jvm.file.descriptors.enabled: Enable reporting metrics about the number of open file descriptors.
- metrics.jvm.gc.enabled: Enable reporting metrics about the duration of garbage collections.
- metrics.jvm.heap.enabled: Enable reporting metrics about the heap memory usage.
- metrics.jvm.memory.enabled: Enable reporting metrics about the memory usage.
- metrics.jvm.pause_time.enabled: Enable reporting metrics about the VM pause time.
- metrics.jvm.threads.enabled: Enable reporting metrics about the current number of threads running.
- metrics.namespaces.enabled: Enable metrics namespaces that separates the global and database specific metrics.
- metrics.neo4j.causal_clustering.enabled: Enable reporting metrics about Causal Clustering mode.
- metrics.neo4j.checkpointing.enabled: Enable reporting metrics about Neo4j check pointing; when it occurs and how much time it takes to complete.
- metrics.neo4j.counts.enabled: Enable reporting metrics about approximately how many entities are in the database; nodes, relationships, properties, etc.
- metrics.neo4j.data.counts.enabled: Enable reporting metrics about number of entities in the database.
- metrics.neo4j.database_operation_count.enabled: Enable reporting metrics for Neo4j dbms operations; how many times databases have been created, started, stopped or dropped, and how many attempted operations have failed and recovered later.
- metrics.neo4j.logs.enabled: Enable reporting metrics about the Neo4j transaction logs.
- metrics.neo4j.pagecache.enabled: Enable reporting metrics about the Neo4j page cache; page faults, evictions, flushes, exceptions, etc.
- metrics.neo4j.pools.enabled: Enable reporting metrics about Neo4j memory pools.

- metrics.neo4j.server.enabled: Enable reporting metrics about Server threading info.
- metrics.neo4j.size.enabled: Enable reporting metrics about the store size of each database.
- metrics.neo4j.tx.enabled: Enable reporting metrics about transactions; number of transactions started, committed, etc.
- 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.

Table 85. browser.allow_outgoing_connections

Description	Configure the policy for outgoing Neo4j Browser connections.
Valid values	browser.allow_outgoing_connections, a boolean
Default value	`true`

Table 86. browser.credential_timeout

Description	Configure the Neo4j Browser to time out logged in users after this idle period. Setting this to 0 indicates no limit.
Valid values	browser.credential_timeout, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`0s`

Table 87. browser.post_connect_cmd

Description	Commands to be run when Neo4j Browser successfully connects to this server. Separate multiple commands with semi-colon.
Valid values	browser.post_connect_cmd, a string
Default value	

Table 88. browser.remote_content_hostname_whitelist

Description	Whitelist of hosts for the Neo4j Browser to be allowed to fetch content from.
Valid values	browser.remote_content_hostname_whitelist, a string
Default value	`guides.neo4j.com,localhost`

Table 89. browser.retain_connection_credentials

Description	Configure the Neo4j Browser to store or not store user credentials.
Valid values	browser.retain_connection_credentials, a boolean
Default value	`true`

Table 90. causal_clustering.catch_up_client_inactivity_timeout

Description	The catch up protocol times out if the given duration elapses with no network activity. Every message received by the client from the server extends the time out duration.
Valid values	causal_clustering.catch_up_client_inactivity_timeout, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`10m`

Table 91. causal_clustering.catchup_batch_size

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

Table 92. causal_clustering.cluster_allow_reads_on_followers

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

Table 93. causal_clustering.cluster_allow_reads_on_leader

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

Dynamic	true
Default value	`false`

Table 94. causal_clustering.cluster_binding_timeout

Description	The time allowed for a database on a Neo4j server to either join a cluster or form a new cluster with the other Neo4j Core Servers provided by `causal_clustering.initial_discovery_members`.
Valid values	causal_clustering.cluster_binding_timeout, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`5m`

Table 95. causal_clustering.cluster_topology_refresh

Description	Time between scanning the cluster to refresh current server's view of topology.
Valid values	causal_clustering.cluster_topology_refresh, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's') which is minimum `1s`
Default value	`5s`

Table 96. causal_clustering.command_applier_parallelism

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

Table 97. causal_clustering.connect_randomly_to_server_group

Description	Comma separated list of groups to be used by the connect-randomly-to-server-group selection strategy. The connect-randomly-to-server-group strategy is used if the list of strategies (`causal_clustering.upstream_selection_strategy`) includes the value `connect-randomly-to-server-group`.
Valid values	causal_clustering.connect_randomly_to_server_group, a ',' separated list with elements of type 'a string identifying a Server Group'.
Dynamic	true
Default value	

Table 98. causal_clustering.discovery_advertised_address

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

Table 99. causal_clustering.discovery_listen_address

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

Table 100. causal_clustering.discovery_type

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

Table 101. causal_clustering.election_failure_detection_window

Description	The rate at which leader elections happen. Note that due to election conflicts it might take several attempts to find a leader. The window should be significantly larger than typical communication delays to make conflicts unlikely.
Valid values	causal_clustering.election_failure_detection_window, a duration-range <min-max> (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`3s-6s`

Table 102. causal_clustering.enable_pre_voting

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

Table 103. causal_clustering.global_session_tracker_state_size

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

Table 104. causal_clustering.handshake_timeout

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

Table 105. 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, a byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `TiB`, `TB`, `PiB`, `PB`, `EiB`, `EB`)
Default value	`2.00GiB`

Table 106. 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, an integer
Default value	`1024`

Table 107. causal_clustering.in_flight_cache.type

Description	Type of in-flight cache.
Valid values	causal_clustering.in_flight_cache.type, one of [NONE, CONSECUTIVE, UNBOUNDED]
Default value	`CONSECUTIVE`

Table 108. causal_clustering.initial_discovery_members

Description	A comma-separated list of other members of the cluster to join.
Valid values	causal_clustering.initial_discovery_members, a ',' separated list with elements of type 'a socket address'.

Table 109. causal_clustering.join_catch_up_max_lag

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

Table 110. causal_clustering.join_catch_up_timeout

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

Table 111. causal_clustering.kubernetes.address

Description	Address for Kubernetes API.
Valid values	causal_clustering.kubernetes.address, a socket address
Default value	`kubernetes.default.svc:443`

Table 112. causal_clustering.kubernetes.ca_crt

Description	File location of CA certificate for Kubernetes API.

Valid values	causal_clustering.kubernetes.ca_crt, a path
Default value	`/var/run/secrets/kubernetes.io/serviceaccount/ca.crt`

Table 113. causal_clustering.kubernetes.cluster_domain

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

Table 114. causal_clustering.kubernetes.label_selector

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

Table 115. causal_clustering.kubernetes.namespace

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

Table 116. causal_clustering.kubernetes.service_port_name

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

Table 117. causal_clustering.kubernetes.token

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

Table 118. causal_clustering.last_applied_state_size

Description	The maximum file size before the storage file is rotated (in unit of entries)

Valid values	causal_clustering.last_applied_state_size, an integer
Default value	`1000`

Table 119. causal_clustering.leader_election_timeout

Description	This setting is moved and enhanced into causal_clustering.leader_failure_detection_window and causal_clustering.election_failure_detection_window.
Valid values	causal_clustering.leader_election_timeout, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`7s`
Deprecated	The `causal_clustering.leader_election_timeout` configuration setting has been deprecated.

Table 120. causal_clustering.leader_failure_detection_window

Description	The time window within which the loss of the leader is detected and the first re-election attempt is held.The window should be significantly larger than typical communication delays to make conflicts unlikely.
Valid values	causal_clustering.leader_failure_detection_window, a duration-range <min-max> (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`20s-23s`

Table 121. causal_clustering.leadership_balancing

Description	Which strategy to use when transferring database leaderships around a cluster. This can be one of `equal_balancing` or `no_balancing`. `equal_balancing` automatically ensures that each Core server holds the leader role for an equal number of databases.`no_balancing` prevents any automatic balancing of the leader role.Note that if a `leadership_priority_group` is specified for a given database, the value of this setting will be ignored for that database.
Valid values	causal_clustering.leadership_balancing, one of [NO_BALANCING, EQUAL_BALANCING]
Default value	`EQUAL_BALANCING`

Table 122. causal_clustering.leadership_priority_group

Description	The name of a server_group whose members should be prioritized as leaders. This does not guarantee that members of this group will be leader at all times, but the cluster will attempt to transfer leadership to such a member when possible. If a database is specified using `causal_clustering.leadership_priority_group`.<database> the specified priority group will apply to that database only. If no database is specified that group will be the default and apply to all databases which have no priority group explicitly set. Using this setting will disable leadership balancing.
Valid values	causal_clustering.leadership_priority_group, a string identifying a Server Group
Default value	

Table 123. causal_clustering.load_balancing.plugin

Description	The load balancing plugin to use.
Valid values	causal_clustering.load_balancing.plugin, a string which depends on dbms.mode. If dbms.mode is `CORE` or is `READ_REPLICA` then it specified load balancer plugin exist. otherwise it is unconstrained.
Default value	`server_policies`

Table 124. causal_clustering.load_balancing.shuffle

Description	Enables shuffling of the returned load balancing result.
Valid values	causal_clustering.load_balancing.shuffle, a boolean
Default value	`true`

Table 125. causal_clustering.log_shipping_max_lag

Description	The maximum lag allowed before log shipping pauses (in unit of entries)
Valid values	causal_clustering.log_shipping_max_lag, an integer
Default value	`256`

Table 126. causal_clustering.log_shipping_retry_timeout

Description	Retry time for log shipping to followers after a stall.
Valid values	causal_clustering.log_shipping_retry_timeout, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')

Default value	`5s`

Table 127. causal_clustering.middleware.logging.level

Description	The level of middleware logging.
Valid values	causal_clustering.middleware.logging.level, one of [DEBUG, INFO, WARN, ERROR, NONE]
Default value	`WARN`

Table 128. causal_clustering.minimum_core_cluster_size_at_formation

Description	Minimum number of Core machines initially required to form a cluster. The cluster will form when at least this many Core members have discovered each other.
Valid values	causal_clustering.minimum_core_cluster_size_at_formation, an integer which is minimum `2`
Default value	`3`

Table 129. causal_clustering.minimum_core_cluster_size_at_runtime

Description	The minimum size of the dynamically adjusted voting set (which only core members may be a part of). Adjustments to the voting set happen automatically as the availability of core members changes, due to explicit operations such as starting or stopping a member, or unintended issues such as network partitions. Note that this dynamic scaling of the voting set is generally desirable as under some circumstances it can increase the number of instance failures which may be tolerated. A majority of the voting set must be available before voting in or out members.
Valid values	causal_clustering.minimum_core_cluster_size_at_runtime, an integer which is minimum `2` and depends on dbms.mode. If dbms.mode is `CORE` or is `READ_REPLICA` then it Must be set less than or equal to value of 'causal_clustering.minimum_core_cluster_size_at_formation' otherwise it is unconstrained.
Default value	`3`

Table 130. causal_clustering.multi_dc_license

Description	Enable multi-data center features. Requires appropriate licensing.
Valid values	causal_clustering.multi_dc_license, a boolean
Default value	`false`

Table 131. causal_clustering.protocol_implementations.catchup

Description	Catchup protocol implementation versions that this instance will allow in negotiation as a comma-separated list. Order is not relevant: the greatest value will be preferred. An empty list will allow all supported versions. Example value: "1.1, 1.2, 2.1, 2.2"
Valid values	causal_clustering.protocol_implementations.catchup, a ',' separated list with elements of type 'an application protocol version'.
Default value	

Table 132. causal_clustering.protocol_implementations.compression

Description	Network compression algorithms that this instance will allow in negotiation as a comma-separated list. Listed in descending order of preference for incoming connections. An empty list implies no compression. For outgoing connections this merely specifies the allowed set of algorithms and the preference of the remote peer will be used for making the decision. Allowable values: [Gzip, Snappy, Snappy_validating, LZ4, LZ4_high_compression, LZ_validating, LZ4_high_compression_validating]
Valid values	causal_clustering.protocol_implementations.compression, a ',' separated list with elements of type 'a string'.
Default value	

Table 133. causal_clustering.protocol_implementations.raft

Description	Raft protocol implementation versions that this instance will allow in negotiation as a comma-separated list. Order is not relevant: the greatest value will be preferred. An empty list will allow all supported versions. Example value: "1.0, 1.3, 2.0, 2.1"
Valid values	causal_clustering.protocol_implementations.raft, a ',' separated list with elements of type 'an application protocol version'.
Default value	

Table 134. causal_clustering.pull_interval

Description	Interval of pulling updates from cores.
Valid values	causal_clustering.pull_interval, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	1s

Table 135. causal_clustering.raft_advertised_address

Description	Advertised hostname/IP address and port for the RAFT server.
Valid values	causal_clustering.raft_advertised_address, a socket address. If missing port or hostname it is acquired from dbms.default_advertised_address
Default value	`:7000`

Table 136. causal_clustering.raft_handler_parallelism

Description	Limits amount of global threads shared by raft groups for handling bathing of messages and timeout events.
Valid values	causal_clustering.raft_handler_parallelism, an integer which is minimum `1`
Default value	`8`

Table 137. causal_clustering.raft_in_queue_max_batch_bytes

Description	Largest batch processed by RAFT in bytes.
Valid values	causal_clustering.raft_in_queue_max_batch_bytes, a byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `TiB`, `TB`, `PiB`, `PB`, `EiB`, `EB`)
Default value	`8.00MiB`

Table 138. causal_clustering.raft_in_queue_max_bytes

Description	Maximum number of bytes in the RAFT in-queue.
Valid values	causal_clustering.raft_in_queue_max_bytes, a byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `TiB`, `TB`, `PiB`, `PB`, `EiB`, `EB`)
Default value	`2.00GiB`

Table 139. causal_clustering.raft_listen_address

Description	Network interface and port for the RAFT server to listen on.
Valid values	causal_clustering.raft_listen_address, a socket address. If missing port or hostname it is acquired from dbms.default_listen_address
Default value	`:7000`

Table 140. causal_clustering.raft_log_implementation

Description	RAFT log implementation.
Valid values	causal_clustering.raft_log_implementation, a string
Default value	`SEGMENTED`

Table 141. causal_clustering.raft_log_prune_strategy

Description	RAFT log pruning strategy.
Valid values	causal_clustering.raft_log_prune_strategy, a string
Default value	`1g size`

Table 142. causal_clustering.raft_log_pruning_frequency

Description	RAFT log pruning frequency.
Valid values	causal_clustering.raft_log_pruning_frequency, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`10m`

Table 143. causal_clustering.raft_log_reader_pool_size

Description	RAFT log reader pool size.
Valid values	causal_clustering.raft_log_reader_pool_size, an integer
Default value	`8`

Table 144. causal_clustering.raft_log_rotation_size

Description	RAFT log rotation size.
Valid values	causal_clustering.raft_log_rotation_size, a byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `TiB`, `TB`, `PiB`, `PB`, `EiB`, `EB`) which is minimum `1.00KiB`
Default value	`250.00MiB`

Table 145. causal_clustering.raft_membership_state_size

Description	The maximum file size before the membership state file is rotated (in unit of entries)

Valid values	causal_clustering.raft_membership_state_size, an integer
Default value	`1000`

Table 146. causal_clustering.raft_term_state_size

Description	The maximum file size before the term state file is rotated (in unit of entries)
Valid values	causal_clustering.raft_term_state_size, an integer
Default value	`1000`

Table 147. causal_clustering.raft_vote_state_size

Description	The maximum file size before the vote state file is rotated (in unit of entries)
Valid values	causal_clustering.raft_vote_state_size, an integer
Default value	`1000`

Table 148. 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, a boolean
Default value	`false`

Table 149. causal_clustering.replicated_lease_state_size

Description	The maximum file size before the replicated lease state file is rotated (in unit of entries)
Valid values	causal_clustering.replicated_lease_state_size, an integer
Default value	`1000`

Table 150. causal_clustering.replication_leader_await_timeout

Description	The duration for which the replicator will await a new leader.
Valid values	causal_clustering.replication_leader_await_timeout, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')

Default value	`10s`

Table 151. causal_clustering.replication_retry_timeout_base

Description	The initial timeout until replication is retried. The timeout will increase exponentially.
Valid values	causal_clustering.replication_retry_timeout_base, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`10s`

Table 152. causal_clustering.replication_retry_timeout_limit

Description	The upper limit for the exponentially incremented retry timeout.
Valid values	causal_clustering.replication_retry_timeout_limit, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`1m`

Table 153. causal_clustering.server_groups

Description	A list of group names for the server used when configuring load balancing and replication policies.
Valid values	causal_clustering.server_groups, a ',' separated list with elements of type 'a string identifying a Server Group'.
Dynamic	true
Default value	

Table 154. causal_clustering.state_machine_apply_max_batch_size

Description	The maximum number of operations to be batched during applications of operations in the state machines.
Valid values	causal_clustering.state_machine_apply_max_batch_size, an integer
Default value	`16`

Table 155. causal_clustering.state_machine_flush_window_size

Description	The number of operations to be processed before the state machines flush to disk.
Valid values	causal_clustering.state_machine_flush_window_size, an integer

Default value	`4096`

Table 156. causal_clustering.status_throughput_window

Description	Sampling window for throughput estimate reported in the status endpoint.
Valid values	causal_clustering.status_throughput_window, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's') which is in the range `1s` to `5m`
Default value	`5s`

Table 157. causal_clustering.store_copy_chunk_size

Description	Store copy chunk size.
Valid values	causal_clustering.store_copy_chunk_size, an integer which is in the range `4096` to `1048576`
Default value	`32768`

Table 158. causal_clustering.store_copy_max_retry_time_per_request

Description	Maximum retry time per request during store copy. Regular store files and indexes are downloaded in separate requests during store copy. This configures the maximum time failed requests are allowed to resend.
Valid values	causal_clustering.store_copy_max_retry_time_per_request, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`20m`

Table 159. causal_clustering.transaction_advertised_address

Description	Advertised hostname/IP address and port for the transaction shipping server.
Valid values	causal_clustering.transaction_advertised_address, a socket address. If missing port or hostname it is acquired from dbms.default_advertised_address
Default value	`:6000`

Table 160. causal_clustering.transaction_listen_address

Description	Network interface and port for the transaction shipping server to listen on. Please note that it is also possible to run the backup client against this port so always limit access to it via the firewall and configure an ssl policy.

Valid values	causal_clustering.transaction_listen_address, a socket address. If missing port or hostname it is acquired from dbms.default_listen_address
Default value	`:6000`

Table 161. causal_clustering.unknown_address_logging_throttle

Description	Throttle limit for logging unknown cluster member address.
Valid values	causal_clustering.unknown_address_logging_throttle, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`10s`

Table 162. causal_clustering.upstream_selection_strategy

Description	An ordered list in descending preference of the strategy which read replicas use to choose the upstream server from which to pull transactional updates.
Valid values	causal_clustering.upstream_selection_strategy, a ',' separated list with elements of type 'a string'.
Default value	`default`

Table 163. causal_clustering.user_defined_upstream_strategy

Description	Configuration of a user-defined upstream selection strategy. The user-defined strategy is used if the list of strategies (`causal_clustering.upstream_selection_strategy`) includes the value `user_defined`.
Valid values	causal_clustering.user_defined_upstream_strategy, a string
Default value	

Table 164. cypher.default_language_version

Description	Set this to specify the default parser (language version).
Valid values	cypher.default_language_version, one of [default, 3.5, 4.1, 4.2]
Default value	`default`

Table 165. cypher.forbid_exhaustive_shortestpath

Description	This setting is associated with performance optimization. Set this to `true` in situations where it is preferable to have any queries using the 'shortestPath' function terminate as soon as possible with no answer, rather than potentially running for a long time attempting to find an answer (even if there is no path to be found). For most queries, the 'shortestPath' algorithm will return the correct answer very quickly. However there are some cases where it is possible that the fast bidirectional breadth-first search algorithm will find no results even if they exist. This can happen when the predicates in the `WHERE` clause applied to 'shortestPath' cannot be applied to each step of the traversal, and can only be applied to the entire path. When the query planner detects these special cases, it will plan to perform an exhaustive depth-first search if the fast algorithm finds no paths. However, the exhaustive search may be orders of magnitude slower than the fast algorithm. If it is critical that queries terminate as soon as possible, it is recommended that this option be set to `true`, which means that Neo4j will never consider using the exhaustive search for shortestPath queries. However, please note that if no paths are found, an error will be thrown at run time, which will need to be handled by the application.
Valid values	cypher.forbid_exhaustive_shortestpath, a boolean
Default value	`false`

Table 166. cypher.forbid_shortestpath_common_nodes

Description	This setting is associated with performance optimization. The shortest path algorithm does not work when the start and end nodes are the same. With this setting set to `false` no path will be returned when that happens. The default value of `true` will instead throw an exception. This can happen if you perform a shortestPath search after a cartesian product that might have the same start and end nodes for some of the rows passed to shortestPath. If it is preferable to not experience this exception, and acceptable for results to be missing for those rows, then set this to `false`. If you cannot accept missing results, and really want the shortestPath between two common nodes, then re-write the query using a standard Cypher variable length pattern expression followed by ordering by path length and limiting to one result.
Valid values	cypher.forbid_shortestpath_common_nodes, a boolean
Default value	`true`

Table 167. cypher.hints_error

Description	Set this to specify the behavior when Cypher planner or runtime hints cannot be fulfilled. If true, then non-conformance will result in an error, otherwise only a warning is generated.

Valid values	cypher.hints_error, a boolean
Default value	`false`

Table 168. cypher.lenient_create_relationship

Description	Set this to change the behavior for Cypher create relationship when the start or end node is missing. By default this fails the query and stops execution, but by setting this flag the create operation is simply not performed and execution continues.
Valid values	cypher.lenient_create_relationship, a boolean
Default value	`false`

Table 169. cypher.min_replan_interval

Description	The minimum time between possible cypher query replanning events. After this time, the graph statistics will be evaluated, and if they have changed by more than the value set by cypher.statistics_divergence_threshold, the query will be replanned. If the statistics have not changed sufficiently, the same interval will need to pass before the statistics will be evaluated again. Each time they are evaluated, the divergence threshold will be reduced slightly until it reaches 10% after 7h, so that even moderately changing databases will see query replanning after a sufficiently long time interval.
Valid values	cypher.min_replan_interval, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`10s`

Table 170. cypher.planner

Description	Set this to specify the default planner for the default language version.
Valid values	cypher.planner, one of [DEFAULT, COST]
Default value	`DEFAULT`

Table 171. cypher.statistics_divergence_threshold

Description	The threshold for statistics above which a plan is considered stale. If any of the underlying statistics used to create the plan have changed more than this value, the plan will be considered stale and will be replanned. Change is calculated as `abs(a-b)/max(a,b)`. This means that a value of `0.75` requires the database to quadruple in size before query replanning. A value of `0` means that the query will be replanned as soon as there is any change in statistics and the replan interval has elapsed. This interval is defined by `cypher.min_replan_interval` and defaults to 10s. After this interval, the divergence threshold will slowly start to decline, reaching 10% after about 7h. This will ensure that long running databases will still get query replanning on even modest changes, while not replanning frequently unless the changes are very large.
Valid values	cypher.statistics_divergence_threshold, a double which is in the range `0.0` to `1.0`
Default value	`0.75`

Table 172. db.temporal.timezone

Description	Database timezone for temporal functions. All Time and DateTime values that are created without an explicit timezone will use this configured default timezone.
Valid values	db.temporal.timezone, a string describing a timezone, either described by offset (e.g. '+02:00') or by name (e.g. 'Europe/Stockholm')
Default value	`Z`

Table 173. dbms.allow_single_automatic_upgrade

Description	Whether to allow a system graph upgrade to happen automatically in single instance mode (dbms.mode=SINGLE). Default is true. In clustering environments no automatic upgrade will happen (dbms.mode=CORE or dbms.mode=READ_REPLICA). If set to false, or when in a clustering environment, it is necessary to call the procedure `dbms.upgrade()` to complete the upgrade.
Valid values	dbms.allow_single_automatic_upgrade, a boolean
Dynamic	true
Default value	`true`

Table 174. dbms.allow_upgrade

Description	Whether to allow a store upgrade in case the current version of the database starts against an older version of the store.
Valid values	dbms.allow_upgrade, a boolean
Dynamic	true
Default value	`false`

Table 175. dbms.backup.enabled

Description	Enable support for running online backups.
Valid values	dbms.backup.enabled, a boolean
Default value	`true`

Table 176. dbms.backup.listen_address

Description	Network interface and port for the backup server to listen on.
Valid values	dbms.backup.listen_address, a socket address
Default value	`127.0.0.1:6362`

Table 177. dbms.checkpoint

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

Table 178. 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, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`15m`

Table 179. 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, an integer which is minimum `1`
Default value	`100000`

Table 180. dbms.checkpoint.iops.limit

Description	Limit the number of IOs the background checkpoint process will consume per second. This setting is advisory, is ignored in Neo4j Community Edition, and is followed to best effort in Enterprise Edition. An IO is in this case a 8 KiB (mostly sequential) write. Limiting the write IO in this way will leave more bandwidth in the IO subsystem to service random-read IOs, which is important for the response time of queries when the database cannot fit entirely in memory. The only drawback of this setting is that longer checkpoint times may lead to slightly longer recovery times in case of a database or system crash. A lower number means lower IO pressure, and consequently longer checkpoint times. Set this to -1 to disable the IOPS limit and remove the limitation entirely; this will let the checkpointer flush data as fast as the hardware will go. Removing the setting, or commenting it out, will set the default value of 600.

Valid values	dbms.checkpoint.iops.limit, an integer
Dynamic	true
Default value	`600`

Table 181. dbms.config.strict_validation

Description	A strict configuration validation will prevent the database from starting up if unknown configuration options are specified in the neo4j settings namespace (such as dbms., cypher., etc).
Valid values	dbms.config.strict_validation, a boolean
Default value	`false`

Table 182. dbms.connector.bolt.advertised_address

Description	Advertised address for this connector.
Valid values	dbms.connector.bolt.advertised_address, a socket address. If missing port or hostname it is acquired from dbms.default_advertised_address
Default value	`:7687`

Table 183. dbms.connector.bolt.enabled

Description	Enable the bolt connector.
Valid values	dbms.connector.bolt.enabled, a boolean
Default value	`true`

Table 184. dbms.connector.bolt.listen_address

Description	Address the connector should bind to.
Valid values	dbms.connector.bolt.listen_address, a socket address. If missing port or hostname it is acquired from dbms.default_listen_address
Default value	`:7687`

Table 185. dbms.connector.bolt.ocsp_stapling_enabled

Description	Enable server OCSP stapling for bolt and http connectors.

Valid values	dbms.connector.bolt.ocsp_stapling_enabled, a boolean
Default value	`false`

Table 186. dbms.connector.bolt.thread_pool_keep_alive

Description	The maximum time an idle thread in the thread pool bound to this connector will wait for new tasks.
Valid values	dbms.connector.bolt.thread_pool_keep_alive, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`5m`

Table 187. dbms.connector.bolt.thread_pool_max_size

Description	The maximum number of threads allowed in the thread pool bound to this connector.
Valid values	dbms.connector.bolt.thread_pool_max_size, an integer
Default value	`400`

Table 188. dbms.connector.bolt.thread_pool_min_size

Description	The number of threads to keep in the thread pool bound to this connector, even if they are idle.
Valid values	dbms.connector.bolt.thread_pool_min_size, an integer
Default value	`5`

Table 189. dbms.connector.bolt.tls_level

Description	Encryption level to require this connector to use.
Valid values	dbms.connector.bolt.tls_level, one of [REQUIRED, OPTIONAL, DISABLED]
Default value	`DISABLED`

Table 190. dbms.connector.bolt.unsupported_thread_pool_shutdown_wait_time

Description	The maximum time to wait for the thread pool to finish processing its pending jobs and shutdown.

Valid values	dbms.connector.bolt.unsupported_thread_pool_shutdown_wait_time, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`5s`

Table 191. dbms.connector.http.advertised_address

Description	Advertised address for this connector.
Valid values	dbms.connector.http.advertised_address, a socket address. If missing port or hostname it is acquired from dbms.default_advertised_address
Default value	`:7474`

Table 192. dbms.connector.http.enabled

Description	Enable the http connector.
Valid values	dbms.connector.http.enabled, a boolean
Default value	`true`

Table 193. dbms.connector.http.listen_address

Description	Address the connector should bind to.
Valid values	dbms.connector.http.listen_address, a socket address. If missing port or hostname it is acquired from dbms.default_listen_address
Default value	`:7474`

Table 194. dbms.connector.https.advertised_address

Description	Advertised address for this connector.
Valid values	dbms.connector.https.advertised_address, a socket address. If missing port or hostname it is acquired from dbms.default_advertised_address
Default value	`:7473`

Table 195. dbms.connector.https.enabled

Description	Enable the https connector.
Valid values	dbms.connector.https.enabled, a boolean

Default value	`false`

Table 196. dbms.connector.https.listen_address

Description	Address the connector should bind to.
Valid values	dbms.connector.https.listen_address, a socket address. If missing port or hostname it is acquired from dbms.default_listen_address
Default value	`:7473`

Table 197. dbms.db.timezone

Description	Database timezone. Among other things, this setting influences which timezone the logs and monitoring procedures use.
Valid values	dbms.db.timezone, one of [UTC, SYSTEM]
Default value	`UTC`

Table 198. dbms.default_advertised_address

Description	Default hostname or IP address the server uses to advertise itself.
Valid values	dbms.default_advertised_address, a socket address which has no specified port
Default value	`localhost`

Table 199. dbms.default_database

Description	Name of the default database.
Valid values	dbms.default_database, A valid database name. Containing only alphabetic characters, numbers, dots and dashes, with a length between 3 and 63 characters. It should be starting with an alphabetic character but not with the name 'system'.
Default value	`neo4j`

Table 200. dbms.default_listen_address

Description	Default network interface to listen for incoming connections. To listen for connections on all interfaces, use "0.0.0.0".
Valid values	dbms.default_listen_address, a socket address which has no specified port
Default value	`localhost`

Table 201. dbms.directories.cluster_state

Description	Directory to hold cluster state including Raft log.
Valid values	dbms.directories.cluster_state, a path. If relative it is resolved from dbms.directories.data
Default value	`cluster-state`

Table 202. dbms.directories.data

Description	Path of the data directory. You must not configure more than one Neo4j installation to use the same data directory.
Valid values	dbms.directories.data, a path. If relative it is resolved from dbms.directories.neo4j_home
Default value	`data`

Table 203. dbms.directories.dumps.root

Description	Root location where Neo4j will store database dumps optionally produced when dropping said databases.
Valid values	dbms.directories.dumps.root, a path. If relative it is resolved from dbms.directories.data
Default value	`dumps`

Table 204. dbms.directories.import

Description	Sets the root directory for file URLs used with the Cypher `LOAD CSV` clause. This should be set to a directory relative to the Neo4j installation path, restricting access to only those files within that directory and its subdirectories. For example the value "import" will only enable access to files within the 'import' folder. Removing this setting will disable the security feature, allowing all files in the local system to be imported. Setting this to an empty field will allow access to all files within the Neo4j installation folder.
Valid values	dbms.directories.import, a path. If relative it is resolved from dbms.directories.neo4j_home

Table 205. dbms.directories.lib

Description	Path of the lib directory.

Valid values	dbms.directories.lib, a path. If relative it is resolved from dbms.directories.neo4j_home
Default value	`lib`

Table 206. dbms.directories.logs

Description	Path of the logs directory.
Valid values	dbms.directories.logs, a path. If relative it is resolved from dbms.directories.neo4j_home
Default value	`logs`

Table 207. dbms.directories.metrics

Description	The target location of the CSV files: a path to a directory wherein a CSV file per reported field will be written.
Valid values	dbms.directories.metrics, a path. If relative it is resolved from dbms.directories.neo4j_home
Default value	`metrics`

Table 208. dbms.directories.neo4j_home

Description	Root relative to which directory settings are resolved.
Valid values	dbms.directories.neo4j_home, a path which is absolute
Default value	`Defaults to current working directory`

Table 209. dbms.directories.plugins

Description	Location of the database plugin directory. Compiled Java JAR files that contain database procedures will be loaded if they are placed in this directory.
Valid values	dbms.directories.plugins, a path. If relative it is resolved from dbms.directories.neo4j_home
Default value	`plugins`

Table 210. dbms.directories.run

Description	Path of the run directory. This directory holds Neo4j's runtime state, such as a pidfile when it is running in the background. The pidfile is created when starting neo4j and removed when stopping it. It may be placed on an in-memory filesystem such as tmpfs.
Valid values	dbms.directories.run, a path. If relative it is resolved from dbms.directories.neo4j_home
Default value	`run`

Table 211. dbms.directories.script.root

Description	Root location where Neo4j will store scripts for configured databases.
Valid values	dbms.directories.script.root, a path. If relative it is resolved from dbms.directories.data
Default value	`scripts`

Table 212. dbms.directories.transaction.logs.root

Description	Root location where Neo4j will store transaction logs for configured databases.
Valid values	dbms.directories.transaction.logs.root, a path. If relative it is resolved from dbms.directories.data
Default value	`transactions`

Table 213. dbms.dynamic.setting.allowlist

Description	A list of setting name patterns (comma separated) that are allowed to be dynamically changed. The list may contain both full setting names, and partial names with the wildcard '*'. If this setting is left empty all dynamic settings updates will be blocked.
Valid values	dbms.dynamic.setting.allowlist, a ',' separated list with elements of type 'a string'.
Default value	`*`

Table 214. dbms.dynamic.setting.whitelist

Description	A list of setting name patterns (comma separated) that are allowed to be dynamically changed. The list may contain both full setting names, and partial names with the wildcard '*'. If this setting is left empty all dynamic settings updates will be blocked. Deprecated, use dbms.dynamic.setting.allowlist

Valid values	dbms.dynamic.setting.whitelist, a ',' separated list with elements of type 'a string'.
Default value	`*`
Deprecated	The `dbms.dynamic.setting.whitelist` configuration setting has been deprecated.

Table 215. dbms.filewatcher.enabled

Description	Allows the enabling or disabling of the file watcher service. This is an auxiliary service but should be left enabled in almost all cases.
Valid values	dbms.filewatcher.enabled, a boolean
Default value	`true`

Table 216. dbms.http_enabled_modules

Description	Defines the set of modules loaded into the Neo4j web server. Options include TRANSACTIONAL_ENDPOINTS, BROWSER, UNMANAGED_EXTENSIONS and ENTERPRISE_MANAGEMENT_ENDPOINTS (if applicable).
Valid values	dbms.http_enabled_modules, a ',' separated set with elements of type 'one of [TRANSACTIONAL_ENDPOINTS, UNMANAGED_EXTENSIONS, BROWSER, ENTERPRISE_MANAGEMENT_ENDPOINTS]'.
Default value	`TRANSACTIONAL_ENDPOINTS,UNMANAGED_EXTENSIONS,BROWSER,ENTERPRISE_MANAGEMENT_ENDPOINTS`

Table 217. dbms.import.csv.buffer_size

Description	The size of the internal buffer in bytes used by `LOAD CSV`. If the csv file contains huge fields this value may have to be increased.
Valid values	dbms.import.csv.buffer_size, a long which is minimum `1`
Default value	`2097152`

Table 218. dbms.import.csv.legacy_quote_escaping

Description	Selects whether to conform to the standard https://tools.ietf.org/html/rfc4180 for interpreting escaped quotation characters in CSV files loaded using `LOAD CSV`. Setting this to `false` will use the standard, interpreting repeated quotes """" as a single in-lined quote, while `true` will use the legacy convention originally supported in Neo4j 3.0 and 3.1, allowing a backslash to include quotes in-lined in fields.
Valid values	dbms.import.csv.legacy_quote_escaping, a boolean

Default value	`true`

Table 219. dbms.index.default_schema_provider

Description	Index provider to use for newly created schema indexes. An index provider may store different value types in separate physical indexes. native-btree-1.0: All value types and arrays of all value types, even composite keys, are stored in one native index. lucene+native-3.0: Like native-btree-1.0 but single property strings are stored in Lucene. A native index has faster updates, less heap and CPU usage compared to a Lucene index. A native index has some limitations around key size and slower execution of CONTAINS and ENDS WITH string index queries, compared to a Lucene index. Deprecated: Which index provider to use will be a fully internal concern.
Valid values	dbms.index.default_schema_provider, a string
Default value	`native-btree-1.0`
Deprecated	The `dbms.index.default_schema_provider` configuration setting has been deprecated.

