The Neo4j Operations Manual v5

Table of Contents

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

For information on upgrading and migrating Neo4j, see Neo4j Upgrade and Migration Guide. **Neo4j AuraDB** is a fully managed Neo4j database, hosted in the cloud and requires no installation. For more information, see the AuraDB product page and AuraDB documentation.

The latest version of Neo4j is 5.4.

Documentation updates for Neo4j 5

- Restructured chapter on clustering.

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

- Composite databases.

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

- `neo4j-admin` refresh.

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

- A major overhaul of backup and restore.

  The new backup subsystem provides:

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

  The Backup subsystem from Neo4j 4.x is available as `neo4j-admin database legacy-backup` and `legacy-restore`.
  See Backup and restore section for detailed descriptions and instructions.

- Incremental offline import.

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

- Log4j integration completion.

  The logging framework is fully integrated with Log4j, providing more functionality and better control of all the database logs.
  Configuration settings are located in the `$NEO4J_HOME/conf` folder in the *user-logs.xml* used for

neo4j.log and *server-logs.xml* used for *debug.log*, *security.log*, *http.log*, and *query.log*. Query log uses the same format as *neo4j.log*.
Users are able to use/modify Log4j XML files.
See the section on the logging mechanisms in Neo4j for more details.

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

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

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

  Immutable privileges are useful for restricting the actions of users who themselves are able to administer privileges.
  Cloud operators can use sidecar design pattern to control RBAC-based permissions.
  You can find a tutorial on how to administer immutable privileges.
- Changes to Neo4j indexes
  - The B-tree index type has been removed.
  - New Range and Point index types are available now.
  - Neo4j 5.1 introduces an improved index provider, `text-2.0`, for enhanced performance. New Text indexes will use the `text-2.0` provider by default.
  - Full-text indexes can now index lists of strings.

    See Index configuration for more details.

License

Introduction

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

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

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

Community Edition

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

Enterprise Edition

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

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 backward-compatible manner.
- `PATCH` release - backward-compatible bug fixes.

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

Cypher

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

For more details, see the The Neo4j Cypher Manual → Introduction.

Interaction

The recommended way of programmatically interacting with the database is either through the official Neo4j Drivers, or through using the Neo4j Java API. Neo4j provides an ACID-compliant transactional backend for your applications.

The official Neo4j Drivers

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

- Neo4j Java Driver

  For Spring-powered applications there is also Spring Data Neo4j.

- Neo4j JavaScript Driver
- Neo4j Python Driver
- Neo4j .NET Driver
- Neo4j Go Driver

See the Neo4j Download Center - Drivers for more links.

Other tools

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

Neo4j feature details

Neo4j key features

Table 1. Key features

Feature	Community Edition	Enterprise Edition
Open source under GPLv3	✓	
Native Graph		

Feature	Community Edition	Enterprise Edition
Property graph model	✓	✓
Native graph processing & storage	✓	✓
Standard and Aligned store format (34 Billion Nodes & Relationships)	✓	✓
High_limit (1 Quadrillion Nodes & Relationships)		✓
ACID-compliant transactions	✓	✓
Cypher graph query language	✓	✓
Slotted Cypher runtime		✓
Pipelined Cypher runtime (faster)		✓
Listing and terminating running queries		✓
High-performance caching	✓	✓
Cost-based query optimizer	✓	✓
Clients and APIs		
Cypher Client	✓	✓
Neo4j Browser with syntax highlighting	✓	✓
Bolt Protocol	✓	✓
Language drivers for C#, Go, Java, JavaScript & Python [1]	✓	✓
High-performance native API	✓	✓
APOC 450+ Core Procedures and Functions	✓	✓
Support for Neo4j Graph Data Science Community Edition [1]	✓	✓
Support for Neo4j Graph Data Science Enterprise Edition [1]		✓
Indexes and constraints		
Fast writes via native label indexes	✓	✓
Composite indexes	✓	✓
Full-text node & relationship indexes	✓	✓
Property-existence constraints		✓
Node Key constraints		✓
Security		
Role-based access control		✓
Sub-graph access control		✓
LDAP and Active Directory integration		✓

Feature	Community Edition	Enterprise Edition
Kerberos security option		✓
Data management		
Offline import		✓
Offline incremental import		✓
Auto-reuse of space	✓	✓
Store copy		✓
Offline backup (dump)	✓	✓
Scale and availability		
Online back-up and restore		✓
Multiple databases (beyond the `system` and default databases)		✓
Autonomous clustering		✓
Sharded and federated Composite databases		✓
Monitoring and management		
Endpoints and metrics for monitoring via Prometheus		✓
Neo4j Operations Manager		✓

[1] Must be downloaded and installed separately.

Installation

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

The following topics are covered:

- System requirements — The system requirements for a production deployment of Neo4j.
- 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 page and AuraDB documentation.

System requirements

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

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

Supported platforms

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

Hardware requirements

In terms of hardware requirements, follow these guidelines:

Table 2. Hardware requirement guidelines.

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

Memory	More memory allows for larger graphs, but it needs to be configured properly to avoid disruptive garbage collection operations. • Consult Disks, RAM and other tips for more details.
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 3. Hardware requirement guidelines for personal use and software development.

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

For cloud environments:

Table 4. Hardware requirement guidelines for cloud environments.

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

For server-based, on-premise environments:

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

CPU	Intel/AMD x86-x64. ARM64.

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

Software requirements

For personal use and software development:

Table 6. Software requirements for personal use and software development.

Operating System	Supported JDK
Debian 11	OpenJDK 17, OracleJDK 17, and ZuluJDK 17
MacOS 11, 12	ZuluJDK 17
SuSE Enterprise Desktop 15	OpenJDK 17, Oracle JDK 17
Ubuntu Desktop 22.04+	OpenJDK 17, OracleJDK 17, and ZuluJDK 17
Windows 10, 11	OracleJDK 17, ZuluJDK 17

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

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

Operating System	Supported JDK
Amazon Linux 2022 AMI	Amazon Corretto 17, and OracleJDK 17
CentOS Stream 8, 9	OpenJDK 17, OracleJDK 17, and ZuluJDK 17
Debian 11	OpenJDK 17, OracleJDK 17, and ZuluJDK 17
Red Hat Enterprise Linux Server 8.4, 8.6	Red Hat OpenJDK 17, Oracle JDK 17, and ZuluJDK 17
Ubuntu Server 16.04, 18.04, 20.04, 22.04	OpenJDK 17, OracleJDK 17, and ZuluJDK 17
Windows Server 2016, 2019, 2022	OracleJDK 17, ZuluJDK 17

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.

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

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

Java

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

Table 8. Neo4j version and JVM requirements.

Neo4j Version	JVM compliancy
3.x	Java SE 8 Platform Specificaton
4.x	Java SE 11 Platform Specificaton
5.x	Java SE 17 Platform Specification

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

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 DBMS, including both Neo4j Server and Neo4j Desktop.

For more information on how to use Neo4j Browser and its capabilities, see the Neo4j Browser documentation.

The following web browsers are supported:

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

Internet Explorer web browser is *not* supported.

Neo4j Desktop

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

Neo4j Desktop is not suited for production environments.

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

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

Linux installation

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

This section describes the following:

Debian-based distributions (.deb)

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

Java prerequisites

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

Oracle JDK, Zulu JDK, or Corretto JDK

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

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

- Oracle JDK
- Zulu JDK
- Amazon Corretto JDK

OpenJDK 17 on Ubuntu 16.04

Add the official OpenJDK package repository to `apt`:

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

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

Dealing with multiple installed Java versions

It is important that you configure your default Java version to point to Java 17, or Neo4j 5.4.0 will be unable to start. Do so with the `update-java-alternatives` command.

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

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

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

- Identify your Java 17 version, in this case, it is `java-1.17.0-openjdk-amd64`. Then set it as the default with (replacing `<java17name>` with the appropriate name from above):

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

- Finally, confirm which version of Java is the default:

  ```
  java -version
  ```

Installation

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`.