Table 220. dbms.index.fulltext.default_analyzer

Description	The name of the analyzer that the fulltext indexes should use by default.
Valid values	dbms.index.fulltext.default_analyzer, a string
Default value	`standard-no-stop-words`

Table 221. dbms.index.fulltext.eventually_consistent

Description	Whether or not fulltext indexes should be eventually consistent by default or not.
Valid values	dbms.index.fulltext.eventually_consistent, a boolean
Default value	`false`

Table 222. dbms.index.fulltext.eventually_consistent_index_update_queue_max_length

Description	The eventually_consistent mode of the fulltext indexes works by queueing up index updates to be applied later in a background thread. This newBuilder sets an upper bound on how many index updates are allowed to be in this queue at any one point in time. When it is reached, the commit process will slow down and wait for the index update applier thread to make some more room in the queue.

Valid values	dbms.index.fulltext.eventually_consistent_index_update_queue_max_length, an integer which is in the range `1` to `50000000`
Default value	`10000`

Table 223. dbms.index_sampling.background_enabled

Description	Enable or disable background index sampling.
Valid values	dbms.index_sampling.background_enabled, a boolean
Default value	`true`

Table 224. dbms.index_sampling.sample_size_limit

Description	Index sampling chunk size limit.
Valid values	dbms.index_sampling.sample_size_limit, an integer which is in the range `1048576` to `2147483647`
Default value	`8388608`

Table 225. dbms.index_sampling.update_percentage

Description	Percentage of index updates of total index size required before sampling of a given index is triggered.
Valid values	dbms.index_sampling.update_percentage, an integer which is minimum `0`
Default value	`5`

Table 226. dbms.index_searcher_cache_size

Description	The maximum number of open Lucene index searchers.
Valid values	dbms.index_searcher_cache_size, an integer which is minimum `1`
Default value	`2147483647`
Deprecated	The `dbms.index_searcher_cache_size` configuration setting has been deprecated.

Table 227. dbms.jvm.additional

Description	Additional JVM arguments. Argument order can be significant. To use a Java commercial feature, the argument to unlock commercial features must precede the argument to enable the specific feature in the config value string.

Valid values	dbms.jvm.additional, a string
Default value	

Table 228. dbms.lock.acquisition.timeout

Description	The maximum time interval within which lock should be acquired. Zero (default) means timeout is disabled.
Valid values	dbms.lock.acquisition.timeout, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`0s`

Table 229. dbms.logs.debug.level

Description	Debug log level threshold.
Valid values	dbms.logs.debug.level, one of [DEBUG, INFO, WARN, ERROR, NONE]
Dynamic	true
Default value	`INFO`

Table 230. dbms.logs.debug.path

Description	Path to the debug log file.
Valid values	dbms.logs.debug.path, a path. If relative it is resolved from dbms.directories.logs
Default value	`debug.log`

Table 231. dbms.logs.debug.rotation.delay

Description	Minimum time interval after last rotation of the debug log before it may be rotated again.
Valid values	dbms.logs.debug.rotation.delay, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`5m`
Deprecated	The `dbms.logs.debug.rotation.delay` configuration setting has been deprecated.

Table 232. dbms.logs.debug.rotation.keep_number

Description	Maximum number of history files for the debug log.
Valid values	dbms.logs.debug.rotation.keep_number, an integer which is minimum `1`
Default value	`7`

Table 233. dbms.logs.debug.rotation.size

Description	Threshold for rotation of the debug log.
Valid values	dbms.logs.debug.rotation.size, a byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `TiB`, `TB`, `PiB`, `PB`, `EiB`, `EB`) which is in the range `0B` to `8388608.00TiB`
Default value	`20.00MiB`

Table 234. dbms.logs.gc.enabled

Description	Enable GC Logging.
Valid values	dbms.logs.gc.enabled, a boolean
Default value	`false`

Table 235. dbms.logs.gc.options

Description	GC Logging Options.
Valid values	dbms.logs.gc.options, a string

Table 236. dbms.logs.gc.rotation.keep_number

Description	Number of GC logs to keep.
Valid values	dbms.logs.gc.rotation.keep_number, an integer
Default value	`0`

Table 237. dbms.logs.gc.rotation.size

Description	Size of each GC log that is kept.
Valid values	dbms.logs.gc.rotation.size, a byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `TiB`, `TB`, `PiB`, `PB`, `EiB`, `EB`)

Table 238. dbms.logs.http.enabled

Description	Enable HTTP request logging.
Valid values	dbms.logs.http.enabled, a boolean
Default value	`false`

Table 239. dbms.logs.http.path

Description	Path to HTTP request log.
Valid values	dbms.logs.http.path, a path. If relative it is resolved from dbms.directories.logs
Default value	`http.log`

Table 240. dbms.logs.http.rotation.keep_number

Description	Number of HTTP logs to keep.
Valid values	dbms.logs.http.rotation.keep_number, an integer
Default value	`5`

Table 241. dbms.logs.http.rotation.size

Description	Size of each HTTP log that is kept.
Valid values	dbms.logs.http.rotation.size, a byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `TiB`, `TB`, `PiB`, `PB`, `EiB`, `EB`) which is in the range `0B` to `8388608.00TiB`
Default value	`20.00MiB`

Table 242. dbms.logs.query.allocation_logging_enabled

Description	Log allocated bytes for the executed queries being logged. The logged number is cumulative over the duration of the query, i.e. for memory intense or long-running queries the value may be larger than the current memory allocation. Requires `dbms.track_query_allocation=true`
Valid values	dbms.logs.query.allocation_logging_enabled, a boolean
Dynamic	true
Default value	`true`

Table 243. dbms.logs.query.early_raw_logging_enabled

Description	Log query text and parameters without obfuscating passwords. This allows queries to be logged earlier before parsing starts.
Valid values	dbms.logs.query.early_raw_logging_enabled, a boolean
Dynamic	true
Default value	`false`

Table 244. dbms.logs.query.enabled

Description	Log executed queries. Valid values are `OFF`, `INFO`, or `VERBOSE`.
	
	`OFF`
	no logging.
	
	`INFO`
	log queries at the end of execution, that take longer than the configured threshold, `dbms.logs.query.threshold`.
	
	`VERBOSE`
	log queries at the start and end of execution, regardless of `dbms.logs.query.threshold`.
	
	Log entries are by default written to the file *query.log* located in the Logs directory. This feature is available in the Neo4j Enterprise Edition.

Valid values	dbms.logs.query.enabled, one of [OFF, INFO, VERBOSE]
Dynamic	true
Default value	`VERBOSE`

Table 245. dbms.logs.query.page_logging_enabled

Description	Log page hits and page faults for the executed queries being logged.
Valid values	dbms.logs.query.page_logging_enabled, a boolean
Dynamic	true
Default value	`false`

Table 246. dbms.logs.query.parameter_full_entities

Description	Log complete parameter entities including id, labels or relationship type, and properties. If false, only the entity id will be logged. This only takes effect if `dbms.logs.query.parameter_logging_enabled = true`.
Valid values	dbms.logs.query.parameter_full_entities, a boolean
Dynamic	true
Default value	`false`

Table 247. dbms.logs.query.parameter_logging_enabled

Description	Log parameters for the executed queries being logged.
Valid values	dbms.logs.query.parameter_logging_enabled, a boolean
Dynamic	true
Default value	`true`

Table 248. dbms.logs.query.path

Description	Path to the query log file.
Valid values	dbms.logs.query.path, a path. If relative it is resolved from dbms.directories.logs
Default value	`query.log`

Table 249. dbms.logs.query.rotation.keep_number

Description	Maximum number of history files for the query log.
Valid values	dbms.logs.query.rotation.keep_number, an integer which is minimum `1`
Dynamic	true
Default value	`7`

Table 250. dbms.logs.query.rotation.size

Description	The file size in bytes at which the query log will auto-rotate. If set to zero then no rotation will occur. Accepts a binary suffix `k`, `m` or `g`.

Valid values	dbms.logs.query.rotation.size, a byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `TiB`, `TB`, `PiB`, `PB`, `EiB`, `EB`) which is in the range `0B` to `8388608.00TiB`
Dynamic	true
Default value	`20.00MiB`

Table 251. dbms.logs.query.runtime_logging_enabled

Description	Logs which runtime that was used to run the query.
Valid values	dbms.logs.query.runtime_logging_enabled, a boolean
Dynamic	true
Default value	`true`

Table 252. dbms.logs.query.threshold

Description	If the execution of query takes more time than this threshold, the query is logged once completed - provided query logging is set to INFO. Defaults to 0 seconds, that is all queries are logged.
Valid values	dbms.logs.query.threshold, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Dynamic	true
Default value	`0s`

Table 253. dbms.logs.query.time_logging_enabled

Description	Log detailed time information for the executed queries being logged, such as `(planning: 92, waiting: 0)`.
Valid values	dbms.logs.query.time_logging_enabled, a boolean
Dynamic	true
Default value	`false`

Table 254. dbms.logs.security.level

Description	Security log level threshold.

Valid values	dbms.logs.security.level, one of [DEBUG, INFO, WARN, ERROR, NONE]
Default value	`INFO`

Table 255. dbms.logs.security.path

Description	Path to the security log file.
Valid values	dbms.logs.security.path, a path. If relative it is resolved from dbms.directories.logs
Default value	`security.log`

Table 256. dbms.logs.security.rotation.delay

Description	Minimum time interval after last rotation of the security log before it may be rotated again.
Valid values	dbms.logs.security.rotation.delay, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`5m`
Deprecated	The `dbms.logs.security.rotation.delay` configuration setting has been deprecated.

Table 257. dbms.logs.security.rotation.keep_number

Description	Maximum number of history files for the security log.
Valid values	dbms.logs.security.rotation.keep_number, an integer which is minimum `1`
Default value	`7`

Table 258. dbms.logs.security.rotation.size

Description	Threshold for rotation of the security log.
Valid values	dbms.logs.security.rotation.size, a byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `TiB`, `TB`, `PiB`, `PB`, `EiB`, `EB`) which is in the range `0B` to `8388608.00TiB`
Default value	`20.00MiB`

Table 259. dbms.logs.user.path

Description	Path to the user log file. Note that if dbms.logs.user.stdout_enabled is enabled this setting will be ignored.
Valid values	dbms.logs.user.path, a path. If relative it is resolved from dbms.directories.logs
Default value	`neo4j.log`

Table 260. dbms.logs.user.rotation.delay

Description	Minimum time interval after last rotation of the user log (*neo4j.log*) before it may be rotated again. Note that if dbms.logs.user.stdout_enabled is enabled this setting will be ignored.
Valid values	dbms.logs.user.rotation.delay, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`5m`
Deprecated	The `dbms.logs.user.rotation.delay` configuration setting has been deprecated.

Table 261. dbms.logs.user.rotation.keep_number

Description	Maximum number of history files for the user log (*neo4j.log*). Note that if dbms.logs.user.stdout_enabled is enabled this setting will be ignored.
Valid values	dbms.logs.user.rotation.keep_number, an integer which is minimum `1`
Default value	`7`

Table 262. dbms.logs.user.rotation.size

Description	Threshold for rotation of the user log (*neo4j.log*). If set to 0, log rotation is disabled. Note that if dbms.logs.user.stdout_enabled is enabled this setting will be ignored.
Valid values	dbms.logs.user.rotation.size, a byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `TiB`, `TB`, `PiB`, `PB`, `EiB`, `EB`) which is in the range `0B` to `8388608.00TiB`
Default value	`0B`

Table 263. dbms.logs.user.stdout_enabled

Description	Send user logs to the process stdout. If this is disabled then logs will instead be sent to the file *neo4j.log* located in the logs directory.

Valid values	dbms.logs.user.stdout_enabled, a boolean
Default value	`true`

Table 264. dbms.max_databases

Description	The maximum number of databases.
Valid values	dbms.max_databases, a long which is minimum `2`
Default value	`100`

Table 265. dbms.memory.heap.initial_size

Description	Initial heap size. By default it is calculated based on available system resources. (valid units are `k`, `K`, `m`, `M`, `g`, `G`).
Valid values	dbms.memory.heap.initial_size, a string
Default value	

Table 266. dbms.memory.heap.max_size

Description	Maximum heap size. By default it is calculated based on available system resources. (valid units are `k`, `K`, `m`, `M`, `g`, `G`).
Valid values	dbms.memory.heap.max_size, a string
Default value	

Table 267. dbms.memory.off_heap.block_cache_size

Description	Defines the size of the off-heap memory blocks cache. The cache will contain this number of blocks for each block size that is power of two. Thus, maximum amount of memory used by blocks cache can be calculated as 2 * dbms.memory.off_heap.max_cacheable_block_size * `dbms.memory.off_heap.block_cache_size`
Valid values	dbms.memory.off_heap.block_cache_size, an integer which is minimum `16`
Default value	`128`

Table 268. dbms.memory.off_heap.max_cacheable_block_size

Description	Defines the maximum size of an off-heap memory block that can be cached to speed up allocations. The value must be a power of 2.

Valid values	dbms.memory.off_heap.max_cacheable_block_size, a byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `TiB`, `TB`, `PiB`, `PB`, `EiB`, `EB`) which is minimum `4.00KiB` and is power of 2
Default value	`512.00KiB`

Table 269. dbms.memory.off_heap.max_size

Description	The maximum amount of off-heap memory that can be used to store transaction state data; it's a total amount of memory shared across all active transactions. Zero means 'unlimited'. Used when dbms.tx_state.memory_allocation is set to 'OFF_HEAP'.
Valid values	dbms.memory.off_heap.max_size, a byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `TiB`, `TB`, `PiB`, `PB`, `EiB`, `EB`) which is minimum `0B`
Default value	`2.00GiB`

Table 270. dbms.memory.pagecache.directio

Description	Use direct I/O for page cache. Setting is supported only on Linux and only for a subset of record formats that use platform aligned page size.
Valid values	dbms.memory.pagecache.directio, a boolean
Default value	`false`

Table 271. dbms.memory.pagecache.flush.buffer.enabled

Description	Page cache can be configured to use a temporal buffer for flushing purposes. It is used to combine, if possible, sequence of several cache pages into one bigger buffer to minimize the number of individual IOPS performed and better utilization of available I/O resources, especially when those are restricted.
Valid values	dbms.memory.pagecache.flush.buffer.enabled, a boolean
Dynamic	true
Default value	`false`

Table 272. dbms.memory.pagecache.flush.buffer.size_in_pages

Description	Page cache can be configured to use a temporal buffer for flushing purposes. It is used to combine, if possible, sequence of several cache pages into one bigger buffer to minimize the number of individual IOPS performed and better utilization of available I/O resources, especially when those are restricted. Use this setting to configure individual file flush buffer size in pages (8KiB). To be able to utilize this buffer during page cache flushing, buffered flush should be enabled.
Valid values	dbms.memory.pagecache.flush.buffer.size_in_pages, an integer which is in the range 1 to 512
Dynamic	true
Default value	128

Table 273. dbms.memory.pagecache.scan.prefetchers

Description	The maximum number of worker threads to use for pre-fetching data when doing sequential scans. Set to '0' to disable pre-fetching for scans.
Valid values	dbms.memory.pagecache.scan.prefetchers, an integer which is in the range 0 to 255
Default value	4

Table 274. dbms.memory.pagecache.size

Description	The amount of memory to use for mapping the store files, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g'). If Neo4j is running on a dedicated server, then it is generally recommended to leave about 2-4 gigabytes for the operating system, give the JVM enough heap to hold all your transaction state and query context, and then leave the rest for the page cache. If no page cache memory is configured, then a heuristic setting is computed based on available system resources.
Valid values	dbms.memory.pagecache.size, a string

Table 275. dbms.memory.pagecache.swapper

Description	This setting is not used anymore.
Valid values	dbms.memory.pagecache.swapper, a string
Deprecated	The `dbms.memory.pagecache.swapper` configuration setting has been deprecated.

Table 276. dbms.memory.pagecache.warmup.enable

Description	Page cache can be configured to perform usage sampling of loaded pages that can be used to construct active load profile. According to that profile pages can be reloaded on the restart, replication, etc. This setting allows disabling that behavior. This feature available in Neo4j Enterprise Edition.
Valid values	dbms.memory.pagecache.warmup.enable, a boolean
Default value	`true`

Table 277. dbms.memory.pagecache.warmup.preload

Description	Page cache warmup can be configured to prefetch files, preferably when cache size is bigger than store size. Files to be prefetched can be filtered by 'dbms.memory.pagecache.warmup.preload.allowlist'. Enabling this disables warmup by profile.
Valid values	dbms.memory.pagecache.warmup.preload, a boolean
Default value	`false`

Table 278. dbms.memory.pagecache.warmup.preload.allowlist

Description	Page cache warmup prefetch file allowlist regex. By default matches all files.
Valid values	dbms.memory.pagecache.warmup.preload.allowlist, a string
Default value	`.*`

Table 279. dbms.memory.pagecache.warmup.preload.whitelist

Description	Page cache warmup prefetch file whitelist regex. By default matches all files. Deprecated, use dbms.memory.pagecache.warmup.preload.allowlist
Valid values	dbms.memory.pagecache.warmup.preload.whitelist, a string
Default value	`.*`
Deprecated	The `dbms.memory.pagecache.warmup.preload.whitelist` configuration setting has been deprecated.

Table 280. dbms.memory.pagecache.warmup.profile.interval

Description	The profiling frequency for the page cache. Accurate profiles allow the page cache to do active warmup after a restart, reducing the mean time to performance. This feature available in Neo4j Enterprise Edition.

Valid values	dbms.memory.pagecache.warmup.profile.interval, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`1m`

Table 281. dbms.memory.tracking.enable

Description	Enable off heap and on heap memory tracking. Should not be set to `false` for clusters.
Valid values	dbms.memory.tracking.enable, a boolean
Default value	`true`

Table 282. dbms.memory.transaction.database_max_size

Description	Limit the amount of memory that all transactions in one database can consume, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g'). Zero means 'unlimited'.
Valid values	dbms.memory.transaction.database_max_size, a byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `TiB`, `TB`, `PiB`, `PB`, `EiB`, `EB`) which is minimum `10.00MiB` or is `0B`
Dynamic	true
Default value	`0B`

Table 283. dbms.memory.transaction.global_max_size

Description	Limit the amount of memory that all of the running transactions can consume, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g'). Zero means 'unlimited'.
Valid values	dbms.memory.transaction.global_max_size, a byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `TiB`, `TB`, `PiB`, `PB`, `EiB`, `EB`) which is minimum `10.00MiB` or is `0B`
Dynamic	true
Default value	`0B`

Table 284. dbms.memory.transaction.max_size

Description	Limit the amount of memory that a single transaction can consume, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g'). Zero means 'largest possible value'. When `dbms.mode=SINGLE` this is 'unlimited'. When `dbms.mode=CORE` or `dbms.mode=READ_REPLICA` this is '2G'.
Valid values	dbms.memory.transaction.max_size, a byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `TiB`, `TB`, `PiB`, `PB`, `EiB`, `EB`) which is minimum `1.00MiB` or is `0B` and depends on dbms.mode. If dbms.mode is `CORE` or is `READ_REPLICA` then it is maximum `2.00GiB` otherwise it is unconstrained.
Dynamic	true
Default value	`0B`

Table 285. dbms.mode

Description	Configure the operating mode of the database — 'SINGLE' for stand-alone operation, 'CORE' for operating as a core member of a Causal Cluster, or 'READ_REPLICA' for operating as a read replica member of a Causal Cluster. Only SINGLE mode is allowed in Community.
Valid values	dbms.mode, one of [SINGLE, CORE, READ_REPLICA]
Default value	`SINGLE`

Table 286. dbms.netty.ssl.provider

Description	Netty SSL provider.
Valid values	dbms.netty.ssl.provider, one of [JDK, OPENSSL, OPENSSL_REFCNT]
Default value	`JDK`

Table 287. dbms.query_cache_size

Description	The number of Cypher query execution plans that are cached.
Valid values	dbms.query_cache_size, an integer which is minimum `0`
Default value	`1000`

Table 288. dbms.read_only

Description	Only allow read operations from this Neo4j instance. This mode still requires write access to the directory for lock purposes.

Valid values	dbms.read_only, a boolean
Default value	`false`

Table 289. dbms.reconciler.max_backoff

Description	Defines the maximum amount of time to wait before retrying after the dbms fails to reconcile a database to its desired state.
Valid values	dbms.reconciler.max_backoff, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's') which is minimum `1m`
Default value	`1h`

Table 290. dbms.reconciler.max_parallelism

Description	Defines the level of parallelism employed by the reconciler. By default the parallelism equals the number of available processors or 8 (whichever is smaller). If configured as 0, the parallelism of the reconciler will be unbounded.
Valid values	dbms.reconciler.max_parallelism, an integer which is minimum `0`
Default value	`8`

Table 291. dbms.reconciler.may_retry

Description	Defines whether the dbms may retry reconciling a database to its desired state.
Valid values	dbms.reconciler.may_retry, a boolean
Default value	`false`

Table 292. dbms.reconciler.min_backoff

Description	Defines the minimum amount of time to wait before retrying after the dbms fails to reconcile a database to its desired state.
Valid values	dbms.reconciler.min_backoff, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's') which is minimum `1s`
Default value	`2s`

Table 293. dbms.record_format

Description	Database record format. Valid values: `standard`, `aligned`, and `high_limit`. The `aligned` format is essentially the `standard` format with some minimal padding at the end of pages such that a single record will never cross a page boundary. The `high_limit` format is available for Enterprise Edition only. It is required if you have a graph that is larger than 34 billion nodes, 34 billion relationships, or 68 billion properties. A change of the record format is irreversible. Certain operations may suffer from a performance penalty of up to 10%, which is why this format is not switched on by default. However, if you want to change the configured record format value, you must also set `dbms.allow_upgrade=true`, because the setting implies a one-way store format migration.
Valid values	dbms.record_format, a string
Default value	`aligned for new databases, latest version of current format for existing databases`

Table 294. dbms.recovery.fail_on_missing_files

Description	If `true`, Neo4j will abort recovery if transaction log files are missing. Setting this to `false` will allow Neo4j to create new empty missing files for the already existing database, but the integrity of the database might be compromised.
Valid values	dbms.recovery.fail_on_missing_files, a boolean
Default value	`true`

Table 295. dbms.relationship_grouping_threshold

Description	Relationship count threshold for considering a node to be dense.
Valid values	dbms.relationship_grouping_threshold, an integer which is minimum `1`
Default value	`50`

Table 296. dbms.rest.transaction.idle_timeout

Description	Timeout for idle transactions in the REST endpoint.
Valid values	dbms.rest.transaction.idle_timeout, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`1m`

Table 297. dbms.routing.advertised_address

Description	The advertised address for the intra-cluster routing connector.

Valid values	dbms.routing.advertised_address, a socket address. If missing port or hostname it is acquired from dbms.default_advertised_address
Default value	`:7688`

Table 298. dbms.routing.driver.api

Description	Determines which driver API will be used. ASYNC must be used when the remote instance is 3.5.
Valid values	dbms.routing.driver.api, one of [RX, ASYNC]
Default value	`RX`

Table 299. dbms.routing.driver.connection.connect_timeout

Description	Socket connection timeout. A timeout of zero is treated as an infinite timeout and will be bound by the timeout configured on the operating system level.
Valid values	dbms.routing.driver.connection.connect_timeout, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`5s`

Table 300. dbms.routing.driver.connection.max_lifetime

Description	Pooled connections older than this threshold will be closed and removed from the pool. Setting this option to a low value will cause a high connection churn and might result in a performance hit. It is recommended to set maximum lifetime to a slightly smaller value than the one configured in network equipment (load balancer, proxy, firewall, etc. can also limit maximum connection lifetime). Zero and negative values result in lifetime not being checked.
Valid values	dbms.routing.driver.connection.max_lifetime, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`1h`

Table 301. dbms.routing.driver.connection.pool.acquisition_timeout

Description	Maximum amount of time spent attempting to acquire a connection from the connection pool. This timeout only kicks in when all existing connections are being used and no new connections can be created because maximum connection pool size has been reached. Error is raised when connection can't be acquired within configured time. Negative values are allowed and result in unlimited acquisition timeout. Value of 0 is allowed and results in no timeout and immediate failure when connection is unavailable.
Valid values	dbms.routing.driver.connection.pool.acquisition_timeout, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`1m`

Table 302. dbms.routing.driver.connection.pool.idle_test

Description	Pooled connections that have been idle in the pool for longer than this timeout will be tested before they are used again, to ensure they are still alive. If this option is set too low, an additional network call will be incurred when acquiring a connection, which causes a performance hit. If this is set high, no longer live connections might be used which might lead to errors. Hence, this parameter tunes a balance between the likelihood of experiencing connection problems and performance Normally, this parameter should not need tuning. Value 0 means connections will always be tested for validity.
Valid values	dbms.routing.driver.connection.pool.idle_test, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`No connection liveliness check is done by default.`

Table 303. dbms.routing.driver.connection.pool.max_size

Description	Maximum total number of connections to be managed by a connection pool. The limit is enforced for a combination of a host and user. Negative values are allowed and result in unlimited pool. Value of 0is not allowed.
Valid values	dbms.routing.driver.connection.pool.max_size, an integer
Default value	`Unlimited`

Table 304. dbms.routing.driver.logging.level

Description	Sets level for driver internal logging.
Valid values	dbms.routing.driver.logging.level, one of [DEBUG, INFO, WARN, ERROR, NONE]
Default value	`Value of dbms.logs.debug.level`

Table 305. dbms.routing.enabled

Description	Enable intra-cluster routing using an additional bolt connector.
Valid values	dbms.routing.enabled, a boolean
Default value	`false`

Table 306. dbms.routing.listen_address

Description	The address the routing connector should bind to.
Valid values	dbms.routing.listen_address, a socket address. If missing port or hostname it is acquired from dbms.default_listen_address
Default value	`:7688`

Table 307. dbms.routing_ttl

Description	How long callers should cache the response of the routing procedure `dbms.routing.getRoutingTable()`
Valid values	dbms.routing_ttl, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's') which is minimum `1s`
Default value	`5m`

Table 308. dbms.security.allow_csv_import_from_file_urls

Description	Determines if Cypher will allow using file URLs when loading data using `LOAD CSV`. Setting this value to `false` will cause Neo4j to fail `LOAD CSV` clauses that load data from the file system.
Valid values	dbms.security.allow_csv_import_from_file_urls, a boolean
Default value	`true`

Table 309. dbms.security.auth_cache_max_capacity

Description	The maximum capacity for authentication and authorization caches (respectively).
Valid values	dbms.security.auth_cache_max_capacity, an integer
Default value	`10000`

Table 310. 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, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`10m`

Table 311. 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 `dbms.security.auth_cache_max_capacity` is exceeded.
Valid values	dbms.security.auth_cache_use_ttl, a boolean
Default value	`true`

Table 312. dbms.security.auth_enabled

Description	Enable auth requirement to access Neo4j.
Valid values	dbms.security.auth_enabled, a boolean
Default value	`true`

Table 313. dbms.security.auth_lock_time

Description	The amount of time user account should be locked after a configured number of unsuccessful authentication attempts. The locked out user will not be able to log in until the lock period expires, even if correct credentials are provided. Setting this configuration option to a low value is not recommended because it might make it easier for an attacker to brute force the password.
Valid values	dbms.security.auth_lock_time, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's') which is minimum `0s`
Default value	`5s`

Table 314. dbms.security.auth_max_failed_attempts

Description	The maximum number of unsuccessful authentication attempts before imposing a user lock for the configured amount of time, as defined by `dbms.security.auth_lock_time`.The locked out user will not be able to log in until the lock period expires, even if correct credentials are provided. Setting this configuration option to values less than 3 is not recommended because it might make it easier for an attacker to brute force the password.
Valid values	dbms.security.auth_max_failed_attempts, an integer which is minimum `0`
Default value	`3`

Table 315. dbms.security.authentication_providers

Description	A list of security authentication providers containing the users and roles. This can be any of the built-in `native` or `ldap` providers, or it can be an externally provided plugin, with a custom name prefixed by `plugin-`, i.e. `plugin-<AUTH_PROVIDER_NAME>`. They will be queried in the given order when login is attempted.
Valid values	dbms.security.authentication_providers, a ',' separated list with elements of type 'a string'.
Default value	`native`

Table 316. dbms.security.authorization_providers

Description	A list of security authorization providers containing the users and roles. This can be any of the built-in `native` or `ldap` providers, or it can be an externally provided plugin, with a custom name prefixed by `plugin-`, i.e. `plugin-<AUTH_PROVIDER_NAME>`. They will be queried in the given order when login is attempted.
Valid values	dbms.security.authorization_providers, a ',' separated list with elements of type 'a string'.
Default value	`native`

Table 317. dbms.security.causal_clustering_status_auth_enabled

Description	Require authorization for access to the Causal Clustering status endpoints.
Valid values	dbms.security.causal_clustering_status_auth_enabled, a boolean
Default value	`true`

Table 318. dbms.security.http_access_control_allow_origin

Description	Value of the Access-Control-Allow-Origin header sent over any HTTP or HTTPS connector. This defaults to '*', which allows broadest compatibility. Note that any URI provided here limits HTTP/HTTPS access to that URI only.
Valid values	dbms.security.http_access_control_allow_origin, a string
Default value	`*`

Table 319. dbms.security.http_auth_allowlist

Description	Defines an allowlist of http paths where Neo4j authentication is not required.
Valid values	dbms.security.http_auth_allowlist, a ',' separated list with elements of type 'a string'.
Default value	`/,/browser.*`

Table 320. dbms.security.http_auth_whitelist

Description	Defines a whitelist of http paths where Neo4j authentication is not required. Deprecated, use dbms.security.http_auth_allowlist
Valid values	dbms.security.http_auth_whitelist, a ',' separated list with elements of type 'a string'.
Default value	`/,/browser.*`
Deprecated	The `dbms.security.http_auth_whitelist` configuration setting has been deprecated.

Table 321. dbms.security.http_strict_transport_security

Description	Value of the HTTP Strict-Transport-Security (HSTS) response header. This header tells browsers that a webpage should only be accessed using HTTPS instead of HTTP. It is attached to every HTTPS response. Setting is not set by default so 'Strict-Transport-Security' header is not sent. Value is expected to contain directives like 'max-age', 'includeSubDomains' and 'preload'.
Valid values	dbms.security.http_strict_transport_security, a string

Table 322. dbms.security.ldap.authentication.cache_enabled

Description	Determines if the result of authentication via the LDAP server should be cached or not. Caching is used to limit the number of LDAP requests that have to be made over the network for users that have already been authenticated successfully. A user can be authenticated against an existing cache entry (instead of via an LDAP server) as long as it is alive (see `dbms.security.auth_cache_ttl`). An important consequence of setting this to `true` is that Neo4j then needs to cache a hashed version of the credentials in order to perform credentials matching. This hashing is done using a cryptographic hash function together with a random salt. Preferably a conscious decision should be made if this method is considered acceptable by the security standards of the organization in which this Neo4j instance is deployed.
Valid values	dbms.security.ldap.authentication.cache_enabled, a boolean
Default value	`true`

Table 323. dbms.security.ldap.authentication.mechanism

Description	LDAP authentication mechanism. This is one of `simple` or a SASL mechanism supported by JNDI, for example `DIGEST-MD5`. `simple` is basic username and password authentication and SASL is used for more advanced mechanisms. See RFC 2251 LDAPv3 documentation for more details.
Valid values	dbms.security.ldap.authentication.mechanism, a string
Default value	`simple`

Table 324. dbms.security.ldap.authentication.use_samaccountname

Description	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.
Valid values	dbms.security.ldap.authentication.use_samaccountname, a boolean
Default value	`false`

Table 325. dbms.security.ldap.authentication.user_dn_template

Description	LDAP user DN template. An LDAP object is referenced by its distinguished name (DN), and a user DN is an LDAP fully-qualified unique user identifier. This setting is used to generate an LDAP DN that conforms with the LDAP directory's schema from the user principal that is submitted with the authentication token when logging in. The special token \{0} is a placeholder where the user principal will be substituted into the DN string.

Valid values	dbms.security.ldap.authentication.user_dn_template, a string
Default value	`uid={0},ou=users,dc=example,dc=com`

Table 326. dbms.security.ldap.authorization.group_membership_attributes

Description	A list of attribute names on a user object that contains groups to be used for mapping to roles when LDAP authorization is enabled.
Valid values	dbms.security.ldap.authorization.group_membership_attributes, a ',' separated list with elements of type 'a string'.
Default value	`memberOf`

Table 327. dbms.security.ldap.authorization.group_to_role_mapping

Description	An authorization mapping from LDAP group names to Neo4j role names. The map should be formatted as a semicolon separated list of key-value pairs, where the key is the LDAP group name and the value is a comma separated list of corresponding role names. For example: group1=role1;group2=role2;group3=role3,role4,role5 You could also use whitespaces and quotes around group names to make this mapping more readable, for example:
	`dbms.security.ldap.authorization.group_to_role_mapping=\`
	`"cn=Neo4j Read Only,cn=users,dc=example,dc=com" = reader; \`
	`"cn=Neo4j Read-Write,cn=users,dc=example,dc=com" = publisher; \`
	`"cn=Neo4j Schema Manager,cn=users,dc=example,dc=com" = architect; \`
	`"cn=Neo4j Administrator,cn=users,dc=example,dc=com" = admin`

Valid values	dbms.security.ldap.authorization.group_to_role_mapping, a string

Table 328. dbms.security.ldap.authorization.system_password

Description	An LDAP system account password to use for authorization searches when `dbms.security.ldap.authorization.use_system_account` is `true`.
Valid values	dbms.security.ldap.authorization.system_password, a secure string

Table 329. dbms.security.ldap.authorization.system_username

Description	An LDAP system account username to use for authorization searches when `dbms.security.ldap.authorization.use_system_account` is `true`. Note that the `dbms.security.ldap.authentication.user_dn_template` will not be applied to this username, so you may have to specify a full DN.
Valid values	dbms.security.ldap.authorization.system_username, a string

Table 330. dbms.security.ldap.authorization.use_system_account

Description	Perform LDAP search for authorization info using a system account instead of the user's own account. If this is set to `false` (default), the search for group membership will be performed directly after authentication using the LDAP context bound with the user's own account. The mapped roles will be cached for the duration of `dbms.security.auth_cache_ttl`, and then expire, requiring re-authentication. To avoid frequently having to re-authenticate sessions you may want to set a relatively long auth cache expiration time together with this option. NOTE: This option will only work if the users are permitted to search for their own group membership attributes in the directory. If this is set to `true`, the search will be performed using a special system account user with read access to all the users in the directory. You need to specify the username and password using the settings `dbms.security.ldap.authorization.system_username` and `dbms.security.ldap.authorization.system_password` with this option. Note that this account only needs read access to the relevant parts of the LDAP directory and does not need to have access rights to Neo4j, or any other systems.
Valid values	dbms.security.ldap.authorization.use_system_account, a boolean
Default value	`false`

Table 331. dbms.security.ldap.authorization.user_search_base

Description	The name of the base object or named context to search for user objects when LDAP authorization is enabled. A common case is that this matches the last part of `dbms.security.ldap.authentication.user_dn_template`.
Valid values	dbms.security.ldap.authorization.user_search_base, a string
Default value	`ou=users,dc=example,dc=com`

Table 332. dbms.security.ldap.authorization.user_search_filter

Description	The LDAP search filter to search for a user principal when LDAP authorization is enabled. The filter should contain the placeholder token \{0} which will be substituted for the user principal.
Valid values	dbms.security.ldap.authorization.user_search_filter, a string
Default value	`(&(objectClass=*)(uid={0}))`

Table 333. dbms.security.ldap.connection_timeout

Description	The timeout for establishing an LDAP connection. If a connection with the LDAP server cannot be established within the given time the attempt is aborted. A value of 0 means to use the network protocol's (i.e., TCP's) timeout value.