  The following method for production or business-critical installations is recommended:

  ```
  wget -O - https://debian.neo4j.com/neotechnology.gpg.key | sudo apt-key add -
  echo 'deb https://debian.neo4j.com stable 5' | 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=1:5.4.0
```

To install Neo4j Enterprise Edition:

```
sudo apt-get install neo4j-enterprise=1:5.4.0
```

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

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

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

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

Non-interactive installation of Neo4j Enterprise Edition Enterprise edition

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

For Neo4j v5.3 and later

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

For Neo4j v5.2 and earlier

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

Offline installation

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

> It is important to note that using this method will mean that the offline machine will not receive the dependencies that are normally downloaded and installed automatically when using `apt` for installing Neo4j; Cypher Shell and Java (if not installed already):
>
> - The Cypher Shell package can be downloaded from Neo4j Download Center.
> - For information on supported versions of Java, see System requirements.

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

     ```
     curl -O https://dist.neo4j.org/deb/neo4j-enterprise_5.4.0_all.deb
     ```

     > To list all files that the Debian software package (`.deb` file) installs:
     >
     > ```
     > dpkg --contents neo4j_5.4.0_all.deb
     > ```

   - Neo4j Community Edition:

```
curl -O https://dist.neo4j.org/deb/neo4j_5.4.0_all.deb
```

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

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

File locations

File locations for all Neo4j packages are documented here.

Operation

Most Neo4j configuration goes into neo4j.conf.

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

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

Environment variable	Default value	Details
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.

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

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

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

Java prerequisites

Neo4j 5 requires the Java 17 runtime.

OpenJDK Java 17

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

Oracle Java 17

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

We provide an adapter for Oracle Java 17 which must be installed before Neo4j. The adapter contains no code, but will stop the package manager from installing OpenJDK 17 as a dependency despite an existing Oracle Java 17 installation.

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

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

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

Zulu JDK 17 or Corretto 17

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

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

- Zulu JDK
- Amazon Corretto JDK

Install on Red Hat, CentOS or Amazon Linux

Set up the repository

To use the repository for generally available versions of Neo4j, run 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/5
enabled=1
gpgcheck=1
EOF
```

Install Neo4j

- To install Neo4j Community Edition as `root`:

  ```
  yum install neo4j-5.4.0
  ```

- To install Neo4j Enterprise Edition as `root`:

  ```
  yum install neo4j-enterprise-5.4.0
  ```

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

Non-interactive installation of Neo4j Enterprise Edition 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-5.4.0
```

Install on SUSE

For SUSE-based distributions the steps are as follows:

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

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

2. Install Neo4j.

   - To install Neo4j Community Edition as `root`:

     ```
     zypper install neo4j-5.4.0
     ```

   - To install Neo4j Enterprise Edition as `root`:

```
zypper install neo4j-enterprise-5.4.0
```

Offline installation

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

> It is important to note that using this method will mean that the offline machine will not receive the dependencies that are normally downloaded and installed automatically when using `yum` for installing Neo4j; Neo4j Cypher Shell and Java.
>
> For information on supported versions of Java, see System requirements.

Downloading the RPM installers

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

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

     ```
     curl -O https://dist.neo4j.org/rpm/neo4j-enterprise-5.4.0-1.noarch.rpm
     ```

   - Neo4j Community Edition:

     ```
     curl -O https://dist.neo4j.org/rpm/neo4j-5.4.0-1.noarch.rpm
     ```

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

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

Performing an offline installation

Before installing Neo4j, you must manually install the required Java 17 packages. Then, Neo4j and Cypher Shell can be installed by running the following commands as `root` user:

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

Offline upgrade from 4.4.0 or later

Before you begin, you will need to have Java 17 pre-installed and set to the default Java version. If using

Oracle Java 17, the same dependency issues apply as with the Oracle Java prerequisites.

Due to strict dependencies between Neo4j and Cypher Shell, both packages must be upgraded simultaneously. Run the following on the offline machine as `root`, 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
```

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

Linux executable (.tar)

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

Install Neo4j from a tarball

1. If it is not already installed, get OpenJDK 17 or Oracle Java 17.
2. Download the latest Neo4j tarball from Neo4j Download Center and unpack it:

   ```
   tar zxf neo4j-enterprise-5.4.0-unix.tar.gz
   ```

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

   ```
   mv neo4j-enterprise-5.4.0 /opt/
   ln -s /opt/neo4j-enterprise-5.4.0/opt/neo4j
   ```

4. Create a `neo4j` user and group:

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

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

     ```
     chown -R neo4j:adm /opt/neo4j-enterprise-5.4.0
     ```

   - RedHat

```
chown -R neo4j /opt/neo4j-enterprise-5.4.0
```

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.