Valid values	dbms.security.ldap.connection_timeout, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`30s`

Table 334. dbms.security.ldap.host

Description	URL of LDAP server to use for authentication and authorization. The format of the setting is `<protocol>://<hostname>:<port>`, where hostname is the only required field. The supported values for protocol are `ldap` (default) and `ldaps`. The default port for `ldap` is 389 and for `ldaps` 636. For example: `ldaps://ldap.example.com:10389`. You may want to consider using STARTTLS (`dbms.security.ldap.use_starttls`) instead of LDAPS for secure connections, in which case the correct protocol is `ldap`.
Valid values	dbms.security.ldap.host, a string
Default value	`localhost`

Table 335. dbms.security.ldap.read_timeout

Description	The timeout for an LDAP read request (i.e. search). If the LDAP server does not respond within the given time the request will be aborted. A value of 0 means wait for a response indefinitely.
Valid values	dbms.security.ldap.read_timeout, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`30s`

Table 336. dbms.security.ldap.referral

Description	The LDAP referral behavior when creating a connection. This is one of `follow`, `ignore` or `throw`. * `follow` automatically follows any referrals * `ignore` ignores any referrals * `throw` throws an exception, which will lead to authentication failure.
Valid values	dbms.security.ldap.referral, a string
Default value	`follow`

Table 337. dbms.security.ldap.use_starttls

Description	Use secure communication with the LDAP server using opportunistic TLS. First an initial insecure connection will be made with the LDAP server, and a STARTTLS command will be issued to negotiate an upgrade of the connection to TLS before initiating authentication.

Valid values	dbms.security.ldap.use_starttls, a boolean
Default value	`false`

Table 338. dbms.security.log_successful_authentication

Description	Set to log successful authentication events to the security log. If this is set to `false` only failed authentication events will be logged, which could be useful if you find that the successful events spam the logs too much, and you do not require full auditing capability.
Valid values	dbms.security.log_successful_authentication, a boolean
Default value	`true`

Table 339. dbms.security.procedures.allowlist

Description	A list of procedures (comma separated) that are to be loaded. The list may contain both fully-qualified procedure names, and partial names with the wildcard '*'. If this setting is left empty no procedures will be loaded.
Valid values	dbms.security.procedures.allowlist, a ',' separated list with elements of type 'a string'.
Default value	`*`

Table 340. dbms.security.procedures.default_allowed

Description	The default role that can execute all procedures and user-defined functions that are not covered by the `dbms.security.procedures.roles` setting. This setting (if not empty string) will be translated to 'GRANT EXECUTE BOOSTED PROCEDURE *' and 'GRANT EXECUTE BOOSTED FUNCTION *' for that role. If `dbms.security.procedures.roles`is not empty, any procedure or function that this role is not mapped to will result in a 'DENY EXECUTE BOOSTED PROCEDURE name' and 'DENY EXECUTE BOOSTED FUNCTION name' for this role. Any privilege mapped in this way cannot be revoked, instead the config must be changed and will take effect after a restart.Deprecated: Replaced by EXECUTE PROCEDURE, EXECUTE BOOSTED PROCEDURE, EXECUTE FUNCTION and EXECUTE BOOSTED FUNCTION privileges.
Valid values	dbms.security.procedures.default_allowed, a string
Default value	
Deprecated	The `dbms.security.procedures.default_allowed` configuration setting has been deprecated.

Table 341. dbms.security.procedures.roles

Description	This provides a finer level of control over which roles can execute procedures than the `dbms.security.procedures.default_allowed` setting. For example: `dbms.security.procedures.roles=apoc.convert.*:reader;apoc.load.json*:writer;apoc.trigger.add:TriggerHappy` will allow the role `reader` to execute all procedures in the `apoc.convert` namespace, the role `writer` to execute all procedures in the `apoc.load` namespace that starts with `json` and the role `TriggerHappy` to execute the specific procedure `apoc.trigger.add`. Procedures not matching any of these patterns will be subject to the `dbms.security.procedures.default_allowed` setting. This setting (if not empty string) will be translated to 'GRANT EXECUTE BOOSTED PROCEDURE name' and 'GRANT EXECUTE BOOSTED FUNCTION name' privileges for the mapped roles. Any privilege mapped in this way cannot be revoked, instead the config must be changed and will take effect after a restart.Deprecated: Replaced by EXECUTE PROCEDURE, EXECUTE BOOSTED PROCEDURE, EXECUTE FUNCTION and EXECUTE BOOSTED FUNCTION privileges.
Valid values	dbms.security.procedures.roles, a string
Default value	
Deprecated	The `dbms.security.procedures.roles` configuration setting has been deprecated.

Table 342. dbms.security.procedures.unrestricted

Description	A list of procedures and user defined functions (comma separated) that are allowed full access to the database. The list may contain both fully-qualified procedure names, and partial names with the wildcard '*'. Note that this enables these procedures to bypass security. Use with caution.
Valid values	dbms.security.procedures.unrestricted, a ',' separated list with elements of type 'a string'.
Default value	

Table 343. 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. Deprecated, use dbms.security.procedures.allowlist
Valid values	dbms.security.procedures.whitelist, a ',' separated list with elements of type 'a string'.
Default value	*

Deprecated	The `dbms.security.procedures.whitelist` configuration setting has been deprecated.

Table 344. dbms.shutdown_transaction_end_timeout

Description	The maximum amount of time to wait for running transactions to complete before allowing initiated database shutdown to continue.
Valid values	dbms.shutdown_transaction_end_timeout, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`10s`

Table 345. dbms.threads.worker_count

Description	Number of Neo4j worker threads. This setting is only valid for REST, and does not influence bolt-server. It sets the amount of worker threads for the Jetty server used by neo4j-server. This option can be tuned when you plan to execute multiple, concurrent REST requests, with the aim of getting more throughput from the database. Your OS might enforce a lower limit than the maximum value specified here.
Valid values	dbms.threads.worker_count, an integer which is in the range `1` to `44738`
Default value	`Number of available processors, or 500 for machines which have more than 500 processors.`

Table 346. dbms.track_query_allocation

Description	Enables or disables tracking of how many bytes are allocated by the execution of a query. If enabled, calling `dbms.listQueries` will display the allocated bytes. This can also be logged in the query log by using `dbms.logs.query.allocation_logging_enabled`.
Valid values	dbms.track_query_allocation, a boolean
Dynamic	true
Default value	`true`

Table 347. dbms.track_query_cpu_time

Description	Enables or disables tracking of how much time a query spends actively executing on the CPU. Calling `SHOW TRANSACTIONS` will display the time, but not in the *query.log*. If you want the CPU time to be logged in the *query.log*, set `db.track_query_cpu_time=true` and `db.logs.query.time_logging_enabled=true` Enterprise.
Valid values	dbms.track_query_cpu_time, a boolean
Dynamic	true
Default value	`false`

Table 348. dbms.transaction.bookmark_ready_timeout

Description	The maximum amount of time to wait for the database state represented by the bookmark.
Valid values	dbms.transaction.bookmark_ready_timeout, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's') which is minimum `1s`
Default value	`30s`

Table 349. dbms.transaction.concurrent.maximum

Description	The maximum number of concurrently running transactions. If set to 0, limit is disabled.
Valid values	dbms.transaction.concurrent.maximum, an integer
Dynamic	true
Default value	`1000`

Table 350. dbms.transaction.monitor.check.interval

Description	Configures the time interval between transaction monitor checks. Determines how often monitor thread will check transaction for timeout.
Valid values	dbms.transaction.monitor.check.interval, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`2s`

Table 351. dbms.transaction.sampling.percentage

Description	Transaction sampling percentage.
Valid values	dbms.transaction.sampling.percentage, an integer which is in the range `1` to `100`
Dynamic	true
Default value	`5`

Table 352. dbms.transaction.timeout

Description	The maximum time interval of a transaction within which it should be completed.
Valid values	dbms.transaction.timeout, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Dynamic	true
Default value	`0s`

Table 353. dbms.transaction.tracing.level

Description	Transaction creation tracing level.
Valid values	dbms.transaction.tracing.level, one of [DISABLED, SAMPLE, ALL]
Dynamic	true
Default value	`DISABLED`

Table 354. dbms.tx_log.preallocate

Description	Specify if Neo4j should try to preallocate the logical log file in advance. It optimizes the filesystem by ensuring there is room to accommodate newly generated files and avoid file-level fragmentation.
Valid values	dbms.tx_log.preallocate, a boolean
Dynamic	true
Default value	`true`

Table 355. dbms.tx_log.rotation.retention_policy

Description	Tell Neo4j how long logical transaction logs should be kept to backup the database.For example, "10 days" will prune logical logs that only contain transactions older than 10 days.Alternatively, "100k txs" will keep the 100k latest transactions from each database and prune any older transactions.
Valid values	dbms.tx_log.rotation.retention_policy, a string which matches the pattern `^(true|keep_all|false|keep_none|(\d+[KkMmGg]?( (files|size|txs|entries|hours|days))))$` (Must be `true` or `keep_all`, `false` or `keep_none`, or of format `<number><optional unit> <type>`. Valid units are `K`, `M` and `G`. Valid types are `files`, `size`, `txs`, `entries`, `hours` and `days`. For example, `100M size` will limit logical log space on disk to 100MB per database,and `200K txs` will limit the number of transactions kept to 200 000 per database.)
Dynamic	true
Default value	`7 days`

Table 356. dbms.tx_log.rotation.size

Description	Specifies at which file size the logical log will auto-rotate. Minimum accepted value is 128 KiB.
Valid values	dbms.tx_log.rotation.size, a byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `TiB`, `TB`, `PiB`, `PB`, `EiB`, `EB`) which is minimum `128.00KiB`
Dynamic	true
Default value	`250.00MiB`

Table 357. dbms.tx_state.memory_allocation

Description	Defines whether memory for transaction state should be allocated on- or off-heap. Note that for small transactions you can gain up to 25% write speed by setting it to `ON_HEAP`.
Valid values	dbms.tx_state.memory_allocation, one of [ON_HEAP, OFF_HEAP]
Default value	`OFF_HEAP`

Table 358. dbms.unmanaged_extension_classes

Description	Comma-separated list of <classname>=<mount point> for unmanaged extensions.

Valid values	dbms.unmanaged_extension_classes, a ',' separated list with elements of type '<classname>=<mount point> string'.
Default value	

Table 359. dbms.upgrade_max_processors

Description	Max number of processors used when upgrading the store. Defaults to the number of processors available to the JVM. There is a certain amount of minimum threads needed so for that reason there is no lower bound for this value. For optimal performance this value shouldn't be greater than the number of available processors.
Valid values	dbms.upgrade_max_processors, an integer which is minimum `0`
Dynamic	true
Default value	`0`

Table 360. dbms.windows_service_name

Description	Name of the Windows Service.
Valid values	dbms.windows_service_name, a string
Default value	`neo4j`

Table 361. fabric.database.name

Description	Name of the Fabric database. Only one Fabric database is currently supported per Neo4j instance.
Valid values	fabric.database.name, A valid database name. Containing only alphabetic characters, numbers, dots and dashes, with a length between 3 and 63 characters. It should be starting with an alphabetic character but not with the name 'system'.

Table 362. fabric.driver.api

Description	Determines which driver API will be used. ASYNC must be used when the remote instance is 3.5.
Valid values	fabric.driver.api, one of [RX, ASYNC]
Default value	`RX`

Table 363. fabric.driver.connection.connect_timeout

Description	Socket connection timeout. A timeout of zero is treated as an infinite timeout and will be bound by the timeout configured on the operating system level.
Valid values	fabric.driver.connection.connect_timeout, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`5s`

Table 364. fabric.driver.connection.max_lifetime

Description	Pooled connections older than this threshold will be closed and removed from the pool. Setting this option to a low value will cause a high connection churn and might result in a performance hit. It is recommended to set maximum lifetime to a slightly smaller value than the one configured in network equipment (load balancer, proxy, firewall, etc. can also limit maximum connection lifetime). Zero and negative values result in lifetime not being checked.
Valid values	fabric.driver.connection.max_lifetime, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`1h`

Table 365. fabric.driver.connection.pool.acquisition_timeout

Description	Maximum amount of time spent attempting to acquire a connection from the connection pool. This timeout only kicks in when all existing connections are being used and no new connections can be created because maximum connection pool size has been reached. Error is raised when connection can't be acquired within configured time. Negative values are allowed and result in unlimited acquisition timeout. Value of 0 is allowed and results in no timeout and immediate failure when connection is unavailable.
Valid values	fabric.driver.connection.pool.acquisition_timeout, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`1m`

Table 366. fabric.driver.connection.pool.idle_test

Description	Pooled connections that have been idle in the pool for longer than this timeout will be tested before they are used again, to ensure they are still alive. If this option is set too low, an additional network call will be incurred when acquiring a connection, which causes a performance hit. If this is set high, no longer live connections might be used which might lead to errors. Hence, this parameter tunes a balance between the likelihood of experiencing connection problems and performance Normally, this parameter should not need tuning. Value 0 means connections will always be tested for validity.
Valid values	fabric.driver.connection.pool.idle_test, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`No connection liveliness check is done by default.`

Table 367. fabric.driver.connection.pool.max_size

Description	Maximum total number of connections to be managed by a connection pool. The limit is enforced for a combination of a host and user. Negative values are allowed and result in unlimited pool. Value of 0is not allowed.
Valid values	fabric.driver.connection.pool.max_size, an integer
Default value	`Unlimited`

Table 368. fabric.driver.logging.level

Description	Sets level for driver internal logging.
Valid values	fabric.driver.logging.level, one of [DEBUG, INFO, WARN, ERROR, NONE]
Default value	`Value of dbms.logs.debug.level`

Table 369. fabric.routing.servers

Description	A comma-separated list of Fabric instances that form a routing group. A driver will route transactions to available routing group members. A Fabric instance is represented by its Bolt connector address.
Valid values	fabric.routing.servers, a ',' separated list with elements of type 'a socket address'.
Dynamic	true

Table 370. fabric.routing.ttl

Description	The time to live (TTL) of a routing table for fabric routing group.

Valid values	fabric.routing.ttl, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`1m`

Table 371. fabric.stream.buffer.low_watermark

Description	Number of records in prefetching buffer that will trigger prefetching again. This is strongly related to fabric.stream.buffer.size
Valid values	fabric.stream.buffer.low_watermark, an integer which is minimum `0`
Default value	`300`

Table 372. fabric.stream.buffer.size

Description	Maximal size of a buffer used for pre-fetching result records of remote queries. To compensate for latency to remote databases, the Fabric execution engine pre-fetches records needed for local executions. This limit is enforced per fabric query. If a fabric query uses multiple remote stream at the same time, this setting represents the maximal number of pre-fetched records counted together for all such remote streams.
Valid values	fabric.stream.buffer.size, an integer which is minimum `1`
Default value	`1000`

Table 373. fabric.stream.concurrency

Description	Maximal concurrency within Fabric queries. Limits the number of iterations of each subquery that are executed concurrently. Higher concurrency may consume more memory and network resources simultaneously, while lower concurrency may force sequential execution, requiring more time.
Valid values	fabric.stream.concurrency, an integer which is minimum `1`
Default value	`The number of remote graphs`

Table 374. metrics.bolt.messages.enabled

Description	Enable reporting metrics about Bolt Protocol message processing. Deprecated - use metrics.filter instead.
Valid values	metrics.bolt.messages.enabled, a boolean
Default value	`false`

Deprecated	The `metrics.bolt.messages.enabled` configuration setting has been deprecated.

Table 375. metrics.csv.enabled

Description	Set to `true` to enable exporting metrics to CSV files.
Valid values	metrics.csv.enabled, a boolean
Default value	`true`

Table 376. metrics.csv.interval

Description	The reporting interval for the CSV files. That is, how often new rows with numbers are appended to the CSV files.
Valid values	metrics.csv.interval, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`30s`

Table 377. metrics.csv.rotation.compression

Description	Decides what compression to use for the csv history files.
Valid values	metrics.csv.rotation.compression, one of [NONE, ZIP, GZ]
Default value	`NONE`

Table 378. metrics.csv.rotation.keep_number

Description	Maximum number of history files for the csv files.
Valid values	metrics.csv.rotation.keep_number, an integer which is minimum `1`
Default value	`7`

Table 379. metrics.csv.rotation.size

Description	The file size in bytes at which the csv files will auto-rotate. If set to zero then no rotation will occur. Accepts a binary suffix `k`, `m` or `g`.
Valid values	metrics.csv.rotation.size, a byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `TiB`, `TB`, `PiB`, `PB`, `EiB`, `EB`) which is in the range `0B` to `8388608.00TiB`
Default value	`10.00MiB`

Table 380. metrics.cypher.replanning.enabled

Description	Enable reporting metrics about number of occurred replanning events. Deprecated - use metrics.filter instead.
Valid values	metrics.cypher.replanning.enabled, a boolean
Default value	`false`
Deprecated	The `metrics.cypher.replanning.enabled` configuration setting has been deprecated.

Table 381. metrics.enabled

Description	Enable metrics. Setting this to `false` will to turn off all metrics.
Valid values	metrics.enabled, a boolean
Default value	`true`

Table 382. metrics.filter

Description	Specifies which metrics should be enabled by using a comma separated list of globbing patterns. Only the metrics matching the filter will be enabled. For example '**check_point**,neo4j.page_cache.evictions' will enable any checkpoint metrics and the pagecache eviction metric.
Valid values	metrics.filter, a ',' separated list with elements of type 'A simple globbing pattern that can use '*' and '?'.'.
Default value	`*bolt.connections*,*bolt.messages_received*,*bolt.messages_started*,*dbms.pool.bolt.free,*dbms.pool.bolt.total_size,*dbms.pool.bolt.total_used,*dbms.pool.bolt.used_heap,*causal_clustering.core.is_leader,*causal_clustering.core.last_leader_message,*causal_clustering.core.replication_attempt,*causal_clustering.core.replication_fail,*check_point.duration,*check_point.total_time,*cypher.replan_events,*ids_in_use*,*pool.transaction.*.total_used,*pool.transaction.*.used_heap,*pool.transaction.*.used_native,*store.size*,*transaction.active_read,*transaction.active_write,*transaction.committed*,*transaction.last_committed_tx_id,*transaction.peak_concurrent,*transaction.rollbacks*,*page_cache.hit*,*page_cache.page_faults,*page_cache.usage_ratio,*vm.file.descriptors.count,*vm.gc.time.*,*vm.heap.used,*vm.memory.buffer.direct.used,*vm.memory.pool.g1_eden_space,*vm.memory.pool.g1_old_gen,*vm.pause_time,*vm.thread*`

Table 383. metrics.graphite.enabled

Description	Set to `true` to enable exporting metrics to Graphite.
Valid values	metrics.graphite.enabled, a boolean
Default value	`false`

Table 384. metrics.graphite.interval

Description	The reporting interval for Graphite. That is, how often to send updated metrics to Graphite.
Valid values	metrics.graphite.interval, a duration (Valid units are: 'ns', 'µs', 'ms', 's', 'm', 'h' and 'd'; default unit is 's')
Default value	`30s`

Table 385. metrics.graphite.server

Description	The hostname or IP address of the Graphite server.
Valid values	metrics.graphite.server, a socket address. If missing port or hostname it is acquired from dbms.default_listen_address
Default value	`:2003`

Table 386. metrics.jmx.enabled

Description	Set to `true` to enable the JMX metrics endpoint.
Valid values	metrics.jmx.enabled, a boolean
Default value	`true for 4.2.2 and later, false on version 4.2.0 & 4.2.1`

Table 387. metrics.jvm.buffers.enabled

Description	Enable reporting metrics about the buffer pools. Deprecated - use metrics.filter instead.
Valid values	metrics.jvm.buffers.enabled, a boolean
Default value	`false`
Deprecated	The `metrics.jvm.buffers.enabled` configuration setting has been deprecated.

Table 388. metrics.jvm.file.descriptors.enabled

Description	Enable reporting metrics about the number of open file descriptors. Deprecated - use metrics.filter instead.
Valid values	metrics.jvm.file.descriptors.enabled, a boolean
Default value	`false`

Deprecated	The `metrics.jvm.file.descriptors.enabled` configuration setting has been deprecated.

Table 389. metrics.jvm.gc.enabled

Description	Enable reporting metrics about the duration of garbage collections. Deprecated - use metrics.filter instead.
Valid values	metrics.jvm.gc.enabled, a boolean
Default value	`false`
Deprecated	The `metrics.jvm.gc.enabled` configuration setting has been deprecated.

Table 390. metrics.jvm.heap.enabled

Description	Enable reporting metrics about the heap memory usage. Deprecated - use metrics.filter instead.
Valid values	metrics.jvm.heap.enabled, a boolean
Default value	`false`
Deprecated	The `metrics.jvm.heap.enabled` configuration setting has been deprecated.

Table 391. metrics.jvm.memory.enabled

Description	Enable reporting metrics about the memory usage. Deprecated - use metrics.filter instead.
Valid values	metrics.jvm.memory.enabled, a boolean
Default value	`false`
Deprecated	The `metrics.jvm.memory.enabled` configuration setting has been deprecated.

Table 392. metrics.jvm.pause_time.enabled

Description	Enable reporting metrics about the VM pause time. Deprecated - use metrics.filter instead.
Valid values	metrics.jvm.pause_time.enabled, a boolean
Default value	`false`

Deprecated	The `metrics.jvm.pause_time.enabled` configuration setting has been deprecated.

Table 393. metrics.jvm.threads.enabled

Description	Enable reporting metrics about the current number of threads running. Deprecated - use metrics.filter instead.
Valid values	metrics.jvm.threads.enabled, a boolean
Default value	`false`
Deprecated	The `metrics.jvm.threads.enabled` configuration setting has been deprecated.

Table 394. metrics.namespaces.enabled

Description	Enable metrics namespaces that separates the global and database specific metrics. If enabled all database specific metrics will have field names starting with <metrics_prefix>.database.<database_name> and all global metrics will start with <metrics_prefix>.dbms. For example neo4j.page_cache.hits will become neo4j.dbms.page_cache.hits and neo4j.system.log.rotation_events will become neo4j.database.system.log.rotation_events.
Valid values	metrics.namespaces.enabled, a boolean
Default value	`false`

Table 395. metrics.neo4j.causal_clustering.enabled

Description	Enable reporting metrics about Causal Clustering mode. Deprecated - use metrics.filter instead.
Valid values	metrics.neo4j.causal_clustering.enabled, a boolean
Default value	`false`
Deprecated	The `metrics.neo4j.causal_clustering.enabled` configuration setting has been deprecated.

Table 396. metrics.neo4j.checkpointing.enabled

Description	Enable reporting metrics about Neo4j check pointing; when it occurs and how much time it takes to complete. Deprecated - use metrics.filter instead.
Valid values	metrics.neo4j.checkpointing.enabled, a boolean

Default value	`false`
Deprecated	The `metrics.neo4j.checkpointing.enabled` configuration setting has been deprecated.

Table 397. metrics.neo4j.counts.enabled

Description	Enable reporting metrics about approximately how many entities are in the database; nodes, relationships, properties, etc. Deprecated - use metrics.filter instead.
Valid values	metrics.neo4j.counts.enabled, a boolean
Default value	`false`
Deprecated	The `metrics.neo4j.counts.enabled` configuration setting has been deprecated.

Table 398. metrics.neo4j.data.counts.enabled

Description	Enable reporting metrics about number of entities in the database. Deprecated - use metrics.filter instead.
Valid values	metrics.neo4j.data.counts.enabled, a boolean
Default value	`false`
Deprecated	The `metrics.neo4j.data.counts.enabled` configuration setting has been deprecated.

Table 399. metrics.neo4j.database_operation_count.enabled

Description	Enable reporting metrics for Neo4j dbms operations; how many times databases have been created, started, stopped or dropped, and how many attempted operations have failed and recovered later. Deprecated - use metrics.filter instead.
Valid values	metrics.neo4j.database_operation_count.enabled, a boolean
Default value	`false`
Deprecated	The `metrics.neo4j.database_operation_count.enabled` configuration setting has been deprecated.

Table 400. metrics.neo4j.logs.enabled

Description	Enable reporting metrics about the Neo4j transaction logs. Deprecated - use metrics.filter instead.
Valid values	metrics.neo4j.logs.enabled, a boolean
Default value	`false`
Deprecated	The `metrics.neo4j.logs.enabled` configuration setting has been deprecated.

Table 401. metrics.neo4j.pagecache.enabled

Description	Enable reporting metrics about the Neo4j page cache; page faults, evictions, flushes, exceptions, etc. Deprecated - use metrics.filter instead.
Valid values	metrics.neo4j.pagecache.enabled, a boolean
Default value	`false`
Deprecated	The `metrics.neo4j.pagecache.enabled` configuration setting has been deprecated.

Table 402. metrics.neo4j.pools.enabled

Description	Enable reporting metrics about Neo4j memory pools. Deprecated - use metrics.filter instead.
Valid values	metrics.neo4j.pools.enabled, a boolean
Default value	`false`
Deprecated	The `metrics.neo4j.pools.enabled` configuration setting has been deprecated.

Table 403. metrics.neo4j.server.enabled

Description	Enable reporting metrics about Server threading info. Deprecated - use metrics.filter instead.
Valid values	metrics.neo4j.server.enabled, a boolean
Default value	`false`
Deprecated	The `metrics.neo4j.server.enabled` configuration setting has been deprecated.

Table 404. metrics.neo4j.size.enabled

Description	Enable reporting metrics about the store size of each database. Deprecated - use metrics.filter instead.
Valid values	metrics.neo4j.size.enabled, a boolean
Default value	`false`
Deprecated	The `metrics.neo4j.size.enabled` configuration setting has been deprecated.

Table 405. metrics.neo4j.tx.enabled

Description	Enable reporting metrics about transactions; number of transactions started, committed, etc. Deprecated - use metrics.filter instead.
Valid values	metrics.neo4j.tx.enabled, a boolean
Default value	`false`
Deprecated	The `metrics.neo4j.tx.enabled` configuration setting has been deprecated.

Table 406. metrics.prefix

Description	A common prefix for the reported metrics field names.
Valid values	metrics.prefix, a string
Default value	`neo4j`

Table 407. metrics.prometheus.enabled

Description	Set to `true` to enable the Prometheus endpoint.
Valid values	metrics.prometheus.enabled, a boolean
Default value	`false`

Table 408. metrics.prometheus.endpoint

Description	The hostname and port to use as Prometheus endpoint.
Valid values	metrics.prometheus.endpoint, a socket address in the format `hostname:port`, `hostname`, or `:port`. If missing, port and hostname are acquired from `dbms.default_listen_address`.
Default value	`localhost:2004`

16.A.2. Procedures

Procedures, editions and modes

Available procedures depends on the type of installation you have:

- Neo4j Enterprise Edition provides a larger set of procedures than Neo4j Community Edition.
- Cluster members have procedures that are not available in standalone mode.

To check which procedures are available in your Neo4j instance, use `dbms.procedures()`.

Example 126. List available procedures

To list the procedures available on your particular installation, run the following Cypher query:

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

List of procedures

Table 409. Neo4j procedures

Name	
db.awaitIndex()	
db.awaitIndexes()	
db.checkpoint()	
db.clearQueryCaches()	
db.constraints()	
db.createIndex()	
db.createLabel()	
db.createNodeKey()	
db.createProperty()	
db.createRelationshipType()	
db.createUniquePropertyConstraint()	
db.index.fulltext.awaitEventuallyConsistentIndexRefresh()	
db.index.fulltext.createNodeIndex()	
db.index.fulltext.createRelationshipIndex()	
db.index.fulltext.drop()	
db.index.fulltext.listAvailableAnalyzers()	
db.index.fulltext.queryNodes()	
db.index.fulltext.queryRelationships()	
db.indexDetails()	

Name	
db.indexes()	
db.info()	
db.labels()	
db.listLocks()	
db.ping()	
db.prepareForReplanning()	
db.propertyKeys()	
db.relationshipTypes()	
db.resampleIndex()	
db.resampleOutdatedIndexes()	
db.schema.nodeTypeProperties()	
db.schema.relTypeProperties()	
db.schema.visualization()	
db.schemaStatements()	
db.stats.clear()	
db.stats.collect()	
db.stats.retrieve()	
db.stats.retrieveAllAnonymized()	
db.stats.status()	
db.stats.stop()	
dbms.cluster.routing.getRoutingTable()	
dbms.cluster.overview()	
dbms.cluster.protocols()	
dbms.cluster.quarantineDatabase()	
dbms.cluster.readReplicaToggle()	
dbms.cluster.role()	
dbms.cluster.setDefaultDatabase()	
dbms.components()	
dbms.database.state()	
dbms.functions()	
dbms.info()	
dbms.killConnection()	
dbms.killConnections()	
dbms.killQueries()	
dbms.killQuery()	
dbms.killTransaction()	

Name	
dbms.killTransactions()	
dbms.listActiveLocks()	
dbms.listConfig()	
dbms.listConnections()	
dbms.listPools()	
dbms.listQueries()	
dbms.listTransactions()	
dbms.procedures()	
dbms.queryJmx()	
dbms.routing.getRoutingTable()	
dbms.scheduler.failedJobs()	
dbms.scheduler.groups()	
dbms.scheduler.jobs()	
dbms.scheduler.profile()	
dbms.security.activateUser()	
dbms.security.addRoleToUser()	
dbms.security.changePassword()	
dbms.security.changeUserPassword()	
dbms.security.clearAuthCache()	
dbms.security.createRole()	
dbms.security.createUser()	
dbms.security.deleteRole()	
dbms.security.deleteUser()	
dbms.security.listRoles()	
dbms.security.listRolesForUser()	
dbms.security.listUsers()	
dbms.security.listUsersForRole()	
dbms.security.removeRoleFromUser()	
dbms.security.suspendUser()	
dbms.setConfigValue()	
dbms.showCurrentUser()	
dbms.upgrade()	
dbms.upgradeStatus()	
tx.getMetaData()	

Name	
`tx.setMetaData()`	

Procedure descriptions

The role-based access control is an Enterprise Edition feature.

Each procedure lists the default roles that can access the procedure.

Table 410. db.awaitIndex()

Description	Wait for an index to come online. Example: `CALL db.awaitIndex("MyIndex", 300)`
Signature	`db.awaitIndex(indexName :: STRING?, timeOutSeconds = 300 :: INTEGER?) :: VOID`
Mode	`READ`
Default roles	`reader, editor, publisher, architect, admin`

Table 411. db.awaitIndexes()

Description	Wait for all indexes to come online. Example: `CALL db.awaitIndexes(300))`
Signature	`db.awaitIndexes(timeOutSeconds = 300 :: INTEGER?) :: VOID`
Mode	`READ`
Default roles	`reader, editor, publisher, architect, admin`

Table 412. db.checkpoint()

Description	Initiate and wait for a new check point, or wait any already on-going check point to complete. Note that this temporarily disables the `dbms.checkpoint.iops.limit` setting in order to make the check point complete faster. This might cause transaction throughput to degrade slightly, due to increased IO load.
Signature	`db.checkpoint() :: (success :: BOOLEAN?, message :: STRING?)`
Mode	`DBMS`
Default roles	`reader, editor, publisher, architect, admin`

Table 413. db.clearQueryCaches()

Description	Clears all query caches.

Signature	`db.clearQueryCaches() :: (value :: STRING?)`
Mode	`DBMS`
Default roles	`admin`

Table 414. db.constraints()

Description	List all constraints in the database.
Signature	`db.constraints() :: (name :: STRING?, description :: STRING?, details :: STRING?)`
Mode	`READ`
Default roles	`reader, editor, publisher, architect, admin`

Table 415. db.createIndex()

Description	Create a named schema index with specified index provider and configuration (optional). Yield: name, labels, properties, providerName, status
Signature	`db.createIndex(indexName :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, config = {} :: MAP?) :: (name :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, status :: STRING?)`
Mode	`SCHEMA`
Default roles	`architect, admin`

Table 416. db.createLabel()

Description	Create a label
Signature	`db.createLabel(newLabel :: STRING?) :: VOID`
Mode	`WRITE`
Default roles	`publisher, architect, admin`

Table 417. db.createNodeKey()

Description	Create a named node key constraint. Backing index will use specified index provider and configuration (optional). Yield: name, labels, properties, providerName, status
Signature	`db.createNodeKey(constraintName :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, config = {} :: MAP?) :: (name :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, status :: STRING?)`
Mode	`SCHEMA`
Default roles	`architect, admin`

Table 418. db.createProperty()

Description	Create a Property
Signature	`db.createProperty(newProperty :: STRING?) :: VOID`
Mode	`WRITE`
Default roles	`publisher, architect, admin`

Table 419. db.createRelationshipType()

Description	Create a RelationshipType
Signature	`db.createRelationshipType(newRelationshipType :: STRING?) :: VOID`
Mode	`WRITE`
Default roles	`publisher, architect, admin`

Table 420. db.createUniquePropertyConstraint()

Description	Create a named unique property constraint. Backing index will use specified index provider and configuration (optional). Yield: name, labels, properties, providerName, status
Signature	`db.createUniquePropertyConstraint(constraintName :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, config = {} :: MAP?) :: (name :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, status :: STRING?)`
Mode	`SCHEMA`
Default roles	`architect, admin`

Table 421. db.index.fulltext.awaitEventuallyConsistentIndexRefresh()

Description	Wait for the updates from recently committed transactions to be applied to any eventually-consistent full-text indexes.
Signature	`db.index.fulltext.awaitEventuallyConsistentIndexRefresh() :: VOID`
Mode	`READ`
Default roles	`reader, editor, publisher, architect, admin`

Table 422. db.index.fulltext.createNodeIndex()

Description	Create a node full-text index for the given labels and properties. The optional 'config' map parameter can be used to supply settings to the index. Supported settings are 'analyzer', for specifying what analyzer to use when indexing and querying. Use the `db.index.fulltext.listAvailableAnalyzers` procedure to see what options are available. And 'eventually_consistent' which can be set to 'true' to make this index eventually consistent, such that updates from committing transactions are applied in a background thread.
Signature	`db.index.fulltext.createNodeIndex(indexName :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, config = {} :: MAP?) :: VOID`
Mode	`SCHEMA`
Default roles	`architect, admin`

Table 423. db.index.fulltext.createRelationshipIndex()

Description	Create a relationship full-text index for the given relationship types and properties. The optional 'config' map parameter can be used to supply settings to the index. Supported settings are 'analyzer', for specifying what analyzer to use when indexing and querying. Use the `db.index.fulltext.listAvailableAnalyzers` procedure to see what options are available. And 'eventually_consistent' which can be set to 'true' to make this index eventually consistent, such that updates from committing transactions are applied in a background thread.
Signature	`db.index.fulltext.createRelationshipIndex(indexName :: STRING?, relationshipTypes :: LIST? OF STRING?, properties :: LIST? OF STRING?, config = {} :: MAP?) :: VOID`
Mode	`SCHEMA`
Default roles	`architect, admin`

Table 424. db.index.fulltext.drop()

Description	Drop the specified index.
Signature	`db.index.fulltext.drop(indexName :: STRING?) :: VOID`
Mode	`SCHEMA`
Default roles	`architect, admin`

Table 425. db.index.fulltext.listAvailableAnalyzers()

Description	List the available analyzers that the full-text indexes can be configured with.
Signature	`db.index.fulltext.listAvailableAnalyzers() :: (analyzer :: STRING?, description :: STRING?, stopwords :: LIST? OF STRING?)`
Mode	`READ`
Default roles	`reader, editor, publisher, architect, admin`

Table 426. db.index.fulltext.queryNodes()

Description	Query the given full-text index. Returns the matching nodes, and their Lucene query score, ordered by score. Valid keys for the options map are: 'skip' to skip the top N results; 'limit' to limit the number of results returned.
Signature	`db.index.fulltext.queryNodes(indexName :: STRING?, queryString :: STRING?, options = {} :: MAP?) :: (node :: NODE?, score :: FLOAT?)`
Mode	`READ`
Default roles	`reader, editor, publisher, architect, admin`

Table 427. db.index.fulltext.queryRelationships()

Description	Query the given full-text index. Returns the matching relationships, and their Lucene query score, ordered by score. Valid keys for the options map are: 'skip' to skip the top N results; 'limit' to limit the number of results returned.
Signature	`db.index.fulltext.queryRelationships(indexName :: STRING?, queryString :: STRING?, options = {} :: MAP?) :: (relationship :: RELATIONSHIP?, score :: FLOAT?)`
Mode	`READ`
Default roles	`reader, editor, publisher, architect, admin`

Table 428. db.indexDetails()

Description	Detailed description of specific index.
Signature	`db.indexDetails(indexName :: STRING?) :: (id :: INTEGER?, name :: STRING?, state :: STRING?, populationPercent :: FLOAT?, uniqueness :: STRING?, type :: STRING?, entityType :: STRING?, labelsOrTypes :: LIST? OF STRING?, properties :: LIST? OF STRING?, provider :: STRING?, indexConfig :: MAP?, failureMessage :: STRING?)`
Mode	`READ`
Default roles	`reader, editor, publisher, architect, admin`

Table 429. db.indexes()

Description	List all indexes in the database.
Signature	`db.indexes() :: (id :: INTEGER?, name :: STRING?, state :: STRING?, populationPercent :: FLOAT?, uniqueness :: STRING?, type :: STRING?, entityType :: STRING?, labelsOrTypes :: LIST? OF STRING?, properties :: LIST? OF STRING?, provider :: STRING?)`
Mode	`READ`
Default roles	`reader, editor, publisher, architect, admin`

Table 430. db.info()

Description	Provides information regarding the database.
Signature	`db.info() :: (id :: STRING?, name :: STRING?, creationDate :: STRING?)`
Mode	`READ`
Default roles	`reader, editor, publisher, architect, admin`

Table 431. db.labels()

Description	List all available labels in the database.
Signature	`db.labels() :: (label :: STRING?)`
Mode	`READ`
Default roles	`reader, editor, publisher, architect, admin`

Table 432. db.listLocks()

Description	List all locks at this database.
Signature	`db.listLocks() :: (mode :: STRING?, resourceType :: STRING?, resourceId :: INTEGER?, transactionId :: STRING?)`
Mode	`DBMS`
Default roles	`admin`

Table 433. db.ping()

Description	This procedure can be used by client side tooling to test whether they are correctly connected to a database. The procedure is available in all databases and always returns true. A faulty connection can be detected by not being able to call this procedure.
Signature	`db.ping() :: (success :: BOOLEAN?)`
Mode	`READ`
Default roles	`reader, editor, publisher, architect, admin`

Table 434. db.prepareForReplanning()

Description	Triggers an index resample and waits for it to complete, and after that clears query caches. After this procedure has finished queries will be planned using the latest database statistics.
Signature	`db.prepareForReplanning(timeOutSeconds = 300 :: INTEGER?) :: VOID`
Mode	`READ`
Default roles	`admin`

Table 435. db.propertyKeys()

Description	List all property keys in the database.
Signature	`db.propertyKeys() :: (propertyKey :: STRING?)`
Mode	`READ`
Default roles	`reader, editor, publisher, architect, admin`

Table 436. db.relationshipTypes()

Description	List all available relationship types in the database.
Signature	`db.relationshipTypes() :: (relationshipType :: STRING?)`
Mode	`READ`
Default roles	`reader, editor, publisher, architect, admin`

Table 437. db.resampleIndex()

Description	Schedule resampling of an index. Example: `CALL db.resampleIndex("MyIndex"))`
Signature	`db.resampleIndex(indexName :: STRING?) :: VOID`
Mode	`READ`
Default roles	`reader, editor, publisher, architect, admin`

Table 438. db.resampleOutdatedIndexes()

Description	Schedule resampling of all outdated indexes.
Signature	`db.resampleOutdatedIndexes() :: VOID`
Mode	`READ`
Default roles	`reader, editor, publisher, architect, admin`

Table 439. db.schema.nodeTypeProperties()

Description	Show the derived property schema of the nodes in tabular form.
Signature	`db.schema.nodeTypeProperties() :: (nodeType :: STRING?, nodeLabels :: LIST? OF STRING?, propertyName :: STRING?, propertyTypes :: LIST? OF STRING?, mandatory :: BOOLEAN?)`
Mode	`READ`
Default roles	`reader, editor, publisher, architect, admin`

Table 440. db.schema.relTypeProperties()

Description	Show the derived property schema of the relationships in tabular form.
Signature	`db.schema.relTypeProperties() :: (relType :: STRING?, propertyName :: STRING?, propertyTypes :: LIST? OF STRING?, mandatory :: BOOLEAN?)`
Mode	`READ`
Default roles	`reader, editor, publisher, architect, admin`

Table 441. db.schema.visualization()

Description	Visualize the schema of the data.
Signature	`db.schema.visualization() :: (nodes :: LIST? OF NODE?, relationships :: LIST? OF RELATIONSHIP?)`
Mode	`READ`
Default roles	`reader, editor, publisher, architect, admin`

Table 442. db.schemaStatements()

Description	List all statements for creating and dropping existing indexes and constraints.
Signature	`db.schemaStatements() :: (name :: STRING?, type :: STRING?, createStatement :: STRING?, dropStatement :: STRING?)`
Mode	`READ`
Default roles	`reader, editor, publisher, architect, admin`

Table 443. db.stats.clear()

Description	Clear collected data of a given data section. Valid sections are `'QUERIES'`
Signature	`db.stats.clear(section :: STRING?) :: (section :: STRING?, success :: BOOLEAN?, message :: STRING?)`
Mode	`READ`
Default roles	`admin`

Table 444. db.stats.collect()

Description	Start data collection of a given data section. Valid sections are `'QUERIES'`
Signature	`db.stats.collect(section :: STRING?, config = {} :: MAP?) :: (section :: STRING?, success :: BOOLEAN?, message :: STRING?)`
Mode	`READ`
Default roles	`admin`

Table 445. db.stats.retrieve()

Description	Retrieve statistical data about the current database. Valid sections are `'GRAPH COUNTS'`, `'TOKENS'`, `'QUERIES'`, `'META'`
Signature	`db.stats.retrieve(section :: STRING?, config = {} :: MAP?) :: (section :: STRING?, data :: MAP?)`
Mode	`READ`
Default roles	`admin`

Table 446. db.stats.retrieveAllAnonymized()

Description	Retrieve all available statistical data about the current database, in an anonymized form.
Signature	`db.stats.retrieveAllAnonymized(graphToken :: STRING?, config = {} :: MAP?) :: (section :: STRING?, data :: MAP?)`
Mode	`READ`
Default roles	`admin`

Table 447. db.stats.status()

Description	Retrieve the status of all available collector daemons, for this database.
Signature	`db.stats.status() :: (section :: STRING?, status :: STRING?, data :: MAP?)`
Mode	`READ`
Default roles	`admin`

Table 448. db.stats.stop()

Description	Stop data collection of a given data section. Valid sections are `'QUERIES'`
Signature	`db.stats.stop(section :: STRING?) :: (section :: STRING?, success :: BOOLEAN?, message :: STRING?)`
Mode	`READ`
Default roles	`admin`

Table 449. dbms.cluster.routing.getRoutingTable()

Description	Returns endpoints of this instance.
Signature	`dbms.cluster.routing.getRoutingTable(context :: MAP?, database = null :: STRING?) :: (ttl :: INTEGER?, servers :: LIST? OF MAP?)`
Mode	`DBMS`
Default roles	`reader, editor, publisher, architect, admin`

Table 450. dbms.cluster.overview()

Description	Overview of all currently accessible cluster members, their databases and roles.
Signature	`dbms.cluster.overview() :: (id :: STRING?, addresses :: LIST? OF STRING?, databases :: MAP?, groups :: LIST? OF STRING?)`
Mode	`READ`
Default roles	`reader, editor, publisher, architect, admin`

Table 451. dbms.cluster.protocols()

Description	Overview of installed protocols. Note that this can only be executed on a cluster core member.
Signature	`dbms.cluster.protocols() :: (orientation :: STRING?, remoteAddress :: STRING?, applicationProtocol :: STRING?, applicationProtocolVersion :: INTEGER?, modifierProtocols :: STRING?)`
Mode	`READ`
Default roles	`reader, editor, publisher, architect, admin`

Table 452. dbms.cluster.quarantineDatabase()

Description	Place a database into quarantine or remove from it.
Signature	`dbms.cluster.quarantineDatabase(databaseName :: STRING?, setStatus :: BOOLEAN?, reason = No reason given :: STRING?) :: (databaseName :: STRING?, quarantined :: BOOLEAN?, result :: STRING?)`
Mode	`DBMS`
Default roles	`admin`

Table 453. dbms.cluster.readReplicaToggle()

Description	The toggle can pause or resume the pulling of new transactions for a specific database. If paused, the Read Replica does not pull new transactions from the other cluster members for the specific database. The Read Replica is still available for reads, you can perform a backup, etc.
	
	 
	What is it for?
	
	You can perform a point in time backup, as the backup will contain only the transactions up to the point where the transaction pulling was paused.
	
	1. Connect directly to the Read Replica cluster member. (Neo4j Driver use `bolt://` or use the HTTP API).
	2. Pause transaction pulling for the specified database.
	3. Create a point in time backup, see Back up an online database.
	
	If connected directly to a Read Replica, Data Scientists can execute analysis on a specific database that is paused, the data will not unexpectedly change while performing the analysis.
	
	 
	This procedure can only be executed on a Read Replica cluster member.
	
	Pause transaction pulling for database `neo4j`
	
	`CALL dbms.cluster.readReplicaToggle("neo4j", true)`
	
	Resume transaction pulling for database `neo4j`
	
	`CALL dbms.cluster.readReplicaToggle("neo4j", false)`

Signature	`dbms.cluster.readReplicaToggle(databaseName :: STRING?, pause :: BOOLEAN?) :: (state :: STRING?)`
Mode	`READ`
Default roles	`admin`

Table 454. dbms.cluster.role()

Description	The role of this instance in the cluster for the specified database.
Signature	`dbms.cluster.role(database :: STRING?) :: (role :: STRING?)`
Mode	`READ`
Default roles	`reader, editor, publisher, architect, admin`

Table 455. dbms.cluster.setDefaultDatabase()

Description	Change the default database to the provided value. The database must exist and the old default database must be stopped. For more information see Change the default database. Note that this can only be executed on a cluster core member.
Signature	`dbms.cluster.setDefaultDatabase(databaseName :: STRING?) :: (result :: STRING?)`
Mode	`WRITE`
Default roles	`admin`

Table 456. dbms.components()

Description	List DBMS components and their versions.
Signature	`dbms.components() :: (name :: STRING?, versions :: LIST? OF STRING?, edition :: STRING?)`
Mode	`DBMS`
Default roles	`reader, editor, publisher, architect, admin`

Table 457. dbms.database.state()

Description	The actual status of the database with the provided name on this neo4j instance.
Signature	`dbms.database.state(databaseName :: STRING?) :: (role :: STRING?, address :: STRING?, status :: STRING?, error :: STRING?)`
Mode	`DBMS`
Default roles	`reader, editor, publisher, architect, admin`

Table 458. dbms.functions()

Description	List all functions in the DBMS.
Signature	`dbms.functions() :: (name :: STRING?, signature :: STRING?, category :: STRING?, description :: STRING?, aggregating :: BOOLEAN?, defaultBuiltInRoles :: LIST? OF STRING?)`
Mode	`DBMS`
Default roles	`reader, editor, publisher, architect, admin`

Table 459. dbms.info()

Description	Provides information regarding the DBMS.
Signature	`dbms.info() :: (id :: STRING?, name :: STRING?, creationDate :: STRING?)`
Mode	`DBMS`
Default roles	`reader, editor, publisher, architect, admin`

Table 460. dbms.killConnection()

Description	Kill network connection with the given connection id.
Signature	`dbms.killConnection(id :: STRING?) :: (connectionId :: STRING?, username :: STRING?, message :: STRING?)`
Mode	`DBMS`
Default roles	`reader, editor, publisher, architect, admin`

Table 461. dbms.killConnections()

Description	Kill all network connections with the given connection ids.
Signature	`dbms.killConnections(ids :: LIST? OF STRING?) :: (connectionId :: STRING?, username :: STRING?, message :: STRING?)`
Mode	`DBMS`
Default roles	`reader, editor, publisher, architect, admin`

Table 462. dbms.killQueries()

Description	Kill all transactions executing a query with any of the given query ids.
Signature	`dbms.killQueries(ids :: LIST? OF STRING?) :: (queryId :: STRING?, username :: STRING?, message :: STRING?)`
Mode	`DBMS`
Default roles	`reader, editor, publisher, architect, admin`

Table 463. dbms.killQuery()

Description	Kill all transactions executing the query with the given query id.
Signature	`dbms.killQuery(id :: STRING?) :: (queryId :: STRING?, username :: STRING?, message :: STRING?)`
Mode	`DBMS`
Default roles	`reader, editor, publisher, architect, admin`

Table 464. dbms.killTransaction()

Description	Kill transaction with provided id.
Signature	`dbms.killTransaction(id :: STRING?) :: (transactionId :: STRING?, username :: STRING?, message :: STRING?)`
Mode	`DBMS`
Default roles	`reader, editor, publisher, architect, admin`

Table 465. dbms.killTransactions()

Description	Kill transactions with provided ids.
Signature	`dbms.killTransactions(ids :: LIST? OF STRING?) :: (transactionId :: STRING?, username :: STRING?, message :: STRING?)`
Mode	`DBMS`
Default roles	`reader, editor, publisher, architect, admin`

Table 466. dbms.listActiveLocks()

Description	List the active lock requests granted for the transaction executing the query with the given query id.
Signature	`dbms.listActiveLocks(queryId :: STRING?) :: (mode :: STRING?, resourceType :: STRING?, resourceId :: INTEGER?)`
Mode	`DBMS`
Default roles	`reader, editor, publisher, architect, admin`

Table 467. dbms.listConfig()

Description	List the currently active config of Neo4j.
Signature	`dbms.listConfig(searchString = :: STRING?) :: (name :: STRING?, description :: STRING?, value :: STRING?, dynamic :: BOOLEAN?)`
Mode	`DBMS`
Default roles	`admin`

Table 468. dbms.listConnections()

Description	List all accepted network connections at this instance that are visible to the user.
Signature	`dbms.listConnections() :: (connectionId :: STRING?, connectTime :: STRING?, connector :: STRING?, username :: STRING?, userAgent :: STRING?, serverAddress :: STRING?, clientAddress :: STRING?)`
Mode	`DBMS`
Default roles	`reader, editor, publisher, architect, admin`

Table 469. dbms.listPools()

Description	List all memory pools, including sub pools, currently registered at this instance that are visible to the user.
Signature	`dbms.listPools() :: (pool :: STRING?, databaseName :: STRING?, heapMemoryUsed :: STRING?, heapMemoryUsedBytes :: STRING?, nativeMemoryUsed :: STRING?, nativeMemoryUsedBytes :: STRING?, freeMemory :: STRING?, freeMemoryBytes :: STRING?, totalPoolMemory :: STRING?, totalPoolMemoryBytes :: STRING?)`
Mode	`DBMS`
Default roles	`reader, editor, publisher, architect, admin`

Table 470. dbms.listQueries()

Description	List all queries currently executing at this instance that are visible to the user.
Signature	`dbms.listQueries() :: (queryId :: STRING?, username :: STRING?, metaData :: MAP?, query :: STRING?, parameters :: MAP?, planner :: STRING?, runtime :: STRING?, indexes :: LIST? OF MAP?, startTime :: STRING?, protocol :: STRING?, clientAddress :: STRING?, requestUri :: STRING?, status :: STRING?, resourceInformation :: MAP?, activeLockCount :: INTEGER?, elapsedTimeMillis :: INTEGER?, cpuTimeMillis :: INTEGER?, waitTimeMillis :: INTEGER?, idleTimeMillis :: INTEGER?, allocatedBytes :: INTEGER?, pageHits :: INTEGER?, pageFaults :: INTEGER?, connectionId :: STRING?, database :: STRING?)`
Mode	`DBMS`
Default roles	`reader, editor, publisher, architect, admin`

Table 471. dbms.listTransactions()

Description	List all transactions currently executing at this instance that are visible to the user.
Signature	`dbms.listTransactions() :: (transactionId :: STRING?, username :: STRING?, metaData :: MAP?, startTime :: STRING?, protocol :: STRING?, clientAddress :: STRING?, requestUri :: STRING?, currentQueryId :: STRING?, currentQuery :: STRING?, activeLockCount :: INTEGER?, status :: STRING?, resourceInformation :: MAP?, elapsedTimeMillis :: INTEGER?, cpuTimeMillis :: INTEGER?, waitTimeMillis :: INTEGER?, idleTimeMillis :: INTEGER?, allocatedBytes :: INTEGER?, allocatedDirectBytes :: INTEGER?, pageHits :: INTEGER?, pageFaults :: INTEGER?, connectionId :: STRING?, initializationStackTrace :: STRING?, database :: STRING?, estimatedUsedHeapMemory :: INTEGER?)`
Mode	`DBMS`
Default roles	`reader, editor, publisher, architect, admin`

Table 472. dbms.procedures()

Description	List all procedures in the DBMS.
Signature	`dbms.procedures() :: (name :: STRING?, signature :: STRING?, description :: STRING?, mode :: STRING?, defaultBuiltInRoles :: LIST? OF STRING?, worksOnSystem :: BOOLEAN?)`
Mode	`DBMS`
Default roles	`reader, editor, publisher, architect, admin`

Table 473. dbms.queryJmx()

Description	Query JMX management data by domain and name. Valid queries should use the syntax outlined in the javax.management.ObjectName API documentation. For instance, use `"*:*"` to find all JMX beans.
Signature	`dbms.queryJmx(query :: STRING?) :: (name :: STRING?, description :: STRING?, attributes :: MAP?)`
Mode	`DBMS`
Default roles	`reader, editor, publisher, architect, admin`

Table 474. dbms.routing.getRoutingTable()

Description	Returns endpoints of this instance.
Signature	`dbms.routing.getRoutingTable(context :: MAP?, database = null :: STRING?) :: (ttl :: INTEGER?, servers :: LIST? OF MAP?)`
Mode	`DBMS`
Default roles	`reader, editor, publisher, architect, admin`

Table 475. dbms.scheduler.failedJobs()

Description	List failed job runs. There is a limit for amount of historical data.
Signature	`dbms.scheduler.failedJobs() :: (jobId :: STRING?, group :: STRING?, database :: STRING?, submitter :: STRING?, description :: STRING?, type :: STRING?, submitted :: STRING?, executionStart :: STRING?, failureTime :: STRING?, failureDescription :: STRING?)`
Mode	`DBMS`
Default roles	`admin`

Table 476. dbms.scheduler.groups()

Description	List the job groups that are active in the database internal job scheduler.
Signature	`dbms.scheduler.groups() :: (group :: STRING?, threads :: INTEGER?)`
Mode	`DBMS`
Default roles	`admin`

Table 477. dbms.scheduler.jobs()

Description	List all jobs that are active in the database internal job scheduler.
Signature	`dbms.scheduler.jobs() :: (jobId :: STRING?, group :: STRING?, submitted :: STRING?, database :: STRING?, submitter :: STRING?, description :: STRING?, type :: STRING?, scheduledAt :: STRING?, period :: STRING?, state :: STRING?, currentStateDescription :: STRING?)`
Mode	`DBMS`
Default roles	`admin`

Table 478. dbms.scheduler.profile()

Description	Begin profiling all threads within the given job group, for the specified duration. Note that profiling incurs overhead to a system, and will slow it down.
Signature	`dbms.scheduler.profile(method :: STRING?, group :: STRING?, duration :: STRING?) :: (profile :: STRING?)`
Mode	`DBMS`
Default roles	`admin`

Table 479. dbms.security.activateUser()

Description	Activate a suspended user.
Signature	`dbms.security.activateUser(username :: STRING?, requirePasswordChange = true :: BOOLEAN?) :: VOID`
Mode	`WRITE`
Default roles	`admin`

Table 480. dbms.security.addRoleToUser()

Description	Assign a role to the user.
Signature	`dbms.security.addRoleToUser(roleName :: STRING?, username :: STRING?) :: VOID`
Mode	`WRITE`
Default roles	`admin`

Table 481. dbms.security.changePassword()

Description	Change the current user's password.
Signature	`dbms.security.changePassword(password :: STRING?, requirePasswordChange = false :: BOOLEAN?) :: VOID`
Mode	`WRITE`
Default roles	`reader, editor, publisher, architect, admin`

Table 482. dbms.security.changeUserPassword()

Description	Change the given user's password.
Signature	`dbms.security.changeUserPassword(username :: STRING?, newPassword :: STRING?, requirePasswordChange = true :: BOOLEAN?) :: VOID`
Mode	`WRITE`
Default roles	`admin`

Table 483. dbms.security.clearAuthCache()

Description	Clears authentication and authorization cache.
Signature	`dbms.security.clearAuthCache() :: VOID`
Mode	`DBMS`
Default roles	`admin`

Table 484. dbms.security.createRole()

Description	Create a new role.
Signature	`dbms.security.createRole(roleName :: STRING?) :: VOID`
Mode	`WRITE`
Default roles	`admin`

Table 485. dbms.security.createUser()

Description	Create a new user.
Signature	`dbms.security.createUser(username :: STRING?, password :: STRING?, requirePasswordChange = true :: BOOLEAN?) :: VOID`
Mode	`WRITE`
Default roles	`admin`

Table 486. dbms.security.deleteRole()

Description	Delete the specified role. Any role assignments will be removed.
Signature	`dbms.security.deleteRole(roleName :: STRING?) :: VOID`
Mode	`WRITE`
Default roles	`admin`

Table 487. dbms.security.deleteUser()

Description	Delete the specified user.
Signature	`dbms.security.deleteUser(username :: STRING?) :: VOID`
Mode	`WRITE`
Default roles	`admin`

Table 488. dbms.security.listRoles()

Description	List all available roles.
Signature	`dbms.security.listRoles() :: (role :: STRING?, users :: LIST? OF STRING?)`
Mode	`READ`
Default roles	`admin`

Table 489. dbms.security.listRolesForUser()

Description	List all roles assigned to the specified user.
Signature	`dbms.security.listRolesForUser(username :: STRING?) :: (value :: STRING?)`

Mode	READ
Default roles	admin

Table 490. dbms.security.listUsers()

Description	List all native users.
Signature	dbms.security.listUsers() :: (username :: STRING?, roles :: LIST? OF STRING?, flags :: LIST? OF STRING?)
Mode	READ
Default roles	admin

Table 491. dbms.security.listUsersForRole()

Description	List all users currently assigned the specified role.
Signature	dbms.security.listUsersForRole(roleName :: STRING?) :: (value :: STRING?)
Mode	READ
Default roles	admin

Table 492. dbms.security.removeRoleFromUser()

Description	Unassign a role from the user.
Signature	dbms.security.removeRoleFromUser(roleName :: STRING?, username :: STRING?) :: VOID
Mode	WRITE
Default roles	admin

Table 493. dbms.security.suspendUser()

Description	Suspend the specified user.
Signature	dbms.security.suspendUser(username :: STRING?) :: VOID
Mode	WRITE
Default roles	admin

Table 494. dbms.setConfigValue()

Description	Update a given setting value. Passing an empty value results in removing the configured value and falling back to the default value. Changes do not persist and are lost if the server is restarted. In a clustered environment, dbms.setConfigValue affects only the cluster member it is run against.
Signature	dbms.setConfigValue(setting :: STRING?, value :: STRING?) :: VOID

Mode	DBMS
Default roles	admin

Table 495. dbms.showCurrentUser()

Description	Show the current user.
Signature	dbms.showCurrentUser() :: (username :: STRING?, roles :: LIST? OF STRING?, flags :: LIST? OF STRING?)
Mode	DBMS
Default roles	reader, editor, publisher, architect, admin

Table 496. dbms.upgrade()

Description	Upgrade the system database schema if it is not the current schema.
Signature	dbms.upgrade() :: (status :: STRING?, upgradeResult :: STRING?)
Mode	WRITE
Default roles	admin

Table 497. dbms.upgradeStatus()

Description	Report the current status of the system database sub-graph schema.
Signature	dbms.upgradeStatus() :: (status :: STRING?, description :: STRING?, resolution :: STRING?)
Mode	READ
Default roles	admin

Table 498. tx.getMetaData()

Description	Provides attached transaction metadata.
Signature	tx.getMetaData() :: (metadata :: MAP?)
Mode	DBMS
Default roles	reader, editor, publisher, architect, admin

Table 499. tx.setMetaData()

Description	Attaches a map of data to the transaction. The data will be printed when listing queries, and inserted into the query log.
Signature	tx.setMetaData(data :: MAP?) :: VOID
Mode	DBMS
Default roles	reader, editor, publisher, architect, admin

Appendix B: Tutorials

The following step-by-step tutorials cover common operational tasks or otherwise exemplify working with Neo4j.

- Set up a local Causal Cluster — This tutorial walks through the basics of setting up a Neo4j Causal Cluster.
- Back up and restore a database in Causal Cluster — This tutorial provides a detailed example of how to back up and restore a database in a running Causal Cluster.
- Neo4j Admin import — This tutorial provides detailed examples to illustrate the capabilities of importing data from CSV files with the command `neo4j-admin import`.
- Set up and use Fabric — This tutorial walks through the basics of setting up and using Neo4j Fabric.

16.B.1. Set up a local Causal Cluster

Introduction

In this tutorial, you will learn how to deploy a Causal Cluster locally on a single machine.

Keep in mind that a cluster on a single machine has no fault tolerance and is therefore not suitable for production use.

A typical Causal Cluster consists of three Core instances and three Read Replicas. The Core instances are responsible for keeping the data safe, and the Read Replicas are responsible for scaling the capacity of the cluster. For details on the number of servers required for a Causal Cluster, see Core Servers.

The Core of the Causal Cluster is intended to remain stable over time. The roles within the Core may change as needed, but the Core itself is long-lived and stable.
Read Replicas live at the edge of the cluster and can be brought up and taken down without affecting the Core. They can be added as needed to increase the operational capacity of the cluster as a whole.

For more information about Causal Clustering architecture, configuration, and operation, see Clustering.

Download Neo4j

You download Neo4j and prepare your local environment.

1. Create a local working directory.
2. Download a copy of the Neo4j Enterprise Edition from the Neo4j download site.
3. Unpack Neo4j in the working directory.

Set up the Core servers

You create and configure three Core instances.

Configure and start the first Core instance

You create and configure the first Core instance.

1. Make a copy of the *neo4j-enterprise-4.2.19* directory and name it *core-01*.
   You have to keep the original directory for setting up the other Core instances and Read Replicas. The *core-01* directory will contain the first Core instance.
2. Open the Neo4j configuration file, *conf/neo4j.conf*, and configure the following settings:

   If you cannot find the configuration file, see File locations.

   a. Locate and uncomment the setting `dbms.mode=CORE`.
   b. Locate and uncomment the setting `causal_clustering.minimum_core_cluster_size_at_formation=3`.
   c. Locate and uncomment the setting `causal_clustering.minimum_core_cluster_size_at_runtime=3`.
   d. Locate and uncomment the setting `causal_clustering.initial_discovery_members=localhost:5000,localhost:5001,localhost:5002`.
   e. Locate and uncomment the setting `causal_clustering.discovery_listen_address=:5000`.
   f. Locate and uncomment the setting `causal_clustering.transaction_listen_address=:6000`.
   g. Locate and uncomment the setting `causal_clustering.raft_listen_address=:7000`.
   h. Locate and uncomment the setting `dbms.connector.bolt.listen_address=:7687`.
   i. Locate and uncomment the setting `dbms.connector.http.listen_address=:7474`.
   j. Locate and uncomment the setting `dbms.connector.https.listen_address`, and change the value to `:6474`.
   k. Locate and uncomment the setting `dbms.backup.listen_address=0.0.0.0:6362`.
3. Save the file.
4. Open a command-line tool and navigate to *core-01* directory.
5. Run the following command to start *core-01*:

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

Create and configure the second Core instance

You create and configure the second Core instance.

1. Make a new copy of the *neo4j-enterprise-4.2.19* directory and name it *core-02*.
2. Overwrite *core-02/conf/neo4j.conf* with the just modified *core-01/conf/neo4j.conf*. Then in the new *core-02* directory, open the *conf/neo4j.conf* file and configure the following settings:
   a. Locate the setting `causal_clustering.discovery_listen_address` and change the value to `:5001`.

b. Locate the setting `causal_clustering.transaction_listen_address` and change the value to `:6001`.
c. Locate the setting `causal_clustering.raft_listen_address` and change the value to `:7001`.
d. Locate the setting `dbms.connector.bolt.listen_address` and change the value to `:7688`.
e. Locate the setting `dbms.connector.http.listen_address` and change the value to `:7475`.
f. Locate the setting `dbms.connector.https.listen_address` and change the value to `:6475`.
g. Locate the setting `dbms.backup.listen_address` and change the value to `0.0.0.0:6363`.

3. Save the file.
4. Open a command-line tool and navigate to *core-02* directory.
5. Run the following command to start *core-02*:

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

Create and configure the third Core instance

You create and configure the third Core instance.

1. Make a new copy of the *neo4j-enterprise-4.2.19* directory and name it *core-03*.
2. Overwrite *core-03/conf/neo4j.conf* with the just modified *core-02/conf/neo4j.conf*. Then in the new *core-03* directory, open the *conf/neo4j.conf* file and configure the following settings:
   a. Locate the setting `causal_clustering.discovery_listen_address` and change the value to `:5002`.
   b. Locate the setting `causal_clustering.transaction_listen_address` and change the value to `:6002`.
   c. Locate the setting `causal_clustering.raft_listen_address` and change the value to `:7002`.
   d. Locate the setting `dbms.connector.bolt.listen_address` and change the value to `:7689`.
   e. Locate the setting `dbms.connector.http.listen_address` and change the value to `:7476`.
   f. Locate the setting `dbms.connector.https.listen_address` and change the value to `:6476`.
   g. Locate the setting `dbms.backup.listen_address` and change the value to `0.0.0.0:6364`.
3. Save the file.
4. Open a command-line tool and navigate to *core-03* directory.
5. Run the following command to start *core-03*:

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

Startup Time

To follow along with the startup of a server, check the messages in *<instance-home>/logs/neo4j.log*:

- On a Unix system, run the command `tail -n100 logs/neo4j.log`.
- On Windows Server, run `Get-Content .\logs\neo4j.log -Tail 10 -Wait`.

While an instance is joining the cluster, the server may appear unavailable. In the case where an instance is joining a cluster with lots of data, it may take a number of minutes for the new instance to download the data from the cluster and become available.

Check the status of the cluster

The minimal cluster of three Core servers is operational and is ready to serve requests.

Connect to any of the three Core instances to check the cluster status.

1. Open core-01 at http://localhost:7474.
2. Authenticate with the default `neo4j/neo4j` credentials, and set a new password when prompted.
3. Check the status of the cluster by running the following in Neo4j Browser:

```
:sysinfo
```

Example 127. A cluster of three Core instances.

Name	Address	Role	Status	Default	Error
neo4j	localhost:7689	follower	online	true	-
neo4j	localhost:7688	follower	online	true	-
neo4j	localhost:7687	leader	online	true	-
system	localhost:7689	follower	online	-	-
system	localhost:7688	follower	online	-	-
system	localhost:7687	leader`	online	-	-

4. Run the following query to create nodes and relationships.

```
UNWIND range(0, 100) AS value
MERGE (person1:Person {id: value})
MERGE (person2:Person {id: toInteger(100.0 * rand())})
MERGE (person1)-[:FRIENDS]->(person2)
```

5. Open a new tab and point your web browser to a follower, for example, core-02 at http://localhost:7475.
6. Authenticate with the credentials you have set up for core-01.

7. Run the following query to verify that the data has been replicated:

```
MATCH path = (person:Person)-[:FRIENDS]-(friend)
RETURN path
LIMIT 10
```

Set up the Read Replicas

Because the Read Replicas do not participate in quorum decisions, their configuration is simpler than the configuration of the Core servers.

You configure a Read Replica by setting the address of a Core instance that it can bind to in order to discover the cluster. For details, see Discovery protocol.
After the initial discovery, the Read Replicas can choose a Core instance from which to catch up. For details, see Catchup protocol.

Configure and start the first Read Replica

You create and configure the first Read Replica.

1. Make a copy of the *neo4j-enterprise-4.2.19* directory and name it *replica-01*.
2. In the new *replica-01* directory, open the *conf/neo4j.conf* file and configure the following settings:
   a. Locate and uncomment the setting `dbms.mode`, and change the value to `READ_REPLICA`.
   b. Locate and uncomment the setting `causal_clustering.initial_discovery_members=localhost:5000,localhost:5001,localhost:5002`.
   c. Locate and uncomment the setting `causal_clustering.discovery_listen_address`, and change the value to `:5003`.
   d. Locate and uncomment the setting `causal_clustering.transaction_listen_address`, and change the value to `:6003`.
   e. Locate and uncomment the setting `dbms.connector.bolt.listen_address`, and change the value to `:7690`.
   f. Locate and uncomment the setting `dbms.connector.http.listen_address`, and change the value to `:7477`.
   g. Locate and uncomment the setting `dbms.connector.https.listen_address`, and change the value to `:6477`.
   h. Locate and uncomment the setting `dbms.backup.listen_address`, and change the values to `0.0.0.0:6365`.
3. Save the file.
4. Open a command-line tool and navigate to *replica-01* directory.
5. Run the following command to start *replica-01*:

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

Configure and start the second Read Replica

You create and configure the second Read Replica.

1. Make a new copy of the *neo4j-enterprise-4.2.19* directory and name it *replica-02*.
2. Overwrite *replica-02/conf/neo4j.conf* with the just modified *replica-01/conf/neo4j.conf*. Then in the new *replica-02* directory, open the *conf/neo4j.conf* file and configure the following settings:
   a. Locate the setting `causal_clustering.discovery_listen_address` and change the value to `:5004`.
   b. Locate the setting `causal_clustering.transaction_listen_address` and change the value to `:6004`.
   c. Locate the setting `dbms.connector.bolt.listen_address` and change the value to `:7691`.
   d. Locate the setting `dbms.connector.http.listen_address` and change the value to `:7478`.
   e. Locate the setting `dbms.connector.https.listen_address` and change the value to `:6478`.
   f. Locate the setting `dbms.backup.listen_address` and change the value to `0.0.0.0:6366`.
3. Save the file.
4. Open a command-line tool and navigate to *replica-02* directory.
5. Run the following command to start *replica-02*:

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

Configure and start the third Read Replica

You create and configure the third Read Replica.

1. Make a new copy of the *neo4j-enterprise-4.2.19* directory and name it *replica-03*.
2. Overwrite *replica-03/conf/neo4j.conf* with the just modified *replica-02/conf/neo4j.conf*. Then in the new *replica-03* directory, open the *conf/neo4j.conf* file and configure the following settings:
   a. Locate the setting `causal_clustering.discovery_listen_address` and change the value to `:5005`.
   b. Locate the setting `causal_clustering.transaction_listen_address` and change the value to `:6005`.
   c. Locate the setting `dbms.connector.bolt.listen_address` and change the value to `:7692`.
   d. Locate the setting `dbms.connector.http.listen_address` and change the value to `:7479`.
   e. Locate the setting `dbms.connector.https.listen_address` and change the value to `:6479`.
   f. Locate the setting `dbms.backup.listen_address` and change the value to `0.0.0.0:6367`.
3. Save the file.

4. Open a command-line tool and navigate to *replica-03* directory.
5. Run the following command to start *replica-03*:

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

Check the status of the cluster

Your cluster of three Core servers and three Read Replicas is operational and is ready to serve requests.

In your *core-01* browser, check the cluster status by running the following in Neo4j Browser:

```
:sysinfo
```

Example 128. A cluster of three Core instances and three Read Replicas.

Name	Address	Role	Status	Default	Error
neo4j	localhost:7689	follower	online	true	-
neo4j	localhost:7688	follower	online	true	-
neo4j	localhost:7687	leader	online	true	-
neo4j	localhost:7692	read_replica	online	true	-
neo4j	localhost:7691	read_replica	online	true	-
neo4j	localhost:76890	read_replica	online	true	-
system	localhost:7689	follower	online	-	-
system	localhost:7688	follower	online	-	-
system	localhost:7687	leader	online	-	-
system	localhost:7692	read_replica	online	-	-
system	localhost:7691	read_replica	online	-	-
system	localhost:7690	read_replica	online	-	-

1. Open a new tab and point your web browser to a Read Replica, for example, *replica-01* at http://localhost:7477.
2. Login with `neo4j` and the previously set password and **use the `bolt://` schema**.
3. Run the following query to verify that the data has been replicated:

```
MATCH path = (person:Person)-[:FRIENDS]-(friend)
RETURN path
LIMIT 10
```

16.B.2. Back up and restore a database in Causal Cluster

The following example assumes that you want to restore a database backup, which has users and roles associated with it, in a running Causal Cluster with three core servers. For more information on how to set up a Causal Cluster with three cores, see Set up a local Causal Cluster.

In a Neo4j DBMS, every database is backed up individually. Therefore, it is very important to plan your backup strategy for each of them. For more detailed information on how to design an appropriate backup strategy for your setup, see Backup and restore.

Prepare to back up your database

Before you perform the backup, it is good to take a note of the data and metadata of the database that you want to restore. You can use this information to later verify that the restore is successful and to recreate the database users and roles. In this example, the database is called `movies1` and uses the Movie Graph dataset from the Neo4j Browser → Favorites → Example Graphs.

This tutorial uses the Linux or macOS tarball installation. It assumes that your current work directory is the *<neo4j-home>* directory of the tarball installation.

1. In the Neo4j instance, where the database is running, log in to the Cypher Shell command-line console with your credentials. For more information about the Cypher Shell command-line interface (CLI) and how to use it, see Cypher Shell.

```
bin/cypher-shell -u neo4j -p <password>
```

```
Connected to Neo4j at neo4j://localhost:7687 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
```

2. Change the active database to `movies1`.

```
:use movies1
```

3. Run a query to count the number of nodes in the database.

```
MATCH (n) RETURN count(n) AS countNode;
```

```
+-----------+
| countNode |
+-----------+
| 171       |
+-----------+

1 row available after 22 ms, consumed after another 1 ms
```

4. Run a query to count the number of relationships.

```
MATCH (n)-[r]->() RETURN count(r) AS countRelationships;
```

```
+--------------------+
| countRelationships |
+--------------------+
| 253                |
+--------------------+

1 row available after 29 ms, consumed after another 0 ms
```

5. Change the active database to `system`, and run a query to see if there are any custom roles, associated with this database, and their privileges.

```
SHOW ALL PRIVILEGES;
```

```
+-----------------------------------------------------------------------------------------------+
| access    | action       | resource         | graph     | segment           | role        |
+-----------------------------------------------------------------------------------------------+
| "GRANTED" | "execute"    | "database"       | "*"       | "FUNCTION(*)"     | "PUBLIC"    |
| "GRANTED" | "execute"    | "database"       | "*"       | "PROCEDURE(*)"    | "PUBLIC"    |
| "GRANTED" | "access"     | "database"       | "DEFAULT" | "database"        | "PUBLIC"    |
| "GRANTED" | "match"      | "all_properties" | "*"       | "NODE(*)"         | "admin"     |
| "GRANTED" | "write"      | "graph"          | "*"       | "NODE(*)"         | "admin"     |
| "GRANTED" | "match"      | "all_properties" | "*"       | "RELATIONSHIP(*)" | "admin"     |
| "GRANTED" | "write"      | "graph"          | "*"       | "RELATIONSHIP(*)" | "admin"     |
| "GRANTED" | "access"     | "database"       | "*"       | "database"        | "admin"     |
| "GRANTED" | "admin"      | "database"       | "*"       | "database"        | "admin"     |
| "GRANTED" | "constraint" | "database"       | "*"       | "database"        | "admin"     |
| "GRANTED" | "index"      | "database"       | "*"       | "database"        | "admin"     |
| "GRANTED" | "token"      | "database"       | "*"       | "database"        | "admin"     |
| "GRANTED" | "match"      | "all_properties" | "*"       | "NODE(*)"         | "architect" |
| "GRANTED" | "write"      | "graph"          | "*"       | "NODE(*)"         | "architect" |
| "GRANTED" | "match"      | "all_properties" | "*"       | "RELATIONSHIP(*)" | "architect" |
| "GRANTED" | "write"      | "graph"          | "*"       | "RELATIONSHIP(*)" | "architect" |
| "GRANTED" | "access"     | "database"       | "*"       | "database"        | "architect" |
| "GRANTED" | "constraint" | "database"       | "*"       | "database"        | "architect" |
| "GRANTED" | "index"      | "database"       | "*"       | "database"        | "architect" |
| "GRANTED" | "token"      | "database"       | "*"       | "database"        | "architect" |
| "GRANTED" | "match"      | "all_properties" | "*"       | "NODE(*)"         | "editor"    |
| "GRANTED" | "write"      | "graph"          | "*"       | "NODE(*)"         | "editor"    |
| "GRANTED" | "match"      | "all_properties" | "*"       | "RELATIONSHIP(*)" | "editor"    |
| "GRANTED" | "write"      | "graph"          | "*"       | "RELATIONSHIP(*)" | "editor"    |
| "GRANTED" | "access"     | "database"       | "*"       | "database"        | "editor"    |
| "GRANTED" | "match"      | "all_properties" | "*"       | "NODE(*)"         | "myrole"    |
| "GRANTED" | "match"      | "all_properties" | "*"       | "RELATIONSHIP(*)" | "myrole"    |
| "GRANTED" | "access"     | "database"       | "*"       | "database"        | "myrole"    |
| "GRANTED" | "write"      | "graph"          | "movies1" | "NODE(*)"         | "myrole"    |
| "GRANTED" | "write"      | "graph"          | "movies1" | "RELATIONSHIP(*)" | "myrole"    |
| "GRANTED" | "match"      | "all_properties" | "*"       | "NODE(*)"         | "publisher" |
| "GRANTED" | "write"      | "graph"          | "*"       | "NODE(*)"         | "publisher" |
| "GRANTED" | "match"      | "all_properties" | "*"       | "RELATIONSHIP(*)" | "publisher" |
| "GRANTED" | "write"      | "graph"          | "*"       | "RELATIONSHIP(*)" | "publisher" |
| "GRANTED" | "access"     | "database"       | "*"       | "database"        | "publisher" |
| "GRANTED" | "token"      | "database"       | "*"       | "database"        | "publisher" |
| "GRANTED" | "match"      | "all_properties" | "*"       | "NODE(*)"         | "reader"    |
| "GRANTED" | "match"      | "all_properties" | "*"       | "RELATIONSHIP(*)" | "reader"    |
| "GRANTED" | "access"     | "database"       | "*"       | "database"        | "reader"    |
+-----------------------------------------------------------------------------------------------+

39 rows available after 868 ms, consumed after another 80 ms
```

The result shows that there is one custom role `myrole`.

6. Run a query to see all users associated with this role.

```
SHOW USERS;
```

```
+-------------------------------------------------------------------+
| user    | roles                | passwordChangeRequired | suspended |
+-------------------------------------------------------------------+
| "neo4j" | ["admin", "PUBLIC"]  | FALSE                  | FALSE     |
| "user1" | ["myrole", "PUBLIC"] | TRUE                   | FALSE     |
+-------------------------------------------------------------------+

2 rows available after 36 ms, consumed after another 2 ms
```

7. Exit the Cypher Shell command-line console.

```
:exit
```

Back up your database

Now you are ready to back up the database.

Run the following command to back up the database in your targeted folder. If the folder where you want to place your backup does not exist, you have to create it. In this example, it is called */tmp/4.2.19*.

To perform the backup, run the following command:

```
bin/neo4j-admin backup --backup-dir=/tmp/4.2.19 --database=movies1 --include-metadata=all
```

The option `--include-metadata=all` creates a cypher script, which you can later use to restore the database's users, roles, and privileges.

For details on performing a backup and the different command options, see Back up an online database.

Delete the database that you want to replace

Before you restore the database backup, you have to delete the database that you want to replace with that backup. If you want to restore the backup as an *additional* database in your DBMS, then you can proceed to Restore the database backup on all cluster members directly.

On one of the cluster members, run the Cypher command `DROP DATABASE` to delete the database that you want to replace. The command is automatically routed to the leader and from there to the other cluster members.

Dropping a database also deletes the users and roles associated with it.

1. In the Cypher Shell command-line console on one of the cluster members, change the active database to `system`, and run the command `DROP DATABASE` to delete the database that you want to replace. In this example, the database is called `movies`.

```
DROP DATABASE movies;
```

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

If you are unable to delete the database (e.g., because Neo4j is not running), you must run `neo4j-admin unbind` first instead. If you fail to do this, the store files you have (post restore) will be out of sync with the cluster state you have for that database, leading to logical corruption.

2. You can run `SHOW DATABASES` to verify that the database `movies` does not exist.

```
SHOW DATABASES;
```

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

6 rows available after 7 ms, consumed after another 3 ms
```

3. Exit the Cypher Shell command-line console.

```
:exit
```

Restore the database backup on all cluster members

On each cluster member, run the following command to restore the database backup. For details on performing a restore and the different command options, see Restore a database backup.

```
bin/neo4j-admin restore --from=/tmp/4.2.19/movies1 --database=movies1
```

```
You need to execute $HOME/path/to/core-member/data/scripts/movies1/restore_metadata.cypher. To execute the
file use cypher-shell command with parameter `movies1`
restorePath=/tmp/{neo4j-version-exact}/movies1, restoreStatus=successful, reason=
```

Then, on each cluster member, run the following command to verify that the database `movies1` exists:

```
ls -al data/databases
```

```
total 0
drwxr-xr-x@  7 username  staff   224 17 Nov 15:50 .
drwxr-xr-x@  8 username  staff   256 17 Nov 15:50 ..
drwxr-xr-x  40 username  staff  1280 17 Nov 15:50 movies1
drwxr-xr-x  37 username  staff  1184 16 Nov 15:00 neo4j
-rw-r--r--   1 username  staff     0 16 Nov 15:00 store_lock
drwxr-xr-x  38 username  staff  1216 16 Nov 15:00 system
```

However, restoring a database does not automatically create it. Therefore, it will not be visible if you do `SHOW DATABASES` in Cypher Shell or Neo4j Browser.

Create the database backup on the cluster leader

You create the database backup **only on one of your cluster members** using the command `CREATE DATABASE`. The command is automatically routed to the leader and from there to the other cluster members.

1. In the Cypher Shell command-line console on one of the cluster members, use the `system` database and create the database `movies1`.

```
CREATE DATABASE movies1;
```

```
0 rows available after 132 ms, consumed after another 0 ms
```

2. Verify that the `movies1` database is online on all members.

```
SHOW DATABASES;
```

```
+-----------------------------------------------------------------------------------------------+
| name      | address          | role       | requestedStatus | currentStatus | error | default |
+-----------------------------------------------------------------------------------------------+
| "movies1" | "localhost:7688" | "follower" | "online"        | "online"      | ""    | FALSE   |
| "movies1" | "localhost:7687" | "leader"   | "online"        | "online"      | ""    | FALSE   |
| "movies1" | "localhost:7689" | "follower" | "online"        | "online"      | ""    | FALSE   |
| "neo4j"   | "localhost:7688" | "leader"   | "online"        | "online"      | ""    | TRUE    |
| "neo4j"   | "localhost:7687" | "follower" | "online"        | "online"      | ""    | TRUE    |
| "neo4j"   | "localhost:7689" | "follower" | "online"        | "online"      | ""    | TRUE    |
| "system"  | "localhost:7688" | "follower" | "online"        | "online"      | ""    | FALSE   |
| "system"  | "localhost:7687" | "leader"   | "online"        | "online"      | ""    | FALSE   |
| "system"  | "localhost:7689" | "follower" | "online"        | "online"      | ""    | FALSE   |
+-----------------------------------------------------------------------------------------------+

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

3. Exit the Cypher Shell command-line console.

```
:exit
```

Recreate the database users and roles

On one of the cluster members, run the restore cypher script *restore_metadata.cypher* to create the database and recreate all users and roles of the database backup. The command is automatically routed to the leader and from there to the other cluster members.

Using `cat` (UNIX)

```
cat data/scripts/movies1/restore_metadata.cypher | bin/cypher-shell -u neo4j -p password -a localhost:7688
-d system --param "database => 'movies1'"
```

Using `type` (Windows)

```
type data\scripts\movies1\restore_metadata.cypher | bin\cypher-shell.bat -u neo4j -p password -a
localhost:7688 -d system --param "database => 'movies1'"
```

Follow the steps from 1 to 6 of section Prepare to back up your database to verify that all data and metadata of the database backup have been successfully restored on all cluster members.

16.B.3. Neo4j Admin import

The `neo4j-admin import` is a command for loading large amounts of data from CSV files into an **unused** database. Importing data from CSV files with `neo4j-admin import` can only be done once into an **unused** database, it is used for initial graph population only. The `neo4j-admin import` command can be used on the local Neo4j instance even if the instance is running or not.

The `neo4j-admin import` command does not create a database, the command only imports data and make it available for the database. It is possible to create the given database either before or after the `neo4j-admin import` command have been executed. If the database already exists the given database needs to be in a state where no data have been introduced before.

Relationships are created by connecting node IDs, each node should have a unique ID to be able to be referenced when creating relationships between nodes. In the following examples, the node IDs are stored as properties on the nodes. If you do not want the IDs to persist as properties after the import completes, then do not specify a property name in the `:ID` field.

The examples show how to import data in a standalone Neo4j DBMS. They use:

- The Neo4j tarball (Unix console application).
- `$NEO4J_HOME` as the current working directory.
- The default database `neo4j`.
- The *import* directory of the Neo4j installation to store all the CSV files. However, the CSV files can be located in any directory of your file system.
- UNIX styled paths.
- The `neo4j-admin import` command.

To create a cluster based on imported data, see Seed a cluster using the import tool.

Handy tips:

- The details of CSV file header format can be found at CSV header format.
- To show available databases, use the Cypher query `SHOW DATABASES` against the `system` database.
- To remove a database, use the Cypher query `DROP DATABASE database_name` against the `system` database.
- To create a database, use the Cypher query `CREATE DATABASE database_name` against the `system` database.

Import a small data set

In this example, you will import a small data set containing nodes and relationships. The data set is split into three CSV files, where each file has a header row describing the data.

The data

The data set contains information about movies, actors, and roles. Data for movies and actors are stored as nodes and the roles are stored as relationships.

The files you want to import data from are:

- `movies.csv`
- `actors.csv`
- `roles.csv`

Each movie in `movies.csv` has an `ID`, a `title` and a `year`, stored as **properties** in the node. All the nodes in `movies.csv` also have the **label** `Movie`. A node can have several labels, as you can see in `movies.csv` there are nodes that also have the label `Sequel`. The node labels are optional, they are very useful for grouping nodes into sets where all nodes that have a certain label belongs to the same set.

movies.csv

```
movieId:ID,title,year:int,:LABEL
tt0133093,"The Matrix",1999,Movie
tt0234215,"The Matrix Reloaded",2003,Movie;Sequel
tt0242653,"The Matrix Revolutions",2003,Movie;Sequel
```

The actors data in `actors.csv` consist of an `ID` and a `name`, stored as **properties** in the node. The ID in this case a shorthand of the actors name. All the nodes in `actors.csv` have the label `Actor`.

actors.csv

```
personId:ID,name,:LABEL
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
```

The roles data in `roles.csv` have only one **property**, `role`. Roles are represented by relationship data that connects actor nodes with movie nodes.

There are three mandatory fields for relationship data:

1. `:START_ID` — ID refering to a node.
2. `:END_ID` — ID refering to a node.
3. `:TYPE` — The relationship type.

In order to create a relationship between two nodes, the IDs defined in `actors.csv` and `movies.csv` are used for the `:START_ID` and `:END_ID` fields. You also need to provide a relationship type (in this case `ACTED_IN`) for the `:TYPE` field.

roles.csv

```
:START_ID,role,:END_ID,:TYPE
keanu,"Neo",tt0133093,ACTED_IN
keanu,"Neo",tt0234215,ACTED_IN
keanu,"Neo",tt0242653,ACTED_IN
laurence,"Morpheus",tt0133093,ACTED_IN
laurence,"Morpheus",tt0234215,ACTED_IN
laurence,"Morpheus",tt0242653,ACTED_IN
carrieanne,"Trinity",tt0133093,ACTED_IN
carrieanne,"Trinity",tt0234215,ACTED_IN
carrieanne,"Trinity",tt0242653,ACTED_IN
```

Importing the data

- Paths to node data is defined with the `--nodes` option.
- Paths to relationship data is defined with the `--relationships` option.

The call to `neo4j-admin import` would look like this:

shell

```
bin/neo4j-admin import --database=neo4j --nodes=import/movies.csv --nodes=import/actors.csv
--relationships=import/roles.csv
```

Query the data

To query the data. Start Neo4j.

The default username and password is `neo4j` and `neo4j`.

shell

```
bin/neo4j start
```

To query the imported data in the graph, try a simple Cypher query.

shell

```
bin/cypher-shell --database=neo4j "MATCH (n) RETURN count(n) as nodes"
```

Stop Neo4j.

shell

```
bin/neo4j stop
```

CSV file delimiters

You can customize the configuration options that the import tool uses (see Options) if your data does not fit the default format.

The details of CSV file header format can be found at CSV header format.

The data

The following CSV files have:

- `--delimiter=";"`
- `--array-delimiter="|"`
- `--quote="'"`

movies2.csv

```
movieId:ID;title;year:int;:LABEL
tt0133093;'The Matrix';1999;Movie
tt0234215;'The Matrix Reloaded';2003;Movie|Sequel
tt0242653;'The Matrix Revolutions';2003;Movie|Sequel
```

actors2.csv

```
personId:ID;name;:LABEL
keanu;'Keanu Reeves';Actor
laurence;'Laurence Fishburne';Actor
carrieanne;'Carrie-Anne Moss';Actor
```

roles2.csv

```
:START_ID;role;:END_ID;:TYPE
keanu;'Neo';tt0133093;ACTED_IN
keanu;'Neo';tt0234215;ACTED_IN
keanu;'Neo';tt0242653;ACTED_IN
laurence;'Morpheus';tt0133093;ACTED_IN
laurence;'Morpheus';tt0234215;ACTED_IN
laurence;'Morpheus';tt0242653;ACTED_IN
carrieanne;'Trinity';tt0133093;ACTED_IN
carrieanne;'Trinity';tt0234215;ACTED_IN
carrieanne;'Trinity';tt0242653;ACTED_IN
```

Importing the data

The call to `neo4j-admin import` would look like this:

shell

```
bin/neo4j-admin import --database=neo4j --delimiter=";" --array-delimiter="|" --quote="'"
--nodes=import/movies2.csv --nodes=import/actors2.csv --relationships=import/roles2.csv
```

Using separate header files

When dealing with very large CSV files, it is more convenient to have the header in a separate file. This makes it easier to edit the header as you avoid having to open a huge data file just to change it. The header file must be specified before the rest of the files in each file group.

The import tool can also process single file compressed archives, for example:

- `--nodes=import/nodes.csv.gz`
- `--relationships=import/relationships.zip`

The data

You will use the same data set as in the previous example but with the headers in separate files.

movies3-header.csv

```
movieId:ID,title,year:int,:LABEL
```

movies3.csv

```
tt0133093,"The Matrix",1999,Movie
tt0234215,"The Matrix Reloaded",2003,Movie;Sequel
tt0242653,"The Matrix Revolutions",2003,Movie;Sequel
```

actors3-header.csv

```
personId:ID,name,:LABEL
```

actors3.csv

```
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
```

roles3-header.csv

```
:START_ID,role,:END_ID,:TYPE
```

roles3.csv

```
keanu,"Neo",tt0133093,ACTED_IN
keanu,"Neo",tt0234215,ACTED_IN
keanu,"Neo",tt0242653,ACTED_IN
laurence,"Morpheus",tt0133093,ACTED_IN
laurence,"Morpheus",tt0234215,ACTED_IN
laurence,"Morpheus",tt0242653,ACTED_IN
carrieanne,"Trinity",tt0133093,ACTED_IN
carrieanne,"Trinity",tt0234215,ACTED_IN
carrieanne,"Trinity",tt0242653,ACTED_IN
```

Importing the data

The call to `neo4j-admin import` would look as follows:

The header line for a file group, whether it is the first line of a file in the group or a dedicated header file, must be the *first* line in the file group.

shell

```
bin/neo4j-admin import --database=neo4j --nodes=import/movies3-header.csv,import/movies3.csv
--nodes=import/actors3-header.csv,import/actors3.csv --relationships=import/roles3
-header.csv,import/roles3.csv
```

Multiple input files

In addition to using a separate header file you can also provide multiple nodes or relationships files. Files within such an input group can be specified with multiple match strings, delimited by `,`, where each match string can be either the exact file name or a regular expression matching one or more files. Multiple matching files will be sorted according to their characters and their natural number sort order for file names containing numbers.

The data

movies4-header.csv

```
movieId:ID,title,year:int,:LABEL
```

movies4-part1.csv

```
tt0133093,"The Matrix",1999,Movie
tt0234215,"The Matrix Reloaded",2003,Movie;Sequel
```

movies4-part2.csv

```
tt0242653,"The Matrix Revolutions",2003,Movie;Sequel
```

actors4-header.csv

```
personId:ID,name,:LABEL
```

actors4-part1.csv

```
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
```

actors4-part2.csv

```
carrieanne,"Carrie-Anne Moss",Actor
```

roles4-header.csv

```
:START_ID,role,:END_ID,:TYPE
```

roles4-part1.csv

```
keanu,"Neo",tt0133093,ACTED_IN
keanu,"Neo",tt0234215,ACTED_IN
keanu,"Neo",tt0242653,ACTED_IN
laurence,"Morpheus",tt0133093,ACTED_IN
laurence,"Morpheus",tt0234215,ACTED_IN
```

roles4-part2.csv

```
laurence,"Morpheus",tt0242653,ACTED_IN
carrieanne,"Trinity",tt0133093,ACTED_IN
carrieanne,"Trinity",tt0234215,ACTED_IN
carrieanne,"Trinity",tt0242653,ACTED_IN
```

Importing the data

The call to `neo4j-admin import` would look like this:

shell

```
bin/neo4j-admin import --database=neo4j --nodes=import/movies4-header.csv,import/movies4
-part1.csv,import/movies4-part2.csv --nodes=import/actors4-header.csv,import/actors4
-part1.csv,import/actors4-part2.csv --relationships=import/roles4-header.csv,import/roles4
-part1.csv,import/roles4-part2.csv
```

Regular expressions

File names can be specified using regular expressions in order to simplify using the command line when there are many data source files. Each file name that matches the regular expression will be included.

If using separate header files, for the import to work correctly, the header file must be the first in the file group. When using regular expressions to specify the input files, the list of files will be sorted according to the names of the files that match the expression. The matching is aware of numbers inside the file names and will sort them accordingly, without the need for padding with zeros.

Example 129. Match order

For example, let's assume that you have the following files:

- `movies4-header.csv`
- `movies4-data1.csv`
- `movies4-data2.csv`
- `movies4-data12.csv`

If you use the regular expression `movies4.*`, the sorting will place the header file last and the import will fail. A better alternative would be to name the header file explicitly and use a regular expression that only matches the names of the data files. For example: `--nodes "import/movies4-header.csv,movies-data.*"` will accomplish this.

Importing the data using regular expressions, the call to `neo4j-admin import` can be simplified to:

shell

```
bin/neo4j-admin import --database=neo4j --nodes="import/movies4-header.csv,import/movies4-part.*"
--nodes="import/actors4-header.csv,import/actors4-part.*" --relationships="import/roles4
-header.csv,import/roles4-part.*"
```

The use of regular expressions should not be confused with file globbing.

The expression `.*` means: "zero or more occurrences of any character except line break". Therefore, the regular expression `movies4.*` will list all files starting with `movies4`. Conversely, with file globbing, `ls movies4.*` will list all files starting with `movies4.`.

Another important difference to pay attention to is the sorting order. The result of a regular expression matching will place the file `movies4-part2.csv` before the file `movies4-part12.csv`. If doing `ls movies4-part*` in a directory containing the above listed files, the file `movies4-part12.csv` will be listed before the file `movies4-part2.csv`.

Using the same label for every node

If you want to use the same node label(s) for every node in your nodes file you can do this by specifying the appropriate value as an option to `neo4j-admin import`. There is then no need to specify the `:LABEL` column in the header file and each row (node) will apply the specified labels from the command line option.

Example 130. Specify node labels option

```
--nodes=LabelOne:LabelTwo=import/example-header.csv,import/example-data1.csv
```

It is possible to apply both the label provided in the file and the one provided on the command line to the node.

The data

In this example you want to have the label `Movie` on every node specified in `movies5a.csv`, and you put the labels `Movie` and `Sequel` on the nodes specified in `sequels5a.csv`.

movies5a.csv

```
movieId:ID,title,year:int
tt0133093,"The Matrix",1999
```

sequels5a.csv

```
movieId:ID,title,year:int
tt0234215,"The Matrix Reloaded",2003
tt0242653,"The Matrix Revolutions",2003
```

actors5a.csv

```
personId:ID,name
keanu,"Keanu Reeves"
laurence,"Laurence Fishburne"
carrieanne,"Carrie-Anne Moss"
```

roles5a.csv

```
:START_ID,role,:END_ID,:TYPE
keanu,"Neo",tt0133093,ACTED_IN
keanu,"Neo",tt0234215,ACTED_IN
keanu,"Neo",tt0242653,ACTED_IN
laurence,"Morpheus",tt0133093,ACTED_IN
laurence,"Morpheus",tt0234215,ACTED_IN
laurence,"Morpheus",tt0242653,ACTED_IN
carrieanne,"Trinity",tt0133093,ACTED_IN
carrieanne,"Trinity",tt0234215,ACTED_IN
carrieanne,"Trinity",tt0242653,ACTED_IN
```

Importing the data

The call to `neo4j-admin import` would look like this:

shell

```
bin/neo4j-admin import --database=neo4j --nodes=Movie=import/movies5a.csv
--nodes=Movie:Sequel=import/sequels5a.csv --nodes=Actor=import/actors5a.csv
--relationships=import/roles5a.csv
```

Using the same relationship type for every relationship

If you want to use the same relationship type for every relationship in your relationships file this can be done by specifying the appropriate value as an option to `neo4j-admin import`.

Example 131. Specify relationship type option

```
--relationships=TYPE=import/example-header.csv,import/example-data1.csv
```

If you provide a relationship type both on the command line and in the relationships file, the one in the file will be applied.

The data

In this example you want the relationship type `ACTED_IN` to be applied on every relationship specified in `roles5b.csv`.

movies5b.csv

```
movieId:ID,title,year:int,:LABEL
tt0133093,"The Matrix",1999,Movie
tt0234215,"The Matrix Reloaded",2003,Movie;Sequel
tt0242653,"The Matrix Revolutions",2003,Movie;Sequel
```

actors5b.csv

```
personId:ID,name,:LABEL
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
```

roles5b.csv

```
:START_ID,role,:END_ID
keanu,"Neo",tt0133093
keanu,"Neo",tt0234215
keanu,"Neo",tt0242653
laurence,"Morpheus",tt0133093
laurence,"Morpheus",tt0234215
laurence,"Morpheus",tt0242653
carrieanne,"Trinity",tt0133093
carrieanne,"Trinity",tt0234215
carrieanne,"Trinity",tt0242653
```

Importing the data

The call to `neo4j-admin import` would look like this:

shell

```
bin/neo4j-admin import --database=neo4j --nodes=import/movies5b.csv --nodes=import/actors5b.csv
--relationships=ACTED_IN=import/roles5b.csv
```

Properties

Nodes and relationships can have properties. The property type are specified in the CSV header row, see CSV header format.

The data

The following example creates a small graph containing one actor and one movie connected by one relationship.

There is a `roles` property on the relationship which contains an array of the characters played by the actor in a movie:

movies6.csv

```
movieId:ID,title,year:int,:LABEL
tt0099892,"Joe Versus the Volcano",1990,Movie
```

actors6.csv

```
personId:ID,name,:LABEL
meg,"Meg Ryan",Actor
```

roles6.csv

```
:START_ID,roles:string[],:END_ID,:TYPE
meg,"DeDe;Angelica Graynamore;Patricia Graynamore",tt0099892,ACTED_IN
```

Importing the data

The call to `neo4j-admin import` would look like this:

shell

```
bin/neo4j-admin import --database=neo4j --nodes=import/movies6.csv --nodes=import/actors6.csv
--relationships=import/roles6.csv
```

ID space

The import tool makes the assumption that identifiers are unique across node files. This may not be the case for data sets which use sequential, auto incremented or otherwise colliding identifiers. Those data sets can define ID spaces where identifiers are unique within their respective ID space.

In cases where the node ID is only unique within files, using ID spaces is a way to ensure uniqueness across all nodes files. See Using ID spaces.

Each node processed by `neo4j-admin import` must provide an ID if it is to be connected in any relationships. The node ID is used to find the start node and end node when creating a relationship.

Example 132. ID space

To define a ID space `Movie-ID` for `movieId:ID` the syntax will be `movieId:ID(Movie-ID)`.

The data

For example, if movies and people both use sequential identifiers, then you would define `Movie` and `Actor` ID spaces.

movies7.csv

```
movieId:ID(Movie-ID),title,year:int,:LABEL
1,"The Matrix",1999,Movie
2,"The Matrix Reloaded",2003,Movie;Sequel
3,"The Matrix Revolutions",2003,Movie;Sequel
```

actors7.csv

```
personId:ID(Actor-ID),name,:LABEL
1,"Keanu Reeves",Actor
2,"Laurence Fishburne",Actor
3,"Carrie-Anne Moss",Actor
```

You also need to reference the appropriate ID space in your relationships file so it knows which nodes to

connect together.

roles7.csv

```
:START_ID(Actor-ID),role,:END_ID(Movie-ID)
1,"Neo",1
1,"Neo",2
1,"Neo",3
2,"Morpheus",1
2,"Morpheus",2
2,"Morpheus",3
3,"Trinity",1
3,"Trinity",2
3,"Trinity",3
```

Importing the data

The call to `neo4j-admin import` would look like this:

shell

```
bin/neo4j-admin import --database=neo4j --nodes=import/movies7.csv --nodes=import/actors7.csv
--relationships=ACTED_IN=import/roles7.csv
```

Skip relationships referring to missing nodes

The import tool has no tolerance for bad entities (relationships or nodes) and will fail the import on the first bad entity. You can specify explicitly that you want it to ignore rows that contain bad entities.

There are two different types of bad input:

1. Bad relationships.
2. Bad nodes.

Relationships that refer to missing node IDs, either for `:START_ID` or `:END_ID` are considered bad relationships. Whether or not such relationships are skipped is controlled with `--skip-bad-relationships` flag, which can have the values `true` or `false` or no value, which means `true`. The default is `false`, which means that any bad relationship is considered an error and will fail the import. For more information, see the `--skip-bad-relationships` option.

The data

In the following example there is a missing `emil` node referenced in the roles file.

movies8a.csv

```
movieId:ID,title,year:int,:LABEL
tt0133093,"The Matrix",1999,Movie
tt0234215,"The Matrix Reloaded",2003,Movie;Sequel
tt0242653,"The Matrix Revolutions",2003,Movie;Sequel
```

actors8a.csv

```
personId:ID,name,:LABEL
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
```

roles8a.csv

```
:START_ID,role,:END_ID,:TYPE
keanu,"Neo",tt0133093,ACTED_IN
keanu,"Neo",tt0234215,ACTED_IN
keanu,"Neo",tt0242653,ACTED_IN
laurence,"Morpheus",tt0133093,ACTED_IN
laurence,"Morpheus",tt0234215,ACTED_IN
laurence,"Morpheus",tt0242653,ACTED_IN
carrieanne,"Trinity",tt0133093,ACTED_IN
carrieanne,"Trinity",tt0234215,ACTED_IN
carrieanne,"Trinity",tt0242653,ACTED_IN
emil,"Emil",tt0133093,ACTED_IN
```

Importing the data

The call to `neo4j-admin import` would look like this:

shell

```
bin/neo4j-admin import --database=neo4j --nodes=import/movies8a.csv --nodes=import/actors8a.csv
--relationships=import/roles8a.csv
```

Since there was a bad relationship in the input data, the import process will fail completely.

Let's see what happens if you append the `--skip-bad-relationships` flag:

shell

```
bin/neo4j-admin import --database=neo4j --skip-bad-relationships --nodes=import/movies8a.csv
--nodes=import/actors8a.csv --relationships=import/roles8a.csv
```

The data files are successfully imported and the bad relationship is ignored. An entry is written to the `import.report` file.

ignore bad relationships

```
InputRelationship:
   source: roles8a.csv:11
   properties: [role, Emil]
   startNode: emil (global id space)
   endNode: tt0133093 (global id space)
   type: ACTED_IN
 referring to missing node emil
```

Skip nodes with same ID

Nodes that specify `:ID` which has already been specified within the ID space are considered bad nodes. Whether or not such nodes are skipped is controlled with `--skip-duplicate-nodes` flag which can have

the values `true` or `false` or no value, which means `true`. The default is `false`, which means that any duplicate node is considered an error and will fail the import. For more information, see the `--skip-duplicate-nodes` option.

The data

In the following example there is a node ID, `laurence`, that is specified twice within the same ID space.

actors8b.csv

```
personId:ID,name,:LABEL
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
laurence,"Laurence Harvey",Actor
```

Importing the data

The call to `neo4j-admin import` would look like this:

shell

```
bin/neo4j-admin import --database=neo4j --nodes=import/actors8b.csv
```

Since there was a bad node in the input data, the import process will fail completely.

Let's see what happens if you append the `--skip-duplicate-nodes` flag:

shell

```
bin/neo4j-admin import --database=neo4j --skip-duplicate-nodes --nodes=import/actors8b.csv
```

The data files are successfully imported and the bad node is ignored. An entry is written to the `import.report` file.

ignore bad nodes

```
ID 'laurence' is defined more than once in global ID space, at least at actors8b.csv:3 and actors8b.csv:5
```

16.B.4. Set up and use Fabric

Neo4j Fabric is a tool for storing and retrieving data in multiple databases, located in one or many Neo4j DBMS(s), with a single Cypher query.

In this tutorial, you will learn how to:

- Model your data for Fabric
- Configure Fabric with three databases
- Import data in your databases

- Retrieve data with a single Cypher query

For more information on how to manage multiple active databases in Neo4j, see Manage databases.
For more details on Fabric, see Fabric.

Model your data for Fabric

Northwind data

The example data in this tutorial is based on the Northwind dataset, created by Microsoft.

It contains the sales data of a fictitious small company called “Northwind Traders”. The data includes customers, products, customer orders, warehouse stock, shipping, suppliers, employees, and sales territories.

The model

The Northwind graph model consists of the following data:

- Node labels
  - `:Product`
  - `:Category`
  - `:Supplier`
  - `:Order`
  - `:Customer`
- Relationship types
  - `:SUPPLIES`
  - `:PART_OF`
  - `:ORDERS`
  - `:PURCHASED`

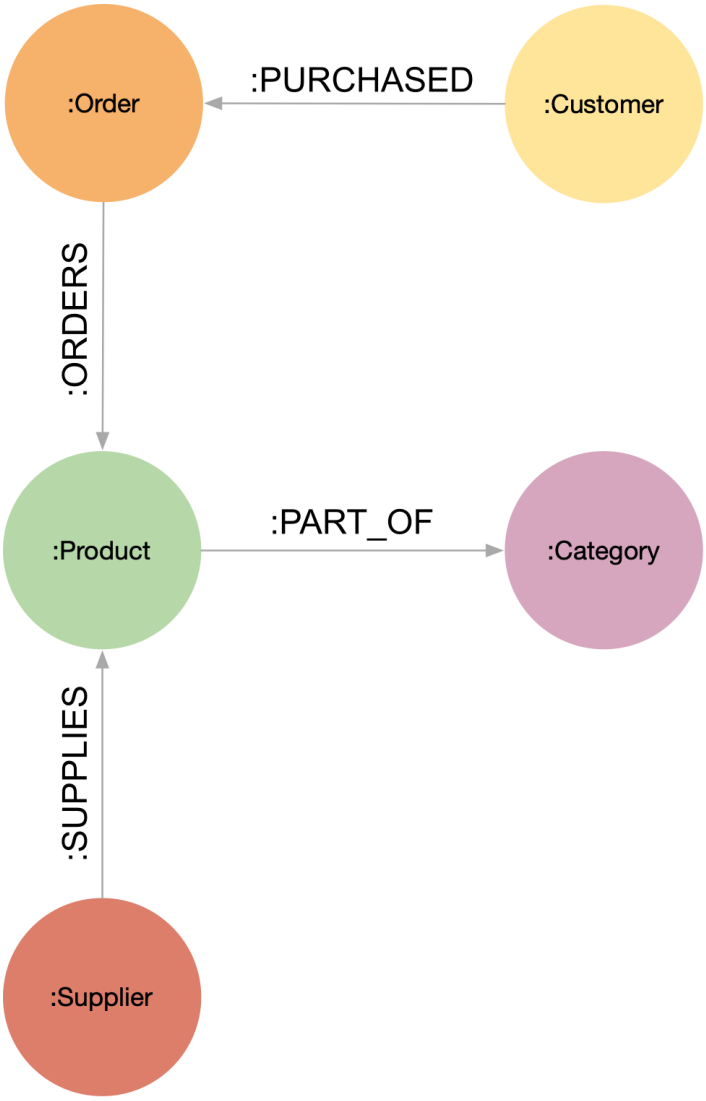


Figure 14. The Northwind data model

Remodeling the Northwind dataset

In this scenario, you imagine that data privacy constraints require customers' data to be stored in their original region. For simplicity, there are two regions: the Americas (AME) and Europe (EU). The first step is to remodel the Northwind dataset, so that customer data can be separated from the Product catalog, which has no privacy constraints. You create two graphs: one for the Product catalog, which includes `:Product`, `:Category`, `:Supplier`, `:PART_OF`, `:SUPPLIES`, and one partitioned graph in two databases for the Customer orders in EU and AME, with `:Product`, `:Order`, `:Customer`, `:PURCHASED`, and `:ORDERS`.

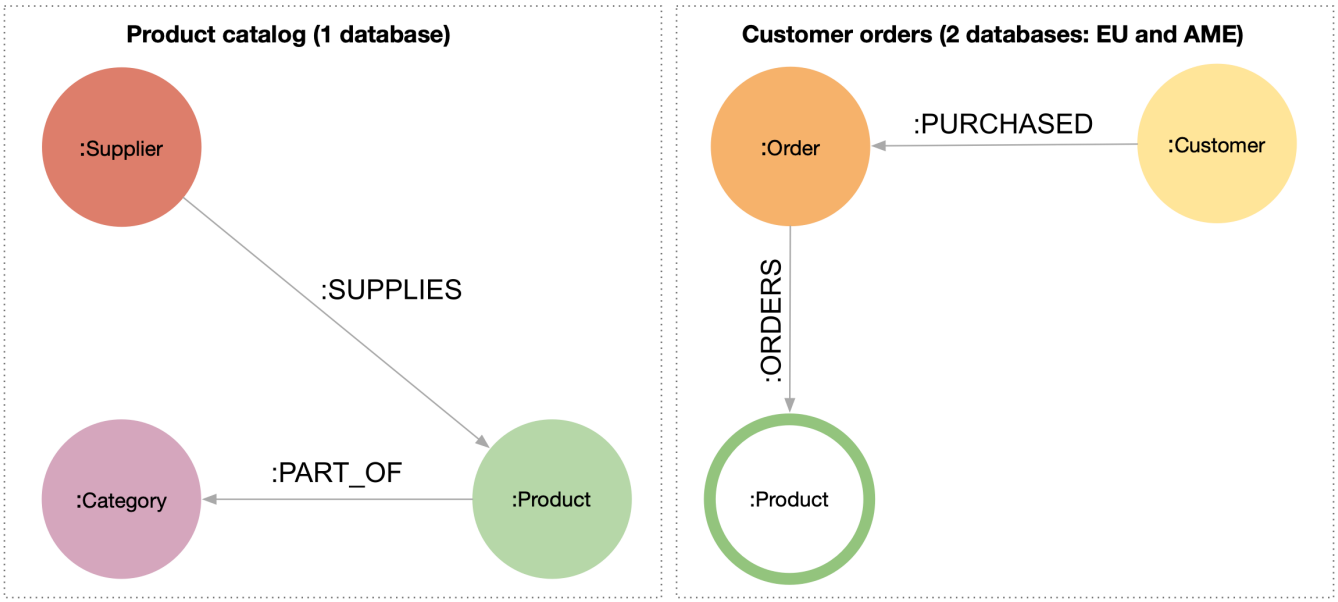


Figure 15. The new data model

Data Federation

This way, the Product and Customer data are in two **disjointed graphs**, with different labels and relationship types. This is called *Data Federation*. To query across them, you have to federate the graphs, because relationships cannot span across them. This is done by using a *proxy node* modeling pattern: nodes with the `:Product` label must be present in both federated domains. In the Product catalog graph, nodes with the `:Product` label contain all the data related to a product, while in the Customer graphs, the same label is associated to a proxy node, which only contains `productID`. The `productID` property allows you to link data across the graphs in this federation.

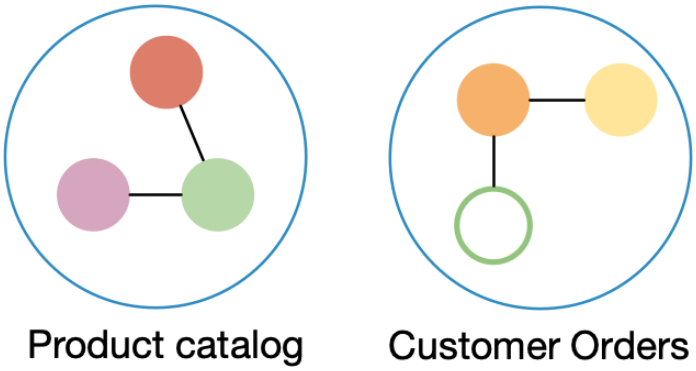


Figure 16. Data Federation

Data Sharding

Since the Customer data is for two regions (EU and AME), you have to partition it into two databases. The resulting two graphs have the same model (same labels, same relationship types), but different data. This is called *Data Sharding*.

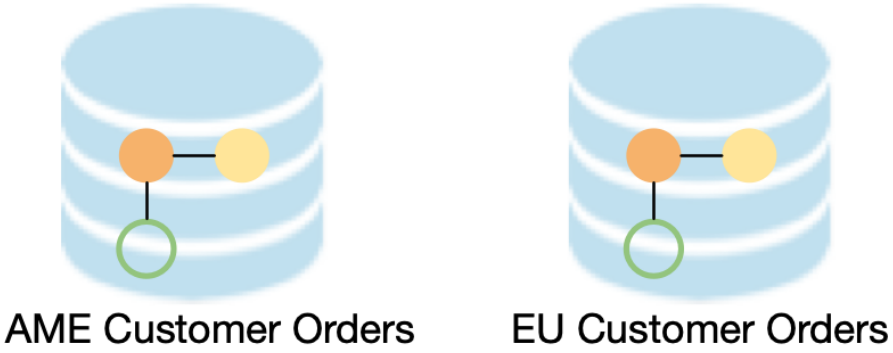


Figure 17. Data Sharding

In general, there are a couple of main use cases that require sharding. The most common is scalability, i.e., different shards can be deployed on different servers, splitting the load on different resources. Another reason could be data regulations: different shards can be deployed on servers, residing in different locations, and managed independently.

Configure Fabric with three databases

Now that you have a new multi-database model defined, you can start to configure the Fabric infrastructure.

This tutorial uses the Linux or macOS tarball installation. It assumes that your current work directory is the *<neo4j-home>* directory of the tarball installation.

Create three databases

You need three databases: `db0` for the Product catalog, `db1` for the EU customer data, and `db2` for the AME customers.

1. Start the Neo4j DBMS.

```
bin/neo4j start
```

2. Check all available databases.

```
ls -al /data/databases/
```

```
total 0
drwxr-xr-x@  5 username  staff   160  9 Jun 12:53 .
drwxr-xr-x@  5 username  staff   160  9 Jun 12:53 ..
drwxr-xr-x  37 username  staff  1184  9 Jun 12:53 neo4j
-rw-r--r--   1 username  staff     0  9 Jun 12:53 store_lock
drwxr-xr-x  38 username  staff  1216  9 Jun 12:53 system
```

3. Connect to the Neo4j DBMS using `cypher-shell` with the default credentials and change the password when prompted. For more information about the Cypher Shell command-line interface (CLI) and how to use it, see Cypher Shell.

```
bin/cypher-shell -u neo4j -p neo4j
```

```
Password change required
new password: *****
Connected to Neo4j 4.1.x at neo4j://localhost:7687 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
```

4. Run the command `SHOW DATABASES` to list all available databases.

```
SHOW DATABASES;
```

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

2 rows available after 102 ms, consumed after another 11 ms
```

5. Run the command `CREATE DATABASE <database-name>` to create the databases.

```
CREATE DATABASE db0;
```

```
0 rows available after 137 ms, consumed after another 0 ms
```

```
CREATE DATABASE db1;
```

```
0 rows available after 14 ms, consumed after another 0 ms
```

```
CREATE DATABASE db2;
```

```
0 rows available after 10 ms, consumed after another 0 ms
```

6. Again run the command `SHOW DATABASES` to verify that the new databases have been created.

```
SHOW DATABASES;
```

```
+-------------------------------------------------------------------------------------------------+
| name     | address          | role         | requestedStatus | currentStatus | error | default |
+-------------------------------------------------------------------------------------------------+
| "db0"    | "localhost:7687" | "standalone" | "online"        | "online"      | ""    | FALSE   |
| "db1"    | "localhost:7687" | "standalone" | "online"        | "online"      | ""    | FALSE   |
| "db2"    | "localhost:7687" | "standalone" | "online"        | "online"      | ""    | FALSE   |
| "neo4j"  | "localhost:7687" | "standalone" | "online"        | "online"      | ""    | TRUE    |
| "system" | "localhost:7687" | "standalone" | "online"        | "online"      | ""    | FALSE   |
+-------------------------------------------------------------------------------------------------+

5 rows available after 8 ms, consumed after another 7 ms
```

7. Exit the Cypher Shell command-line tool.

```
:exit
```

Configure Fabric

You set up Fabric by configuring the fabric database and the graph names and IDs in the *neo4j.conf* file. In this example, the Fabric database is called `fabricnw`.

1. Navigate to the *<neo4j-home>/conf/* folder and open the *neo4j.conf* file.
2. Add the following lines and save it.

```
#*****************************************************************
# Fabric tutorial
#*****************************************************************

fabric.database.name=fabricnw

fabric.graph.0.uri=neo4j://localhost:7687
fabric.graph.0.name=product
fabric.graph.0.database=db0

fabric.graph.1.uri=neo4j://localhost:7687
fabric.graph.1.name=customerEU
fabric.graph.1.database=db1

fabric.graph.2.uri=neo4j://localhost:7687
fabric.graph.2.name=customerAME
fabric.graph.2.database=db2
```

3. Navigate back to the *<neo4j-home>* folder and restart the Neo4j DBMS.

```
bin/neo4j restart
```

4. Connect to the Neo4j DBMS using `cypher-shell` and your credentials.

```
bin/cypher-shell -u neo4j -p your-password
```

5. Run the command `SHOW DATABASES` to verify that the Fabric database has been configured and is `online`.

```
SHOW DATABASES;
```

```
+-----------------------------------------------------------------------------------------------------+
| name       | address          | role         | requestedStatus | currentStatus | error | default |
+-----------------------------------------------------------------------------------------------------+
| "db0"      | "localhost:7687" | "standalone" | "online"        | "online"      | ""    | FALSE   |
| "db1"      | "localhost:7687" | "standalone" | "online"        | "online"      | ""    | FALSE   |
| "db2"      | "localhost:7687" | "standalone" | "online"        | "online"      | ""    | FALSE   |
| "fabricnw" | "localhost:7687" | "standalone" | "online"        | "online"      | ""    | FALSE   |
| "neo4j"    | "localhost:7687" | "standalone" | "online"        | "online"      | ""    | TRUE    |
| "system"   | "localhost:7687" | "standalone" | "online"        | "online"      | ""    | FALSE   |
+-----------------------------------------------------------------------------------------------------+

6 rows available after 242 ms, consumed after another 18 ms
```

Import data in your databases

You can use the command `LOAD CSV WITH HEADERS FROM` to import data in the databases.

Load the Product catalog in db0

1. Run the following Cypher query to change the active database to `db0`, and add the product data.

```
:use db0;

LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/products.csv" AS row
CREATE (n:Product)
SET n = row,
n.unitPrice = toFloat(row.unitPrice),
n.unitsInStock = toInteger(row.unitsInStock), n.unitsOnOrder = toInteger(row.unitsOnOrder),
n.reorderLevel = toInteger(row.reorderLevel), n.discontinued = (row.discontinued <> "0");

LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/categories.csv" AS row
CREATE (n:Category)
SET n = row;

LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/suppliers.csv" AS row
CREATE (n:Supplier)
SET n = row;

CREATE INDEX FOR (p:Product) ON (p.productID);
CREATE INDEX FOR (c:Category) ON (c.categoryID);
CREATE INDEX FOR (s:Supplier) ON (s.supplierID);

MATCH (p:Product),(c:Category)
WHERE p.categoryID = c.categoryID
CREATE (p)-[:PART_OF]->(c);

MATCH (p:Product),(s:Supplier)
WHERE p.supplierID = s.supplierID
CREATE (s)-[:SUPPLIES]->(p);
```

2. Press Enter.
3. Verify that the product data is loaded in `db0`.

```
MATCH (s:Supplier)-[:SUPPLIES]->(p:Product)-[:PART_OF]->(c:Category)
RETURN s.companyName AS Supplier, p.productName AS Product, c.categoryName AS Category
LIMIT 5;
```

```
+--------------------------------------------------------------------------+
| Supplier                    | Product                    | Category    |
+--------------------------------------------------------------------------+
| "Bigfoot Breweries"         | "Sasquatch Ale"            | "Beverages" |
| "Pavlova"                   | "Outback Lager"            | "Beverages" |
| "Bigfoot Breweries"         | "Laughing Lumberjack Lager" | "Beverages" |
| "Bigfoot Breweries"         | "Steeleye Stout"           | "Beverages" |
| "Aux joyeux ecclésiastiques" | "Côte de Blaye"            | "Beverages" |
+--------------------------------------------------------------------------+

5 rows available after 202 ms, consumed after another 5 ms
```

Load EU customers and related orders in db1

1. Run the following Cypher query to change the active database to db1, and add the EU customers and orders.

```
:use db1;

:param europe => ['Germany', 'UK', 'Sweden', 'France', 'Spain', 'Switzerland', 'Austria', 'Italy',
'Portugal', 'Ireland', 'Belgium', 'Norway', 'Denmark', 'Finland'];

LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/customers.csv" AS row
WITH row
WHERE row.country IN $europe
CREATE (n:Customer)
SET n = row;

CREATE INDEX FOR (c:Customer) ON (c.customerID);

LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/orders.csv" AS row
WITH row
MATCH (c:Customer)
WHERE row.customerID = c.customerID
CREATE (o:Order)
SET o = row;

CREATE INDEX FOR (o:Order) ON (o.orderID);

MATCH (c:Customer),(o:Order)
WHERE c.customerID = o.customerID
CREATE (c)-[:PURCHASED]->(o);

LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/products.csv" AS row
CREATE (n:Product)
SET n.productID = row.productID;

CREATE INDEX FOR (p:Product) ON (p.productID);

LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/order-details.csv" AS row
MATCH (p:Product), (o:Order)
WHERE p.productID = row.productID AND o.orderID = row.orderID
CREATE (o)-[details:ORDERS]->(p)
SET details = row, details.quantity = toInteger(row.quantity);
```

2. Press Enter.
3. Verify that the EU Customer orders data is loaded in db1.

```
MATCH (c:Customer)-[:PURCHASED]->(o:Order)-[:ORDERS]->(p:Product)
RETURN c.companyName AS Customer, c.country AS CustomerCountry, o.orderID AS Order, p.productID AS
Product
LIMIT 5;
```

```
+-----------------------------------------------------------+
| Customer              | CustomerCountry | Order   | Product |
+-----------------------------------------------------------+
| "Alfreds Futterkiste" | "Germany"       | "10692" | "63"    |
| "Alfreds Futterkiste" | "Germany"       | "10835" | "77"    |
| "Alfreds Futterkiste" | "Germany"       | "10835" | "59"    |
| "Alfreds Futterkiste" | "Germany"       | "10702" | "76"    |
| "Alfreds Futterkiste" | "Germany"       | "10702" | "3"     |
+-----------------------------------------------------------+

5 rows available after 47 ms, consumed after another 2 ms
```

Load AME customers and related orders in db2

1. Run the following Cypher query to change the active database to db2 and add the AME customers and orders.

```
:use db2;

:param americas => ['Mexico', 'Canada', 'Argentina', 'Brazil', 'USA', 'Venezuela'];

LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/customers.csv" AS row
WITH row
WHERE row.country IN $americas
CREATE (n:Customer)
SET n = row;

CREATE INDEX FOR (c:Customer) ON (c.customerID);

LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/orders.csv" AS row
WITH row
MATCH (c:Customer)
WHERE row.customerID = c.customerID
CREATE (o:Order)
SET o = row;

CREATE INDEX FOR (o:Order) ON (o.orderID);

MATCH (c:Customer),(o:Order)
WHERE c.customerID = o.customerID
CREATE (c)-[:PURCHASED]->(o);

LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/products.csv" AS row
CREATE (n:Product)
SET n.productID = row.productID;

CREATE INDEX FOR (p:Product) ON (p.productID);

LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/order-details.csv" AS row
MATCH (p:Product), (o:Order)
WHERE p.productID = row.productID AND o.orderID = row.orderID
CREATE (o)-[details:ORDERS]->(p)
SET details = row,
details.quantity = toInteger(row.quantity);
```

2. Press Enter.
3. Verify that the AME Customer orders data is loaded in db2.

```
MATCH (c:Customer)-[:PURCHASED]->(o:Order)-[:ORDERS]->(p:Product)
RETURN c.companyName AS Customer, c.country AS CustomerCountry, o.orderID AS Order, p.productID AS
Product
LIMIT 5;
```

```
+-------------------------------------------------------------------------------+
| Customer                              | CustomerCountry | Order   | Product |
+-------------------------------------------------------------------------------+
| "Ana Trujillo Emparedados y helados" | "Mexico"        | "10759" | "32"    |
| "Ana Trujillo Emparedados y helados" | "Mexico"        | "10926" | "72"    |
| "Ana Trujillo Emparedados y helados" | "Mexico"        | "10926" | "13"    |
| "Ana Trujillo Emparedados y helados" | "Mexico"        | "10926" | "19"    |
| "Ana Trujillo Emparedados y helados" | "Mexico"        | "10926" | "11"    |
+-------------------------------------------------------------------------------+

5 rows available after 42 ms, consumed after another 1 ms
```

Retrieve data with a single Cypher query

Fabric allows you to retrieve data from all your databases with a single Cypher query.

As the databases `db0`, `db1`, `db2` in this tutorial are part of the same Neo4j DBMS, you can also access them directly, using their database names. This is especially useful when you want to set up Fabric locally for development or testing purposes. In this case, you only have to add `fabric.database.name=fabricnw` to the *neo4j.conf* file, and use queries as the following one.

```
:use fabricnw
```

```
USE db1
MATCH (c:Customer)
WHERE c.customerID STARTS WITH 'A'
RETURN c.customerID AS name, c.country AS country
  UNION
USE db2
MATCH (c:Customer)
WHERE c.customerID STARTS WITH 'A'
RETURN c.customerID AS name, c.country AS country
LIMIT 5;
```

```
+---------------------+
| name    | country   |
+---------------------+
| "ALFKI" | "Germany" |
| "AROUT" | "UK"      |
| "ANATR" | "Mexico"  |
| "ANTON" | "Mexico"  |
+---------------------+

4 rows available after 404 ms, consumed after another 1 ms
```

However, if your databases `db0`, `db1`, `db2` are located in other Neo4j DBMSs, on completely different servers for example, then you must update the URI settings to connect to them.

In this tutorial, you will try the Fabric capabilities as if the data is deployed on different servers.

Query a single database

You can retrieve data from a single database by using the cypher clause `USE` and the name of the Fabric graph. When querying a single database, you do not have to change the active database to Fabric.

```
USE fabricnw.product
MATCH (p:Product)
RETURN p.productName AS product
LIMIT 5;
```

```
+--------------------------------+
| product                        |
+--------------------------------+
| "Chai"                         |
| "Chang"                        |
| "Aniseed Syrup"                |
| "Chef Anton's Cajun Seasoning" |
| "Chef Anton's Gumbo Mix"       |
+--------------------------------+

5 rows available after 6 ms, consumed after another 21 ms
```

Query across multiple shards

Use Fabric to query both shards and get customers whose name starts with A.

When you want to retrieve data from multiple databases, you have to change the active database to `fabricnw`.

```
:use fabricnw
```

```
USE fabricnw.customerAME
MATCH (c:Customer)
WHERE c.customerID STARTS WITH 'A'
RETURN c.customerID AS name, c.country AS country
  UNION
USE fabricnw.customerEU
MATCH (c:Customer)
WHERE c.customerID STARTS WITH 'A'
RETURN c.customerID AS name, c.country AS country
LIMIT 5;
```

```
+---------------------+
| name    | country   |
+---------------------+
| "ANATR" | "Mexico"  |
| "ANTON" | "Mexico"  |
| "ALFKI" | "Germany" |
| "AROUT" | "UK"      |
+---------------------+

4 rows available after 25 ms, consumed after another 56 ms
```

Or, using a more common Fabric idiom:

```
UNWIND [1,2]AS gid
CALL {
    USE fabricnw.graph(gid)
    MATCH (c:Customer)
    WHERE c.customerID STARTS WITH 'A'
    RETURN c.customerID AS name, c.country AS country
}
RETURN name, country
LIMIT 5;
```

```
+---------------------+
| name    | country   |
+---------------------+
| "ANATR" | "Mexico"  |
| "ANTON" | "Mexico"  |
| "ALFKI" | "Germany" |
| "AROUT" | "UK"      |
+---------------------+

4 rows available after 61 ms, consumed after another 8 ms
```

Query across federation and shards

Finally, a more complex query that uses all 3 databases to find all customers who have bought discontinued products in the Meat/Poultry category.

```
CALL {
USE fabricnw.product
MATCH (p:Product{discontinued:true})-[:PART_OF]->(c:Category{categoryName:'Meat/Poultry'})
              RETURN COLLECT(p.productID) AS pids
}
WITH *, [g IN fabricnw.graphIds() WHERE g<>0] AS gids
UNWIND gids AS gid
CALL {
    USE fabricnw.graph(gid)
    WITH pids
    UNWIND pids as pid
    MATCH (p:Product{productID:pid})<-[:ORDERS]-(:Order)<-[:PURCHASED]-(c:Customer)
    RETURN DISTINCT c.customerID AS customer, c.country AS country
}
RETURN customer, country
LIMIT 20;
```

```
+--------------------------+
| customer | country       |
+--------------------------+
| "RICSU"  | "Switzerland" |
| "PERIC"  | "Mexico"      |
| "WARTH"  | "Finland"     |
| "WELLI"  | "Brazil"      |
| "DRACD"  | "Germany"     |
| "RATTC"  | "USA"         |
| "HUNGO"  | "Ireland"     |
| "QUEDE"  | "Brazil"      |
| "SEVES"  | "UK"          |
| "ANTON"  | "Mexico"      |
| "BERGS"  | "Sweden"      |
| "SAVEA"  | "USA"         |
| "AROUT"  | "UK"          |
| "FAMIA"  | "Brazil"      |
| "WANDK"  | "Germany"     |
| "WHITC"  | "USA"         |
| "ISLAT"  | "UK"          |
| "LONEP"  | "USA"         |
| "QUICK"  | "Germany"     |
| "HILAA"  | "Venezuela"   |
+--------------------------+

20 rows available after 51 ms, consumed after another 2 ms
```

First, `fabricnw` calls database `db0` to retrieve all discontinued products in the Meat/Poultry category. Then, using the returned product IDs, it queries both `db1` and `db2` **in parallel** and gets the customers who have purchased these products and their country.

The end

You have just learned how to store and retrieve data from multiple databases using a single Cypher query. For more details on the Neo4j Fabric, see Fabric.

Appendix C: Advanced Causal Clustering

This section includes information about advanced deployments and configuration options for multi-data center operations.

- Causal Clustering lifecycle — A walk-through of the lifecycle of a cluster.
- Multi-data center — Overview of the multi-data center section.
  - Licensing for multi-data center operations — Information about licensing for multi-data center operations.
  - Multi-data center design — Patterns for multi-data center deployments.
  - Multi-data center operations — Configuration options for multi-data center deployments.
  - Multi-data center load balancing — Configuration options for making client applications aware of multi-data center topologies.
  - Data center disaster recovery — How to recover a cluster to full working capability after data center loss.
- Embedded usage — How to embed a Neo4j Causal Cluster in your application.

For details on the configuration and operation of a Neo4j Causal Cluster, see Clustering.

For descriptions of settings related to running a Neo4j Causal Cluster, see Settings reference.

16.C.1. Causal Clustering lifecycle

This section includes:

- Introduction
- Discovery protocol
- Core membership
- Read Replica membership
- Transacting via the Raft protocol
- Catchup protocol
- Read Replica shutdown

- Core shutdown

Introduction

In this section we will develop some deeper knowledge of how the cluster operates. By developing our understanding of how the cluster works we will be better equipped to design, deploy, and troubleshoot our production systems.

Our in-depth tour will follow the lifecycle of a cluster. We will boot a Core cluster and pick up key architectural foundations as the cluster forms and transacts. We will then add in Read Replicas and show how they bootstrap join the cluster and then catchup and remain caught up with the Core Servers. We will then see how backup is used in live cluster environments before shutting down Read Replicas and Core Servers.

Discovery protocol

The discovery protocol is the first step in forming a Causal Cluster. It takes in some information about existing Core cluster servers, and uses this to initiate a network join protocol.

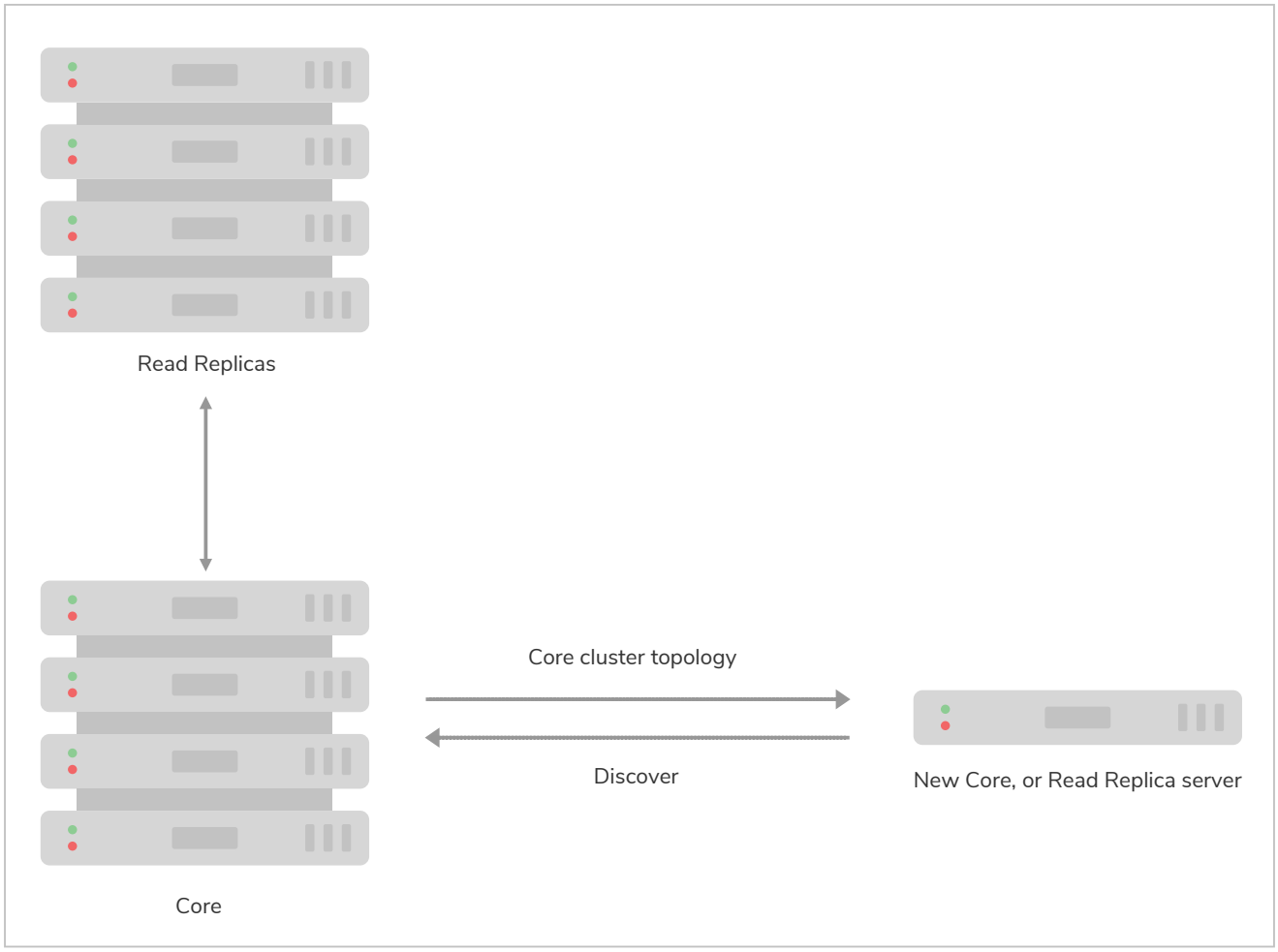


Figure 18. Causal Cluster discovery protocol: Core-to-Core or Read Replica-to-Core only.

Using this information, the server will either join an existing cluster or form one of its own.

The discovery protocol targets Core Servers only regardless of whether it is a Core Server or Read Replica performing discovery. It is because we expect Read Replicas to be both numerous and, relatively speaking, transient whereas Core Servers will likely be fewer in number and relatively stable over time.

The discovery protocol takes information from `causal_clustering.initial_discovery_members` in *neo4j.conf*, which lists which IP addresses and ports that form the cluster on startup. Detailed information about discovery and discovery configuration options is given in the Initial discovery of cluster members section. When consuming this information, the server will try to handshake with the other listed servers. On successful handshake with another server (or servers), the current server will discover the whole current topology.

The discovery protocol continues to run throughout the lifetime of the Causal Cluster and is used to maintain the current state of available servers and to help clients route queries to an appropriate server via the client-side drivers.

Core membership

If it is a Core Server that is performing discovery, once it has made a connection to the one of the existing Core Servers, it then joins the Raft protocol. Each database is replicated by a logically separate Raft group, so the process below is repeated for every one.

Raft is a distributed algorithm for maintaining a consistent log across multiple shared-nothing servers designed by Diego Ongaro for his 2014 Ph.D. thesis. See the Raft thesis for details.

Raft handles cluster membership by making it a normal part of keeping a distributed log in sync. Joining a cluster involves the insertion of a cluster membership entry into the Raft log which is then reliably replicated around the existing cluster. Once that entry is applied to enough members of the Raft consensus group (those machines running the specific instance of the algorithm), they update their view of the cluster to include the new server. Thus membership changes benefit from the same safety properties as other data transacted via Raft (see Transacting via the Raft protocol for more information).

The new Core Server must also catch up its own Raft logs with respect to the other Core Servers as it initializes its internal Raft instance. This is the normal case when a cluster is first booted and has performed few operations. There will be a delay before the new Core Server becomes available if it also needs to catch up (as per Catchup protocol) graph data from other servers. This is the normal case for a long lived cluster where the servers holds a great deal of graph data.

Where a joining Neo4j instance has databases whose names match databases which already exist in the cluster, the database stores on the joining instance must be the same as their counterparts on cluster members (although they are allowed to be in previous states). For example, if a cluster contains a database named *products*, a new instance may join with a backup of *products*, but not a database named *products* with different contents. A new instance may also join a cluster if it does not contain any matching databases.

The described catchup process is repeated for each database which exists in the cluster.

Read Replica membership

When a Read Replica performs discovery, once it has made a connection to any of the available Core clusters it proceeds to add itself into a shared whiteboard.

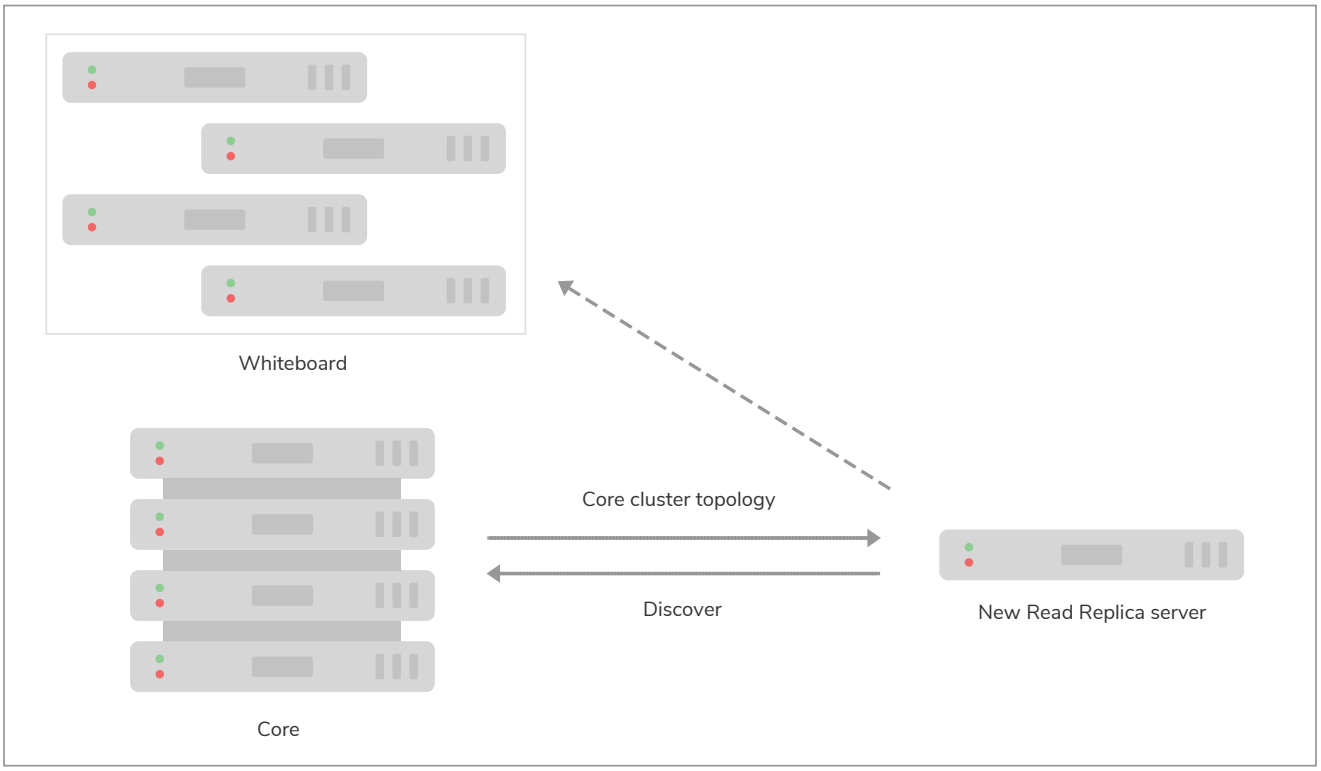


Figure 19. All Read Replicas registered with shared whiteboard.

This whiteboard provides a view of all live Read Replicas and is used both for routing requests from database drivers that support end-user applications and for monitoring the state of the cluster.

The Read Replicas are not involved in the Raft protocol, nor are they able to influence cluster topology. Hence a shared whiteboard outside of Raft comfortably scales to very large numbers of Read Replicas.

The whiteboard is kept up to date as Read Replicas join and leave the cluster, even if they fail abruptly rather than leaving gracefully.

Transacting via the Raft protocol

Once bootstrapped, each Core Server spends its time processing database transactions. Updates are reliably replicated around Core Servers via the Raft protocol. Updates appear in the form of a (committed) Raft log entry containing transaction commands which is subsequently applied to update the database.

One of Raft's primary design goals is to be easily understandable so that there are fewer places for tricky bugs to hide in implementations. As a side-effect, it is also possible for database operators to reason about their Core Servers in their Causal Clusters.

The Raft Leader for the current term (a logical clock) appends the transaction (an 'entry' in Raft terminology) to the head of its local log and asks the other instances to do the same. When the Leader can see that a majority instances have appended the entry, it can be considered committed into the Raft log. The client application can now be informed that the transaction has safely committed since there is

sufficient redundancy in the system to tolerate any (non-pathological) faults.

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.

As each database operates within a logically separate Raft group, a core server can have multiple roles: one for each database. For example, it may be the *Leader* for database `system` and at the same time be a *Follower* for database `neo4j`.

For safety, within any Raft protocol instance there is only one Leader able to make forward progress in any given term. The Leader bears the responsibility for imposing order on Raft log entries and driving the log forward with respect to the *Followers*.

Followers maintain their logs with respect to the current Leader's log. Should any participant in the cluster suspect that the Leader has failed (not receiving new entries or heartbeats), then they can instigate a leadership election by entering the *Candidate* state. In Neo4j Core Servers this failure detection window is set by default above 20s to enable more stable leaders.

Whichever instance is in the best state (including the existing Leader, if it remains available) can emerge from the election as Leader. The "best state" for a Leader is decided by highest term, then by longest log, then by highest committed entry.

The ability to fail over roles without losing data allows forward progress even in the event of faults. Even where Raft instances fail, the protocol can rapidly piece together which of the remaining instances is best placed to take over from the failed instance (or instances) **without data loss**. This is the essence of a *non-blocking* consensus protocol which allows Neo4j Causal Clustering to provide continuous availability to applications.

Catchup protocol

Read Replicas spend their time concurrently processing graph queries and applying a stream of transactions from the Core Servers to update their local graph store.

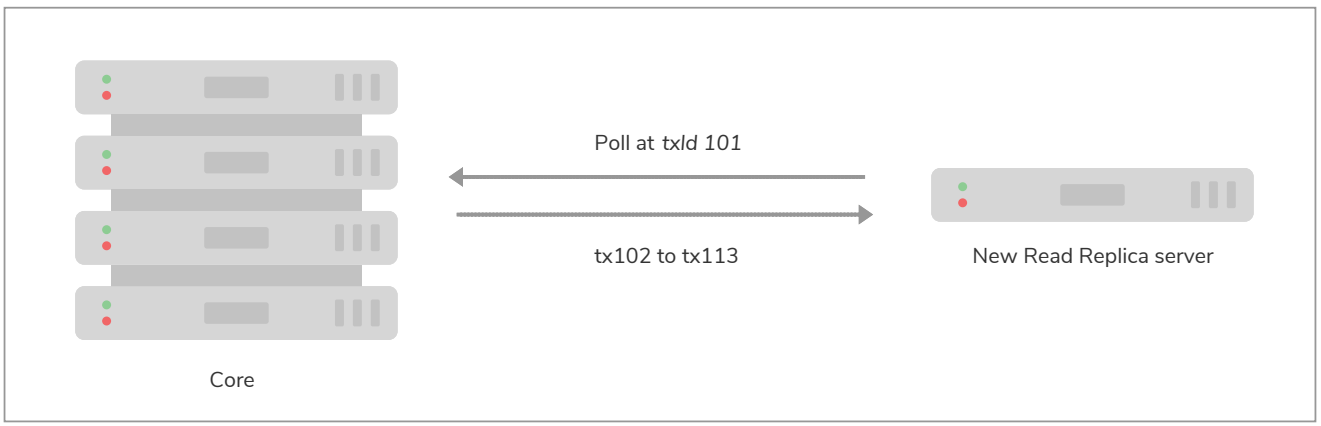


Figure 20. Transactions shipped from Core to Read Replica.

Updates from Core Servers to Read Replicas are propagated by transaction shipping. Transaction shipping is instigated by Read Replicas frequently *polling* any of the Core Servers specifying the ID of the last transaction they received and processed. The frequency of polling is an operational choice.

Neo4j transaction IDs are strictly monotonic integer values (they always increase). This makes it possible to determine whether or not a transaction has been applied to a Read Replica by comparing its last processed transaction ID with that of a Core Server.

If there is a large difference between an Read Replica's transaction history and that of a Core Server, polling may not result in any transactions being shipped. This is quite expected, for example when a new Read Replica is introduced to a long-running cluster or where a Read Replica has been down for some significant period of time. In such cases the catchup protocol will realize the gap between the Core Servers and Read Replica is too large to fill via transaction shipping and will fall back to copying the database store directly from Core Server to Read Replica. Since we are working with a live system, at the end of the database store copy the Core Server's database is likely to have changed. The Read Replica completes the catchup by asking for any transactions missed during the copy operation before becoming available.

A very slow database store copy could conceivably leave the Read Replica too far behind to catch up via transaction log shipping as the Core Server has substantially moved on. In such cases the Read Replica server repeats the catchup protocol. In pathological cases the operator can intervene to snapshot, restore, or file copy recent store files from a fast backup.

Read Replica shutdown

On clean shutdown, a Read Replica will invoke the discovery protocol to remove itself from the shared whiteboard overview of the cluster. It will also ensure that the database is cleanly shutdown and consistent, immediately ready for future use.

On an unclean shutdown such as a power outage, the Core Servers maintaining the overview of the cluster will notice that the Read Replica's connection has been abruptly been cut. The discovery machinery will initially hide the Read Replica's whiteboard entry, and if the Read Replica does not reappear quickly its modest memory use in the shared whiteboard will be reclaimed.

On unclean shutdown it is possible the Read Replica will not have entirely consistent store files or transaction logs. On subsequent reboot the Read Replica will rollback any partially applied transactions such that the database is in a consistent state.

Core shutdown

A shutdown of a Core Server, like Core Server booting, is handled via the Raft protocol. When a member is shutdown, either cleanly or by force, it will eventually be voted out from the Raft group. All remaining instances accept that the cluster has grown smaller, and is therefore less fault tolerant. For any databases where the leaver was playing the *Leader* role, each of those leaderships will be transferred to other Core Servers. Once the new Leader is established, the Core cluster continues albeit with less redundancy.

If more members than the current fault tolerance leaves the cluster within a very short time period, the cluster cannot proceed and will lose quorum. However, if members are gradually lost, the cluster may have

time to reduce the size of the cluster. A Core cluster of 5 members reduced to 3 can still continue operate normally with a fault tolerance reduced from 2 to 0. After the Raft protocol votes out the lost members which reduces the cluster size to 3, our fault tolerance has been increased from 0 to 1, and can lose yet another member and keep operating. This is because the Raft protocol has had time to vote out the lost members, and changed the cluster size of 5 (fault tolerance of 2) to 3 (fault tolerance of 1).

Raft may only reduce a cluster size to the configured causal_clustering.minimum_core_cluster_size_at_runtime. Once the cluster has reached this size, it will stop voting out members.

16.C.2. Multi-data center

Some use cases present high needs for availability, redundancy, locality of client applications, or simply scale. In these cases it is important that the cluster is aware of its physical topology so that it can optimize for workload. This makes configuring a single cluster to span multiple data centers a necessary proposition.

The following sections are dedicated to describing the different aspects of multi-data center operations of a Causal Cluster.

- Licensing for multi-data center operations
- Multi-data center design
  - Introduction
  - Core Server deployment scenarios
  - Allowing Read Replicas to catch up from other Read Replicas
- Multi-data center operations
  - Enable multi-data center operations
  - Server groups
  - Strategy plugins
- Multi-data center load balancing
  - Introduction
  - Prerequisite configuration
  - The load balancing framework
  - Load balancing examples
- Data center disaster recovery
  - Data center loss scenario
  - Procedure for recovering from data center loss

Licensing for multi-data center operations

Multi-data center functionality is intended for very demanding users of Neo4j who typically operate under a commercial database license. As a result, multi-data center functionality is licensed separately from the

single-data center Causal Clustering features.

In order to confirm that you are operating under a suitable license, you must explicitly set the following in *neo4j.conf*:

```
causal_clustering.multi_dc_license=true
```

Without this configuration, all of the multi-data center features will remain disabled.

16.C.3. Multi-data center design

This section describes the following:

- Introduction
- Core Server deployment scenarios
- Allowing Read Replicas to catch up from other Read Replicas
  - Hierarchical Read Replica deployment
  - Catch up (mostly) from peer Read Replicas
  - Maintaining causal consistency in scale-out topologies

Introduction

This section is based on a series of examples to illustrate the different considerations we should take into account when designing our Causal Cluster for a multi-data center environment. We'll come to understand the weaknesses and benefits of common multi-data center deployment scenarios. Each scenario is presented at a high architectural level for clarity. In subsequent sections we will go into more detail on how such deployments are configured.

Core Server deployment scenarios

We will start with the conceptually simplest multi-data center scenario where we deploy the same number and kind of instances into each DC. This is a *homogeneous* deployment because each data center is identical to the other.

Example 133. Homogeneous three data center deployment

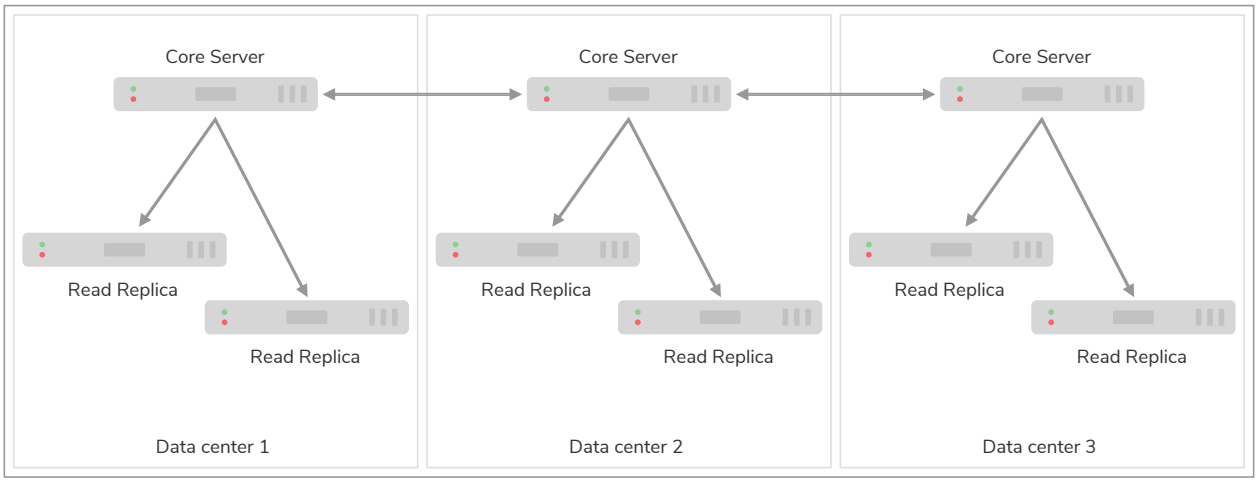


Figure 21. Homogeneous deployment across three data centers with one Core instance in each

In diagram above we have three data centers, each identically equipped with a single Core Server and a small number of Read Replicas.

Since Raft only requires a majority of the instances to acknowledge a write before it is safely committed, the latency of the commit path for this pattern involves only the two fastest data centers. As such the cost of committing to this setup is two WAN messages: one to send the transaction and one ACK message. In a non-failure case the other data center will not be far behind and will apply the transaction as well.

Within each of the data centers we can increase machine-level redundancy by adding more Core instances. For example we could add two more machines in each data center so that we can tolerate the spontaneous loss of up to four machines anywhere in the cluster or a single data center as a whole.

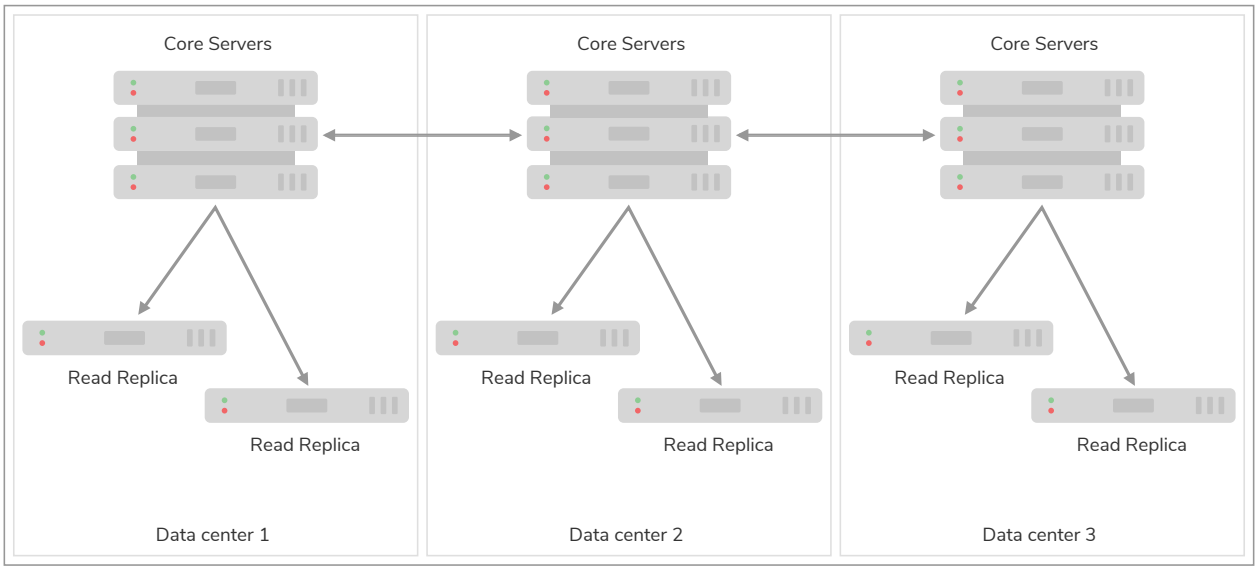


Figure 22. Homogeneous deployment across three data centers with three Core instances in each

To recap the strengths and weaknesses of this deployment pattern:

- We can lose an entire data center without losing availability and, depending on the number of

machines in each data center, we may still be able to tolerate the loss of individual servers regardless of which data center they are in.

- The commit path for transactions is short, just two WAN messages exchanged.
- While the loss of majority data centers will need to be recovered, the operational procedure is identical irrespective of which of the data centers are lost.

As will be shown in the section on multi-data center configuration the Read Replicas can be biased to catchup from their data center-local Core Servers to minimize catchup latency. Data center-local client applications would also likely be routed to those same Read Replicas both for topological locality and scaling. More details are available in the section on multi-data center load balancing.

In the two data center case, our first instinct is to balance the available servers for operational consistency. An example of a homogeneous deployment across two data centers with two Core instances in each is illustrated in the diagram below:

Example 134. Homogeneous two data center deployment

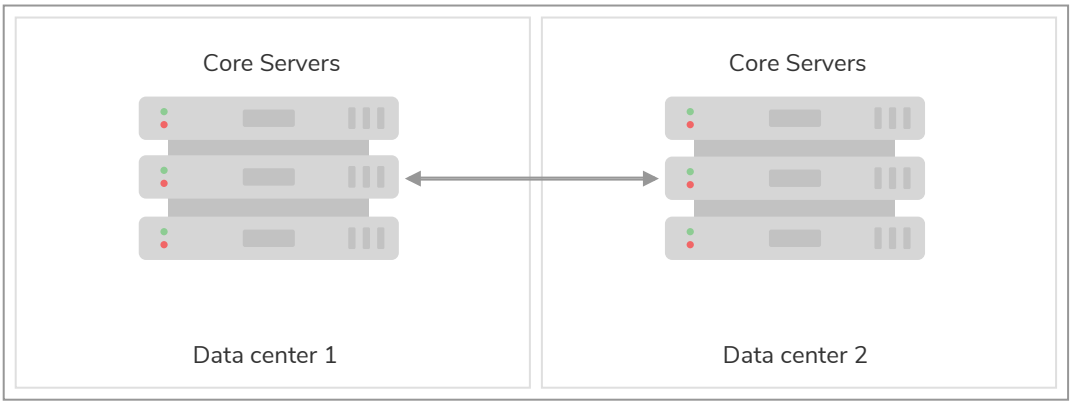


Figure 23. Homogeneous deployment across two data centers

The problem with this configuration is that while architecturally simple, it does not play to the strengths of the Raft protocol which is based on majority consensus. In the non-failure case, we incur two WAN messages to commit any transaction because a majority commit implies at least one response from the non-local data center instances. Worse, if we lose either data center the cluster will become read-only because it is impossible to achieve a majority.

As seen in the example above, the homogeneous deployment over two data centers does not take full advantage of the strengths of Causal Clustering. However it guarantees that the full Raft log will be present in either data center in the case of total data center loss.

The opposite of spreading Core Servers around our data centers, is to have them all hosted in a single one. This may be for technical or governance reasons, but either way has the advantage of LAN commit latencies for writes.

While our Core Servers are colocated, we spread out our Read Replicas close to the client applications to enable fan-out scaling.

Example 135. Core Servers and Read Replicas segregated by data center

The diagram below shows an example of a heterogeneous deployment directing writes to one data center, and reads to all. This pattern provides high survivability for data because of geo-replication. It also provides locality for client applications. However, if the Core Server data center is lost, we must immediately instigate recovery and turn one of the remaining Read Replica data centers into a new Core cluster.

It is possible that none of the Read Replicas have received all of the confirmed transactions prior to losing Data Center 1. While this is a convenient pattern for geo-replication, its semantics are best-effort. Cluster designers must take this aspect under consideration when deciding on recovery strategy.

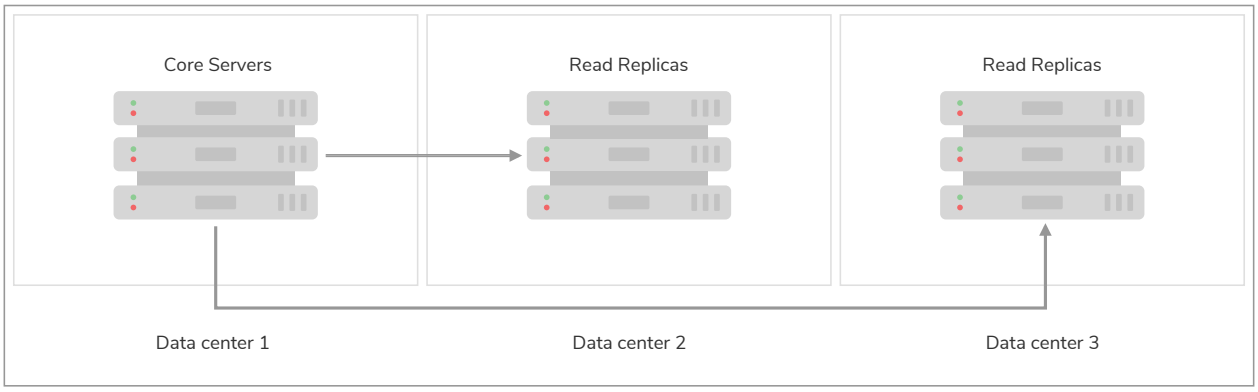


Figure 24. Heterogeneous deployment separating Read Replicas from the Core cluster

An operational tweak to this approach would be to host a Core Server in Data Center 2 and 3 as the starting point for recovery. During normal operations, these extra Core Servers should be configured with `causal_clustering.refuse_to_be_leader=true`. Should we lose Data Center 1, then we can use one of these Core Servers to quickly bootstrap a new Core cluster and return to full service rapidly.

To recap the strengths of this deployment pattern:

- Core Servers commit at LAN latencies if using the setup with Core Servers exclusively in one data center.
- Read Replicas provide scale and locality for client applications.
- Geo-replication provides high survivability for data.

Allowing Read Replicas to catch up from other Read Replicas

With an understanding of the basic multi-data center patterns at our disposal, we can refine our deployment models to embrace local catchup within data centers. This means that any server, including Read Replicas, can act as a source of transactions for Read Replica server. When catching up from data center-local instances we aim to amortize the cost of WAN traffic catchup across many local replications.

Allowing Read Replicas to choose a data center-local Core Server or even another Read Replica gives us a great deal of design freedom, and importantly allows us to scale to truly huge numbers of Read Replicas. Using this feature we might choose to fan-out Read Replicas so that the catchup load on the Core Servers grows (approximately) logarithmically rather than linearly.

Hierarchical Read Replica deployment

The primary motivation for Read Replicas catching up from other Read Replicas is to allow for fan-out scale. To achieve a fan-out we arrange the Read Replicas in a hierarchy, with each layer of the hierarchy being broader than the one above.

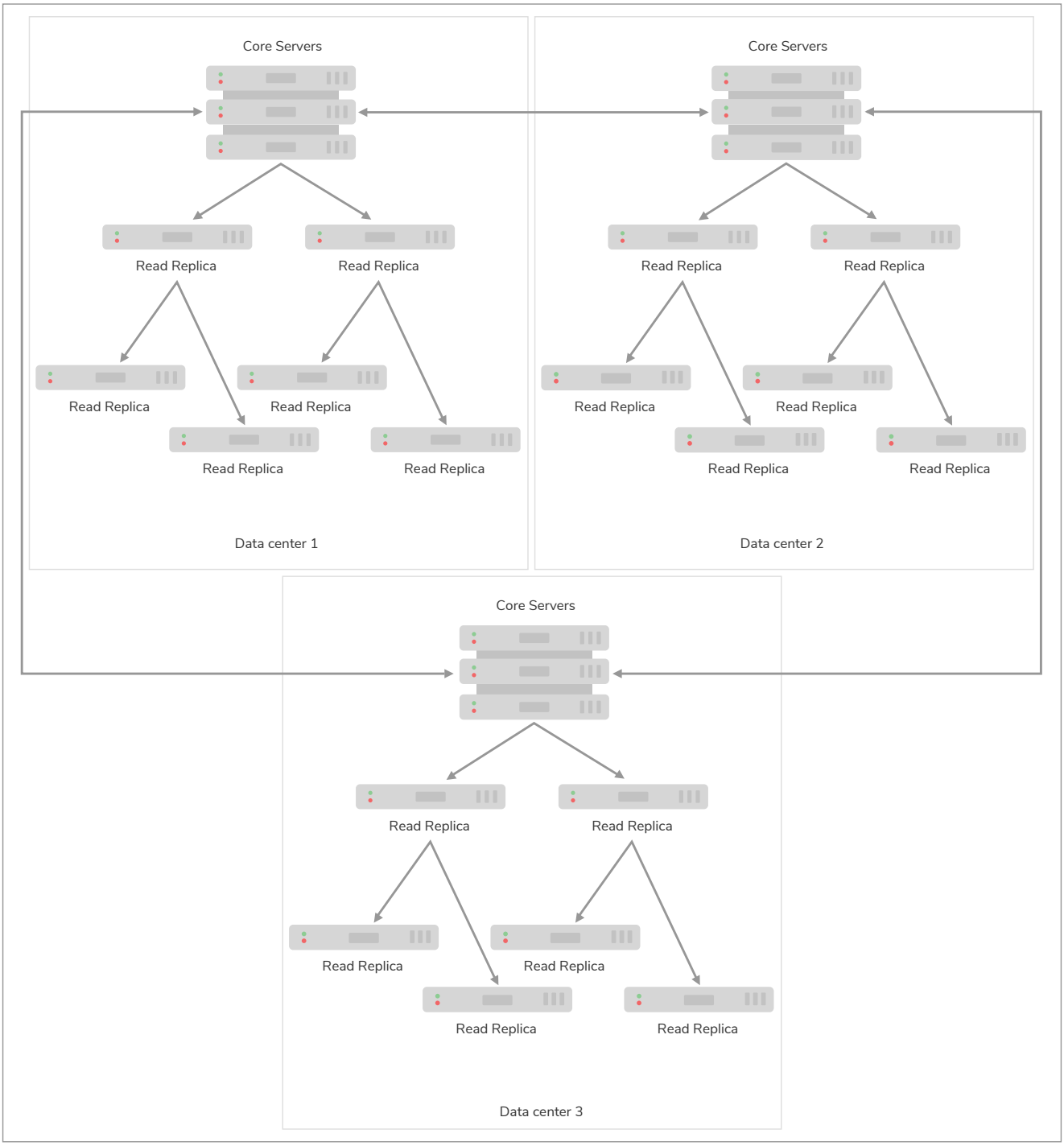


Figure 25. Fan out from Core Servers for scale at log cost

An illustrative hierarchy is presented in the diagram above. The Core Servers supply transactions to a relatively small number of Read Replicas at the first tier. This results in a relatively modest load on the Core Servers, freeing up resources to focus on the commit path. Those Read Replicas in the first tier in turn feed a larger number of Read Replicas in the second tier. This pattern can be reasonably extended to several tiers to provide enormous fan-out.

At each tier we expand the scalability of the Read Replicas, but we add another level of catchup latency. By careful measurement we can ascertain the appropriate depth and breadth of the hierarchy to match the application requirements.

We should also take care that each tier in the hierarchy has sufficient redundancy so that failures do not compromise transmission of data from the Core Servers. A strategy for keeping Read Replicas current in the presence of failures is to occasionally have them subvert the hierarchy. That is, if a given Read Replica occasionally goes to its grandparents or even directly to the Core Servers then we can avoid pathologically high replication latencies under fault conditions.

Catch up (mostly) from peer Read Replicas

Another strategy for Read Replica catchup is to treat them all as peers and have peer-to-peer catchup. This avoids the need to manage tiers of replicas to maintain availability since the Read Replicas catch up from one another in a mesh.

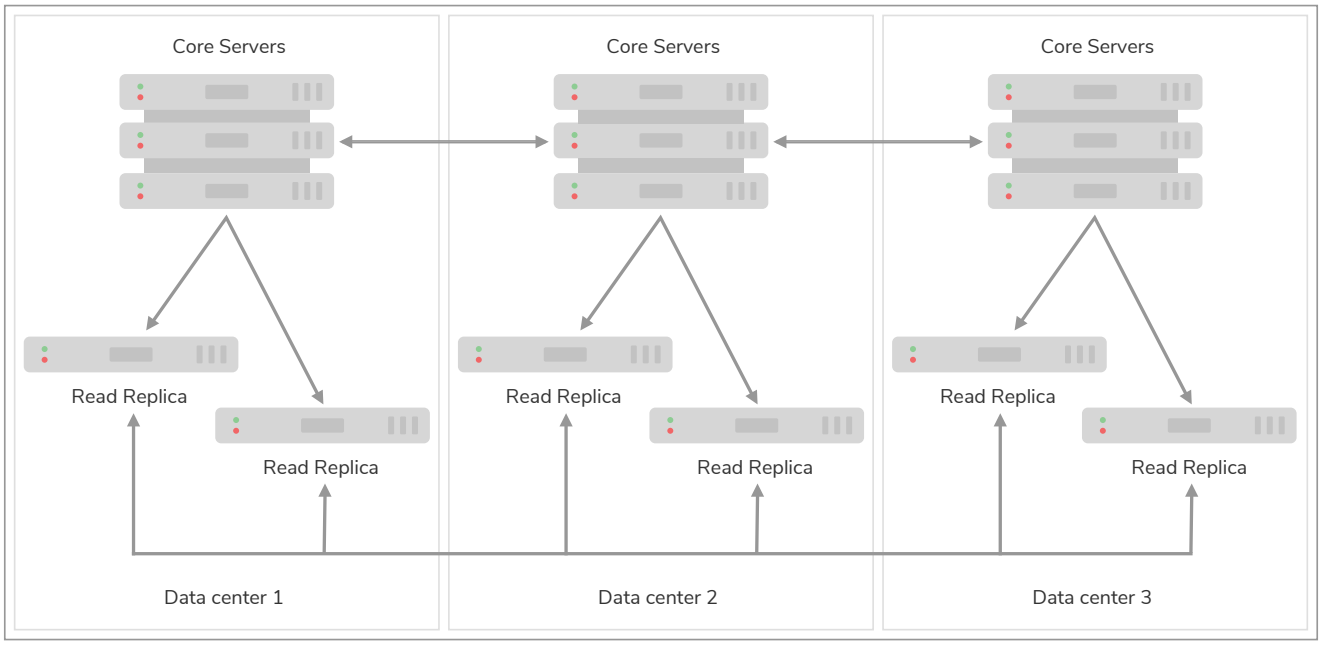


Figure 26. Peer-to-peer Read Replica catchup

Having a reduced load on the Core Servers allows us to scale out. For example if only one in ten catchup requests goes to the Core Servers, we could expand the number of Read Replicas by approximately a factor of 10.

To avoid groups of orphans in the mesh, Read Replicas will occasionally catch up directly from Core Servers. Having Read Replicas catch up with Core Servers ensures that no Read Replica is left behind indefinitely, placing an upper bound on replication latency. While this places some load on the Core Servers, it is far less than if all catch up attempts from Read Replicas were directed to a Core Server.

The upper bound on replication latency for this mode of operation is the number of catchup attempts served by Read Replicas before trying core. The average replication latency will be half the number of attempts to replicate. This is because on average half the Read Replicas will be ahead and half behind any given Read Replica.

Connecting to a random Core Server on failure to retrieve updates from other sources is the default behavior of Read Replicas.

Maintaining causal consistency in scale-out topologies

Causal consistency is always maintained, even in extreme situations with chains of Read Replicas catching up from other upstream Read Replicas. The key trade-off to understand, as so often in distributed systems, is that of latency for scale.

In Fan out from Core Servers for scale at log cost", role="middle we see that number of hops required for a transaction to propagate to the lowest tier is 2: the highest latency in this topology. Equally we see how the bottommost tier has far more members than any other tier giving it scale advantages.

Correspondingly, in the middle tier we have better latency (one hop) but less scale. At the top most tier (Core Servers) we have very little latency (just the Raft commit path) but the fewest available servers. This means we should target queries at the most appropriate tier based on latency, scale, and locality.

Summary on latency versus scalability:

- Issuing read queries to a Core Server generally has the lowest latency in principle but may have the highest contention.
- Issuing read queries to a Read Replica topologically closest to Core Servers typically has higher latency but also higher scalability.
- Issuing read queries to a Read Replica topologically further from Core Servers typically has the highest latency but also the highest scalability.

In large systems like the scale-out hierarchy above, we are conventionally used to having relaxed or *eventual* consistency semantics. With Neo4j multi-data center setups, that is also possible. Where we don't care about causality we can read from any Read Replica and accept that we might see older values. However the causal consistency semantics are maintained.

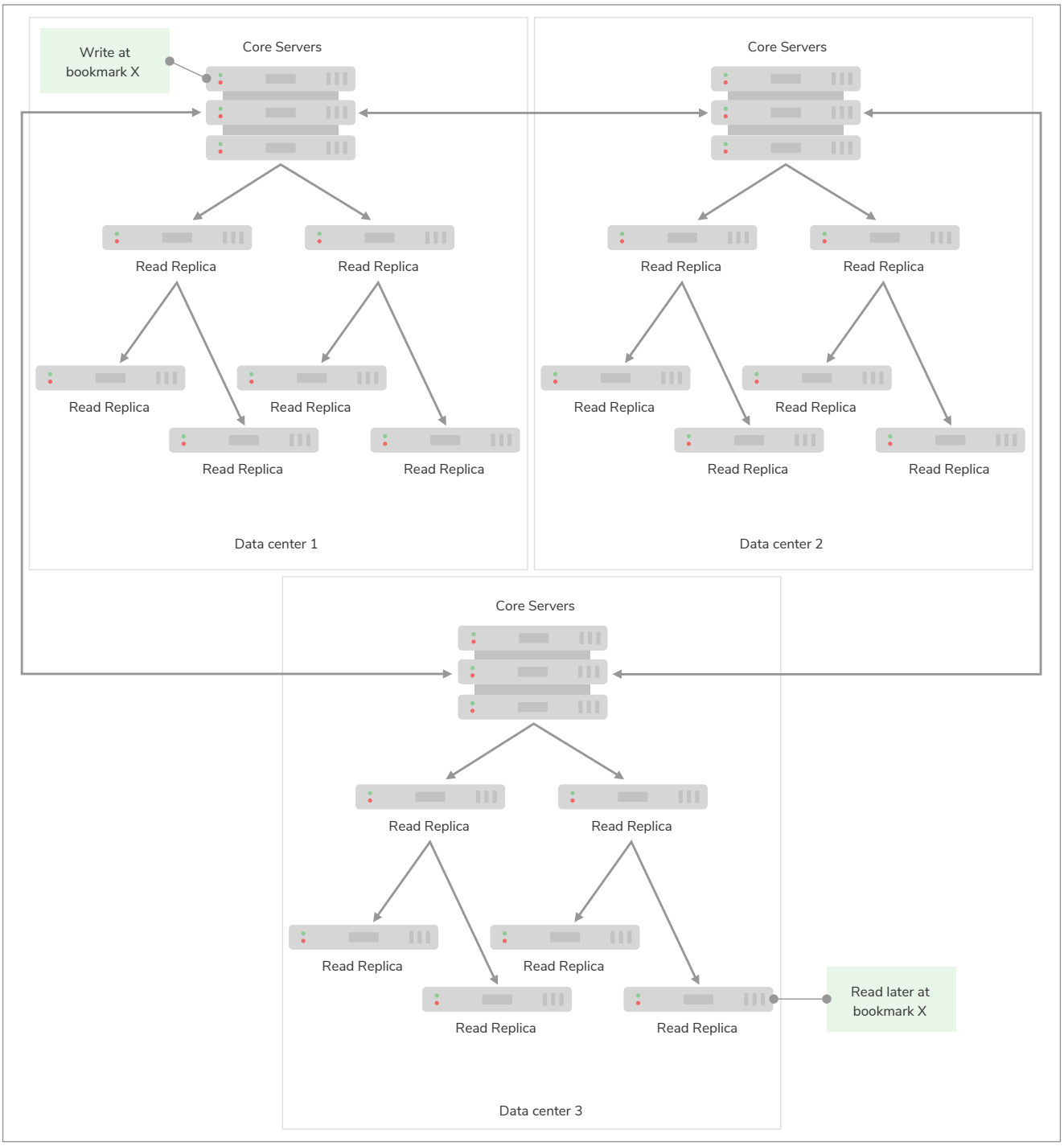


Figure 27. Each tier in the Read Replicas is further behind the source of truth, but offers greater scale-out

As we can see in diagram above, even if the client binds to a Read Replica that is multiple hops/data centers away from the source of truth, causal consistency is maintained. While the query may be suspended while the necessary transaction propagates to the Read Replica, the benefit is that there will be more Read Replicas available and so overall client throughput is higher than with a single-tier configuration.

16.C.4. Multi-data center operations

This section describes the following:

- Enable multi-data center operations

Enable multi-data center operations

Before doing anything else, we must enable the multi-data center functionality. This is described in Licensing for multi-data center operations.

Licensing for multi-data center

The multi-data center functionality is separately licensed and must be specifically enabled.

Server groups

In order to optimize the use of our Causal Cluster servers according to our specific requirements, we sort them into *Server Groups*. Server Group membership can map to data centers, availability zones, or any other significant topological elements from the operator's domain. Server Groups can also overlap.

Server Groups are defined as a key that maps onto a set of servers in a Causal Cluster. Server Group membership is defined on each server using the `causal_clustering.server_groups` parameter in *neo4j.conf*. Each server in a Causal Cluster can belong to zero or more server groups.

Example 136. Definition of Server Group membership

The membership of a server group or groups can be set in *neo4j.conf* as in the following examples:

```
# Add the current instance to the groups `us` and `us-east`
causal_clustering.server_groups=us,us-east
```

```
# Add the current instance into the group `london`
causal_clustering.server_groups=london
```

```
# Add the current instance into the group `eu`
causal_clustering.server_groups=eu
```

We must be aware that membership of each server group is explicit. For example, a server in the `gb-london` group is not automatically part of some `gb` or `eu` group unless that server is explicitly added to those groups. That is, any (implied) relationship between groups is reified only when those groups are used as the basis for requesting data from upstream systems.

Server Groups are not mandatory, but unless they are present, we cannot set up specific upstream

transaction dependencies for servers. In the absence of any specified server groups, the cluster defaults to its most pessimistic fall-back behavior: each Read Replica will catch up from a random Core Server.

Strategy plugins

Strategy plugins are sets of rules that define how Read Replicas contact servers in the cluster in order to synchronize transaction logs. Neo4j comes with a set of pre-defined strategies, and also provides a Domain Specific Language, *DSL*, to flexibly create user-defined strategies. Finally, Neo4j supports an API which advanced users may use to enhance upstream recommendations.

Once a strategy plugin resolves a satisfactory upstream server, it is used for pulling transactions to update the local Read Replica for a single synchronization. For subsequent updates, the procedure is repeated so that the most preferred available upstream server is always resolved.

Configuring upstream selection strategy using pre-defined strategies

Neo4j ships with the following pre-defined strategy plugins. These provide coarse-grained algorithms for choosing an upstream instance:

Plugin name	Resulting behavior
`connect-to-random-core-server`	Connect to any **Core Server** selecting at random from those currently available.
`typically-connect-to-random-read-replica`	Connect to any available **Read Replica**, but around 10% of the time connect to any random Core Server.
`connect-randomly-to-server-group`	Connect at random to any available **Read Replica** in any of the server groups specified in the comma-separated list `causal_clustering.connect-randomly-to-server-group`.
`leader-only`	Connect only to the current Raft leader of the Core Servers.
`connect-randomly-within-server-group`	Connect at random to any available **Read Replica** in any of the server groups to which this server belongs. Deprecated, please use `connect-randomly-to-server-group`.

Pre-defined strategies are used by configuring the `causal_clustering.upstream_selection_strategy` option. Doing so allows us to specify an ordered preference of strategies to resolve an upstream provider of transaction data. We provide a comma-separated list of strategy plugin names with preferred strategies earlier in that list. The upstream strategy is chosen by asking each of the strategies in list-order whether they can provide an upstream server from which transactions can be pulled.

Example 137. Define an upstream selection strategy

Consider the following configuration example:

```
causal_clustering.upstream_selection_strategy=connect-randomly-to-server-group,typically-connect-to-random-read-replica
```

With this configuration the instance will first try to connect to any other instance in the group(s) specified in `causal_clustering.connect-randomly-to-server-group`. Should we fail to find any live instances in those groups, then we will connect to a random Read Replica.

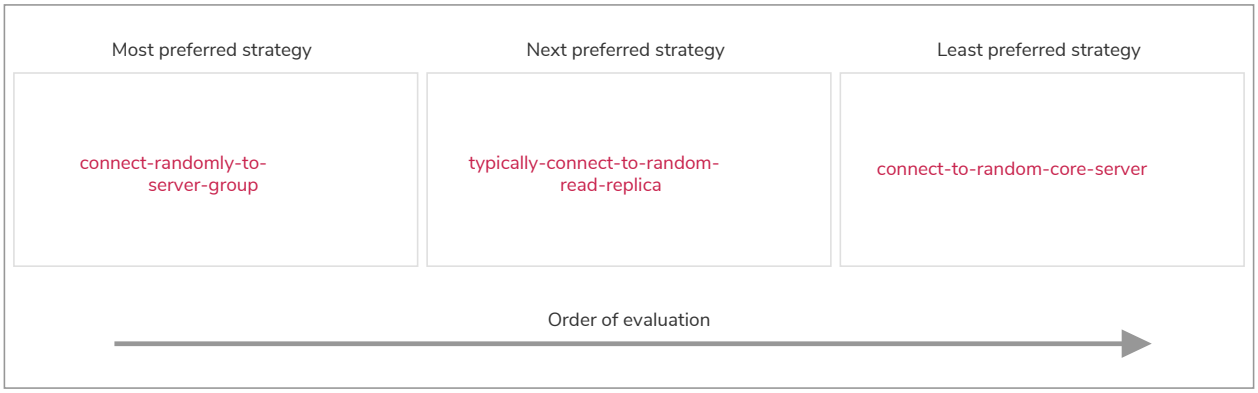


Figure 28. The first satisfactory response from a strategy will be used.

To ensure that downstream servers can still access live data in the event of upstream failures, the last resort of any instance is always to contact a random Core Server. This is equivalent to ending the `causal_clustering.upstream_selection_strategy` configuration with `connect-to-random-core-server`.

Configuring user-defined strategies

Neo4j Causal Clusters support a small DSL for the configuration of client-cluster load balancing. This is described in detail in Policy definitions and Filters. The same DSL is used to describe preferences for how an instance binds to another instance to request transaction updates.

The DSL is made available by selecting the `user-defined` strategy as follows:

```
causal_clustering.upstream_selection_strategy=user-defined
```

Once the user-defined strategy has been specified, we can add configuration to the `causal_clustering.user_defined_upstream_strategy` setting based on the server groups that have been set for the cluster.

We will describe this functionality with two examples:

Example 138. Defining a user-defined strategy

For illustrative purposes we propose four regions: `north`, `south`, `east`, and `west` and within each region we have a number of data centers such as `north1` or `west2`. We configure our server groups so that each data center maps to its own server group. Additionally we will assume that each data center fails independently from the others and that a region can act as a supergroup of its constituent data centers. So an instance in the `north` region might have configuration like `causal_clustering.server_groups=north2,north` which puts it in two groups that match to our physical topology as shown in the diagram below.

Figure 29. Mapping regions and data centers onto server groups

Once we have our server groups, our next task is to define some upstream selection rules based on them. For our design purposes, let's say that any instance in one of the `north` region data centers prefers to catchup within the data center if it can, but will resort to any northern instance otherwise. To configure that behavior we add:

```
causal_clustering.user_defined_upstream_strategy=groups(north2); groups(north); halt()
```

The configuration is in precedence order from left to right. The `groups()` operator yields a server group from which to catch up. In this case only if there are no servers in the `north2` server group will we proceed to the `groups(north)` rule which yields any server in the `north` server group. Finally, if we cannot resolve any servers in any of the previous groups, then we will stop the rule chain via `halt()`.

Note that the use of `halt()` will end the rule chain explicitly. If we don't use `halt()` at the end of the rule chain, then the `all()` rule is implicitly added. `all()` is expansive: it offers up all servers and so increases the likelihood of finding an available upstream server. However `all()` is indiscriminate and the servers it offers are not guaranteed to be topologically or geographically local, potentially increasing the latency of synchronization.

The example above shows a simple hierarchy of preferences. But we can be more sophisticated if we so choose. For example we can place conditions on the server groups from which we catch up.

Example 139. User-defined strategy with conditions

In this example we wish to roughly qualify cluster health before choosing from where to catch up. For this we use the `min()` filter as follows:

```
causal_clustering.user_defined_upstream_strategy=groups(north2)->min(3), groups(north)->min(3);
all();
```

`groups(north2)->min(3)` states that we want to catch up from the `north2` server group if it has three available machines, which we here take as an indicator of good health. If `north2` can't meet that requirement (is not healthy enough) then we try to catch up from any server across the `north` region provided there are at least three of them available as per `groups(north)->min(3)`. Finally, if we cannot catch up from a sufficiently healthy `north` region, then we'll (explicitly) fall back to the whole cluster with `all()`.

The `min()` filter is a simple but reasonable indicator of server group health.

Building upstream strategy plugins using Java

Neo4j supports an API which advanced users may use to enhance upstream recommendations in arbitrary ways: load, subnet, machine size, or anything else accessible from the JVM. In such cases we are invited to build our own implementations of `org.neo4j.causalclustering.readreplica.UpstreamDatabaseSelectionStrategy` to suit our own needs, and register them with the strategy selection pipeline just like the pre-packaged plugins.

We have to override the `org.neo4j.causalclustering.readreplica.UpstreamDatabaseSelectionStrategy#upstreamDatabase()` method in our code. Overriding that class gives us access to the following items:

Resource	Description
`org.neo4j.causalclustering.discovery.TopologyService`	This is a directory service which provides access to the addresses of all servers and server groups in the cluster.
`org.neo4j.kernel.configuration.Config`	This provides the configuration from *neo4j.conf* for the local instance. Configuration for our own plugin can reside here.
`org.neo4j.causalclustering.identity.MemberId`	This provides the unique cluster `MemberId` of the current instance.

Once our code is written and tested, we have to prepare it for deployment. `UpstreamDatabaseSelectionStrategy` plugins are loaded via the Java Service Loader. This means when we package our code into a jar file, we'll have to create a file *META-INF.services/org.neo4j.causalclustering.readreplica.UpstreamDatabaseSelectionStrategy* in which we write the fully qualified class name(s) of the plugins, e.g. `org.example.myplugins.PreferServersWithHighIOPS`.

To deploy this jar into the Neo4j server we copy it into the *plugins* directory and restart the instance.

Favoring data centers

In a multi-DC scenario, while it remains a rare occurrence, it is possible to bias where writes for the specified database should be directed. We can apply `causal_clustering.leadership_priority_group` to specify a group of servers which should have priority when selecting the leader for a given database. The priority group can be set on one or multiple databases and it means that the cluster will attempt to keep the leadership for the configured database on an instance tagged with the configured server group.

A database for which `leadership_priority_group` has been configured will be excluded from the automatic balancing of leaderships across a cluster. It is therefore recommended to not use this configuration unless it is necessary.

16.C.5. Multi-data center load balancing

This section describes the following:

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 Neo4j Driver manuals for instructions on how to configure drivers to use custom load balancing.

Prerequisite configuration

Enable multi-data center operations

In order to configure a cluster for load balancing we must enable the multi-data center functionality. This is described in Licensing for multi-data center operations.

Server groups

In common with server-to-server catchup, load balancing across multiple data centers is predicated on the *server group* concept. Servers can belong to one or more potentially overlapping server groups, and decisions about where to route requests from client to cluster member are parameterized based on that configuration. For details on server group configuration, refer to Server groups.

Cores for reading

Depending on the deployment and the available number of servers in the cluster different strategies make sense for whether or not the reading workload should be routed to the Core Servers. The following configuration will allow the routing of read workload to Core Servers. Valid values are `true` and `false`.

```
causal_clustering.cluster_allow_reads_on_followers=true
```

The load balancing framework

The load balancing system is based on a plugin architecture for future extensibility and for allowing user customizations. The current version ships with exactly one such canned plugin called the *server policies* plugin.

The server policies plugin is selected by setting the following property:

```
causal_clustering.load_balancing.plugin=server_policies
```

Under the server policies plugin, a number of load balancing policies can be configured server-side and be exposed to drivers under unique names. The drivers, in turn, must on instantiation select an appropriate policy by specifying its name. Common patterns for naming policies are after geographical regions or intended application groups.

It is of crucial importance to define the exact same policies on all core machines since this is to be regarded as cluster-wide configuration and failure to do so will lead to surprising behavior. Similarly, policies which are in active use should not be removed or renamed since it will break applications trying to use these policies. It is perfectly acceptable and expected however that policies be modified under the same name.

If a driver asks for a policy name which is not available, then it will not be able to use the cluster. A driver which does not specify any name at all will get the behavior of the default policy as configured. The default policy, if left unchanged, distributes the load across all servers. It is possible to change the default policy to any behavior that a named policy can have.

A misconfigured driver or load balancing policy will result in suboptimal routing choices or even prevent successful interactions with the cluster entirely.

The details of how to write a custom plugin is not documented here. Please get in contact with Neo4j Professional Services if you think that you need a custom plugin.

Policy definitions

The configuration of load balancing policies is transparent to client applications and expressed via a simple DSL. The syntax consists of a set of rules which are considered in order. The first rule to produce a non-empty result will be the final result.

```
rule1; rule2; rule3
```

Each rule in turn consists of a set of filters which limit the considered servers, starting with the complete set. Note that the evaluation of each rule starts fresh with the complete set of available servers.

There is a fixed set of filters which compose a rule and they are chained together using arrows

```
filter1 -> filter2 -> filter3
```

If there are any servers still left after the last filter then the rule evaluation has produced a result and this will be returned to the driver. However, if there are no servers left then the next rule will be considered. If no rule is able to produce a usable result then the driver will be signalled a failure.

Policy names

The policies are configured under the namespace of the *server policies* plugin and named as desired. Policy names can contain alphanumeric characters and underscores, and they are case sensitive. Below is the property key for a policy with the name `mypolicy`.

`causal_clustering.load_balancing.config.server_policies.mypolicy=`

The actual policy is defined in the value part using the DSL.

The `default` policy name is reserved for the default policy. It is possible to configure this policy like any other and it will be used by driver clients which do not specify a policy.

Additionally, any number of policies can be created using unique policy names. The policy name can suggest a particular region or an application for which it is intended to be used.

Filters

There are four filters available for specifying rules, detailed below. The syntax is similar to a method call with parameters.

- `groups(name1, name2, …)`
  - Only servers which are part of any of the specified groups will pass the filter.
  - The defined names must match those of the *server groups*.
- `min(count)`
  - Only the minimum amount of servers will be allowed to pass (or none).
  - Allows overload conditions to be managed.
- `all()`
  - No need to specify since it is implicit at the beginning of each rule.
  - Implicitly the last rule (override this behavior using halt).
- `halt()`
  - Only makes sense as the last filter in the last rule.
  - Will stop the processing of any more rules.

The groups filter is essentially an OR-filter, e.g. `groups(A,B)` which will pass any server in either A, B or both (the union of the server groups). An AND-filter can also be created by chaining two filters as in `groups(A) -> groups(B)`, which will only pass servers in both groups (the intersect of the server groups).

Load balancing examples

In our discussion on multi-data center clusters we introduced a four region, multi-data center setup. We used the cardinal compass points for regions and numbered data centers within those regions. We'll use the same hypothetical setup here too.

Figure 30. Mapping regions and data centers onto server groups

We configure the behavior of the load balancer in the property `causal_clustering.load_balancing.config.server_policies.<policy-name>`. The rules we specify will allow us to fine tune how the cluster routes requests under load.

In the examples we will make use of the line continuation character `\` for better readability. It is valid syntax in *neo4j.conf* as well and it is recommended to break up complicated rule definitions using this and a new rule on every line.

The most restrictive strategy would be to insist on a particular data center to the exclusion of all others:

Example 140. Specific data center only

```
causal_clustering.load_balancing.config.server_policies.north1_only=\
groups(north1)->min(2); halt();
```

In this case we're stating that we are only interested in sending queries to servers in the `north1` server group, which maps onto a specific physical data center, provided there are two of them available. If we cannot provide at least two servers in `north1` then we should `halt()`, i.e. not try any other data center.

While the previous example demonstrates the basic form of our load balancing rules, we can be a little more expansive:

Example 141. Specific data center preferably

```
causal_clustering.load_balancing.config.server_policies.north1=\
groups(north1)->min(2);
```

In this case if at least two servers are available in the `north1` data center then we will load balance across them. Otherwise we will use any server in the whole cluster, falling back to the implicit, final `all()` rule.

The previous example considered only a single data center before resorting to the whole cluster. If we have a hierarchy or region concept exposed through our server groups we can make the fall back more graceful:

Example 142. Gracefully falling back to neighbors

```
causal_clustering.load_balancing.config.server_policies.north_app1=\
groups(north1,north2)->min(2);\
groups(north);\
all();
```

In this case we're saying that the cluster should load balance across the `north1` and `north2` data centers provided there are at least two machines available across them. Failing that, we'll resort to any instance in the `north` region, and if the whole of the north is offline we'll resort to any instances in the cluster.

16.C.6. Data center disaster recovery

This section describes the following:

- Data center loss scenario
- Procedure for recovering from data center loss

Data center loss scenario

This section describes how to recover a multi-data center deployment which owing to external circumstances has reduced the cluster below half of its members. It is most easily typified by a 2x2 deployment with 2 data centers each containing two instances. This deployment topology can either arise because of other data center failures, or be a deliberate choice to ensure the geographic survival of data for catastrophe planning. However, by distributing an instance over three data centers instead, you could avoid having the cluster lose quorum through a single data center failure. For example, in a *1x1x1* deployment.

Under normal operation this provides a stable majority quorum where the fastest three out of four machines will execute users' transactions, as we see highlighted in Two Data Center Deployment with Four Core Instances", role="middle.

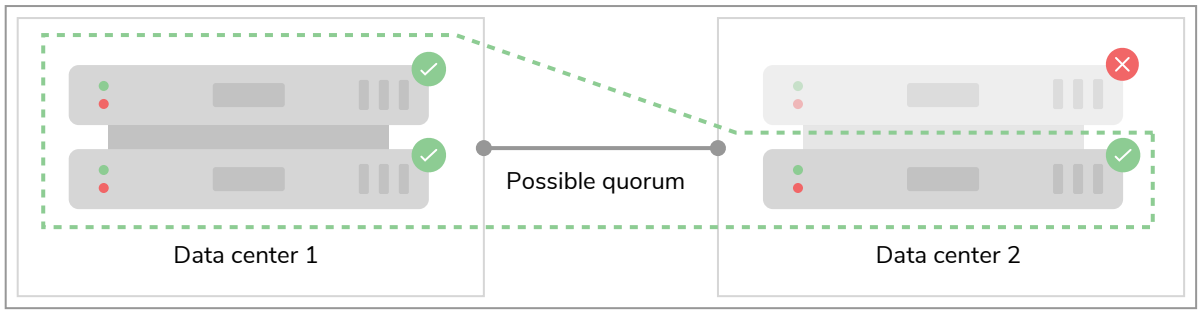


Figure 31. Two Data Center Deployment with Four Core Instances

However if an entire data center becomes offline because of some disaster, then a *majority quorum* cannot be formed in this case.

Neo4j Core clusters are based on the Raft consensus protocol for processing transactions. The Raft protocol requires a majority of cluster members to agree in order to ensure the safety of the cluster and data. As such, the loss of a majority quorum results in a read-only situation for the remaining cluster members.

When data center is lost abruptly in a disaster rather than having the instances cleanly shut down, the surviving members still believe that they are part of a larger cluster. This is different from even the case of rapid failures of individual instances in a live data center which can often be detected by the underlying cluster middleware, allowing the cluster to automatically reconfigure.

Conversely if we lose a data center, there is no opportunity for the cluster to automatically reconfigure. The loss appears instantaneous to other cluster members. However, because each remaining machine has only a partial view of the state of the cluster (its own), it is not safe to allow any individual machine to make an arbitrary decision to reform the cluster.

In this case we are left with two surviving machines which cannot form a quorum and thus make progress.

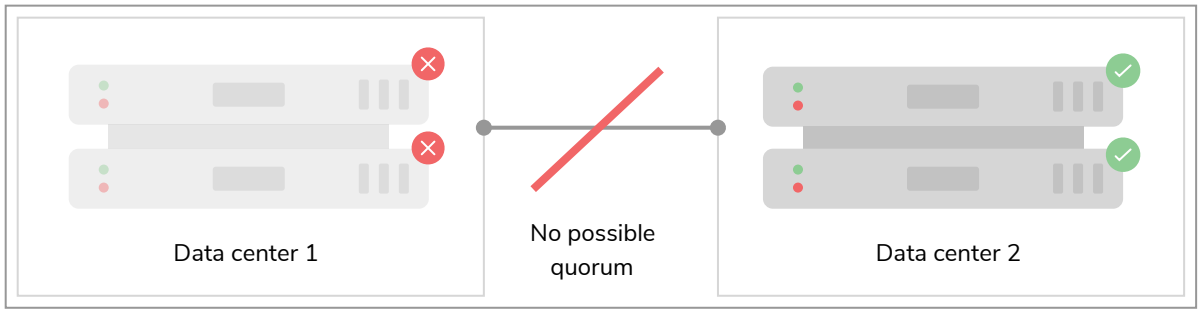


Figure 32. Data Center Loss Requires Guided Recovery

But, from a birds's eye view, it's clear we have surviving machines which are sufficient to allow a non-fault tolerant cluster to form under operator supervision.

Groups of individual cluster members (e.g. those in a single data center) may become isolated from the cluster during network partition for example. If they arbitrarily reformed a new, smaller cluster there is a risk of *split-brain*. That is from the clients' point of view there may be two or more smaller clusters that are available for reads and writes depending on the nature of the partition. Such situations lead to divergence that is tricky and laborious to reconcile and so best avoided.

To be safe, an operator or other out-of-band agent (e.g. scripts triggered by well-understood, trustworthy alerts) that has a trusted view on the whole of the system estate must make that decision. In the surviving data center, the cluster can be rebooted into a smaller configuration whilst retaining all data committed to that point. While end users may experience unavailability during the switch over, no committed data will be lost.

Procedure for recovering from data center loss

The following procedure for performing recovery of a data center should not be done lightly. It assumes that we are completely confident that a disaster has occurred and our previously data center-spanning cluster has been reduced to a read-only cluster in a single data center, where there is no possible way to repair a connection to the lost instances. Further it assumes that the remaining cluster members are fit to provide a seed from which a new cluster can be created from a data quality point of view.

Having acknowledged the above, the procedure for returning the cluster to full availability following catastrophic loss of all but one data centers can be done using one of the following options, depending on your infrastructure.

Please note that the main difference between the options is that Option 2 will allow read-availability during recovery.

Option 1.

If you are unable to add instances to the current data-center, and can only use the current read-only cluster, the following steps are recommended:

1. Verify that a catastrophe has occurred, and that access to the surviving members of the cluster in the surviving data center is possible. Then for each instance:
    a. Stop the instance with `bin/neo4j stop` or shut down the service.
    b. Change the configuration in *neo4j.conf* such that the `causal_clustering.initial_discovery_members` property contains the DNS names or IP addresses of the other surviving instances.
    c. Optional: you may need to update `causal_clustering.minimum_core_cluster_size_at_formation`, depending on the current size of the cluster (in the current example, two cores).
    d. Unbind the instance using `neo4j-admin unbind`.
    e. Start the instance with `bin/neo4j start` or start the `neo4j` service.

Option 2.

If it is possible to create a new cluster while the previous read-only cluster is still running, then the following steps will enable you to keep read-availability during recovery:

1. Verify that a catastrophe has occurred, and that access to the surviving members of the cluster in the surviving data center is possible.
2. Perform an online backup of the currently running, read-only, cluster.
3. Seed a new cluster (in the current example, two new cores) using the backup from the read-only cluster, as described in Seed a cluster.
4. When the new cluster is up, load balance your workload over to the new cluster.
5. Shutdown the old, read-only, cluster.

Once your chosen recovery procedure is completed for each instance, they will form a cluster that is available for reads and writes. It recommended at this point that other cluster members are incorporated into the cluster to improve its load handling and fault tolerance. See Deploy a cluster for details of how to configure instances to join the cluster from scratch.

16.10. Embedded usage

For users coming to Causal Clustering from Neo4j HA embedded, there are a small number of changes required. The Neo4j routing driver is used for routing and load balancing queries in server deployments (other setups are possible with 3rd party load balancers).

The driver also handles bookmarks, which are essential for causal consistency, and as such is a fundamental part of the Causal Clustering architecture. In an embedded deployment the driver can be used either for routing queries externally from another application into the embedded cluster, or using an embedded driver internally within the cluster.

The workload must be comprised, in its entirety, of Cypher statements. If your workload depends on the Java Core API for writing, then you have to package those pieces as procedures which are (remotely) invoked using Cypher, via the driver. Read-only queries can still access the Core API directly.

For a detailed tutorial on how to embed Neo4j in your Java application, see Neo4j Java Reference → Including Neo4j in your project.

Appendix D: Deprecated security procedures

This appendix describes deprecated procedures for security management:

- Enterprise Edition
- Community Edition

The procedures described in this appendix have been deprecated and will be removed in a future release.

It is strongly recommended to migrate to the security features as described in Cypher Manual → Administration → Security

See also a worked example in Fine-grained access control.

16.D.1. Enterprise Edition

A subset of this functionality is also available in Community Edition. The table below includes an indication of which functions this is valid for. Refer to Community Edition for a complete description.

In Neo4j, native user and role management are managed by using built-in procedures through Cypher. This section gives a list of all the security procedures for user management along with some simple examples. Use Neo4j Browser or Neo4j Cypher Shell to run the examples provided.

The following table lists the available procedures:

Procedure name	Description	Executable by role(s)	Available in Community Edition
`dbms.security.activateUser`	Activate a suspended user	[admin]	
`dbms.security.addRoleToUser`	Assign a role to the user	[admin]	
`dbms.security.changePassword`	Change the current user's password	[reader,editor,publisher,architect,admin]	✓
`dbms.security.changeUserPassword`	Change the given user's password	[admin]	
`dbms.security.createRole`	Create a new role	[admin]	
`dbms.security.createUser`	Create a new user	[admin]	✓
`dbms.security.deleteRole`	Delete the specified role. Any role assignments will be removed	[admin]	
`dbms.security.deleteUser`	Delete the specified user	[admin]	✓
`dbms.security.listRoles`	List all available roles	[admin]	
`dbms.security.listRolesForUser`	List all roles assigned to the specified user	[admin]	
`dbms.security.listUsers`	List all local users	[admin]	✓
`dbms.security.listUsersForRole`	List all users currently assigned the specified role	[admin]	
`dbms.security.removeRoleFromUser`	Unassign a role from the user	[admin]	
`dbms.security.suspendUser`	Suspend the specified user	[admin]	

Activate a suspended user

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

Syntax:

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

Arguments:

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

Exceptions:

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

Considerations:

This is an idempotent procedure.

Example 143. 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 144. Assign a role to the user

The following example assigns the role `publisher` to the user with username **'johnsmith'**.

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

Change the current user's password

The procedure `dbms.security.changePassword(newPassword, requirePasswordChange)` has been entirely removed since the corresponding Cypher administration command also requires the old password, and thus is more secure. Please use `ALTER CURRENT USER SET PASSWORD FROM 'oldPassword' TO 'newPassword'`, documented in the Cypher Manual, instead.

Change the given user's password

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

Syntax:

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

Arguments:

Name	Type	Description
`username`	String	This is the username of the user whose password is to be changed.
`newPassword`	String	This is the new password for the user.

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

Exceptions:

The current user is not an administrator and the username does not match that of the current user.
The username does not exist in the system.
The password is the empty string.
The password is the same as the user's previous password.

Considerations:

This procedure may be invoked by the current user to change their own password, irrespective of whether or not the current user is an administrator.
This procedure may be invoked by an administrator to change another user's password.
In addition to changing the user's password, this will terminate with immediate effect all of the user's sessions and roll back any running transactions.

Example 145. 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 146. 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 147. Create a new user

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

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

Delete the specified role

An administrator is able to delete roles from the system.

Syntax:

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

Arguments:

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

Exceptions:

The current user is not an administrator.
The role name does not exist in the system.
The role name matches one of the native roles: `reader`, `publisher`, `architect`, and `admin`.

Considerations:

Any role assignments will be removed.

Example 148. 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 149. 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 150. 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 151. List all roles assigned to the specified user

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

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

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

List all local users

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

Syntax:

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

Returns:

Name	Type	Description
`username`	String	This is the user's username.
`roles`	List<String>	This is a list of roles assigned to the user.
`flags`	List<String>	This is a series of flags indicating whether the user is suspended or needs to change their password.

Exceptions:

The current user is not an administrator.

Example 152. 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 153. 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 154. 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 155. Suspend the specified user

The following example suspends a user with the username '**billjones**'.

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

16.D.2. Community Edition

User and password management for Community Edition is a subset of the functionality available in Enterprise Edition. The following is true for user management in Community Edition:

- It is possible to create multiple users.
- All users assume the privileges of an `admin` for the available functionality.

Users are managed by using built-in procedures through Cypher. This section gives a list of all the security procedures for user management along with some simple examples. Use Neo4j Browser or Neo4j Cypher Shell to run the examples provided. Unless stated otherwise, all arguments to the procedures described in this section must be supplied.

Name	Description
dbms.security.changePassword	Change the current user's password
dbms.security.createUser	Add a user
dbms.security.deleteUser	Delete a user
dbms.security.listUsers	List all users

Change the current user's password

The procedure `dbms.security.changePassword(newPassword, requirePasswordChange)` has been entirely removed since the corresponding Cypher administration command also requires the old password, and thus is more secure. Please use `ALTER CURRENT USER SET PASSWORD FROM 'oldPassword' TO 'newPassword'`, documented in the Cypher Manual, instead.

Add a user

The current user is able to add a user to the system.

Syntax:

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

Arguments:

Name	Type	Description
`username`	String	This is the user's username.
`password`	String	This is the user's password.

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

Exceptions:

The username either contains characters other than the ASCII characters between `!` and `~`, or contains `:` and `,`.
The username is already in use within the system.
The password is the empty string.

Example 156. Add a user

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

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

Delete a user

The current user is able to delete permanently a user from the system.

Syntax:

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

Arguments:

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

Exceptions:

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

Considerations:

Deleting a user will terminate with immediate effect all of the user's sessions and roll back any running transactions.
As it is not possible for the current user to delete themselves, there will always be at least one user in the system.

Example 157. Delete a user

The following example deletes a user with the username '**janebrown**'.

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

List all native users

The current user is able to view the details of every user in the system.

Syntax:

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

Returns:

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

Example 158. List all users

The following example shows the username for each user in the system, and whether the user needs to change their password.

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

```
+-----------------------------------------+
| username | flags                        |
+-----------------------------------------+
| "neo4j"  | []                           |
| "anne"   | ["password_change_required"] |
| "bill"   | []                           |
+-----------------------------------------+
3 rows
```

License