```
ulimit -n
```

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

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

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

Running Neo4j as a service

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

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

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

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

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

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

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

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

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

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

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

4. Define the following values:

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

5. Reload the `systemd` settings.

```
sudo systemctl daemon-reload
```

6. Reboot your machine.

Neo4j system service

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

Setting the number of open files.

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

Configuration

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

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.

macOS installation

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

Unix console application

1. If it is not already installed, get OpenJDK 17 or Oracle Java 17.
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>. For example, `tar -xf neo4j-community-5.4.0-unix.tar.gz`.
5. Place the extracted files in a permanent home on your server. The top level directory is referred to as NEO4J_HOME.
   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`.
6. Visit http://localhost:7474 in your web browser.
7. Connect using the username 'neo4j' with default password 'neo4j'. You'll then be prompted to change

the password.

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

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

macOS service

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

macOS file descriptor limits

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

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

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

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

Windows installation

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

Windows console application

1. If it is not already installed, get OpenJDK 17 or Oracle Java 17.
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. Place the extracted files in a permanent home on your server, for example `D:\neo4j\`. The top level

directory is referred to as NEO4J_HOME.

   a. To run Neo4j as a console application, use: `<NEO4J_HOME>\bin\neo4j console`.

   b. To install Neo4j as a service use: `<NEO4J_HOME>\bin\neo4j windows-service install`.

   c. For additional commands and to learn about the Windows PowerShell module included in the Zip file, see Windows PowerShell module.

6. Visit http://localhost:7474 in your web browser.

7. Connect using the username 'neo4j' with default password 'neo4j'. You'll then be prompted to change the password.

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

Windows service

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

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

When installing a new release of Neo4j, you must first run `bin\neo4j windows-service uninstall` 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 `server.memory.heap.initial_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. To update the service, run `bin\neo4j windows-service update`.

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

Example 1. Update service example

1. Install service

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

2. Change memory configuration

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

3. Update service

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

4. Restart service

```
bin\neo4j restart
```

Windows PowerShell module

The Neo4j PowerShell module allows administrators to:

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

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

System requirements

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

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:\Neo4*j* 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                        5.4.0      Neo4j-Management
Function        Invoke-Neo4jAdmin                   5.4.0      Neo4j-Management
Function        Invoke-Neo4jBackup                  5.4.0      Neo4j-Management
Function        Invoke-Neo4jImport                  5.4.0      Neo4j-Management
Function        Invoke-Neo4jShell                   5.4.0      Neo4j-Management
```

The module also supports the standard PowerShell help commands.

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

Run the following to see examples of help commands:

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

Example usage

- List of available commands:

  ```
  Invoke-Neo4j
  ```

- Current status of the Neo4j service:

```
Invoke-Neo4j status
```

- Install the service with verbose output:

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

- Available commands for administrative tasks:

```
Invoke-Neo4jAdmin
```

Common PowerShell parameters

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

Cloud deployments

There are different options for deploying Neo4j in the cloud. Described here are:

- Neo4j on AWS — Deploying Neo4j on AWS.
- 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 on AWS

You can deploy Neo4j Enterprise Edition on Elastic Compute Cloud (EC2) instances in AWS directly from the AWS Marketplace.

The listing uses the Neo4j CloudFormation template (hosted in a public GitHub repository), which can be customized to meet more complex use cases.

CloudFormation templates for Neo4j v4.4 are also supported, as well as some custom templates that serve other commonly requested use cases.

Note that Neo4j does not provide AMIs with a pre-installed version of the product. The Neo4j Marketplace listings and custom listings on Github leverage CloudFormation templates and user-data scripting to deploy Neo4j.

For more information, see the following links:

- All CloudFormation templates for Neo4j v5.
- All CloudFormation templates for Neo4j v4.4.

Neo4j CloudFormation template

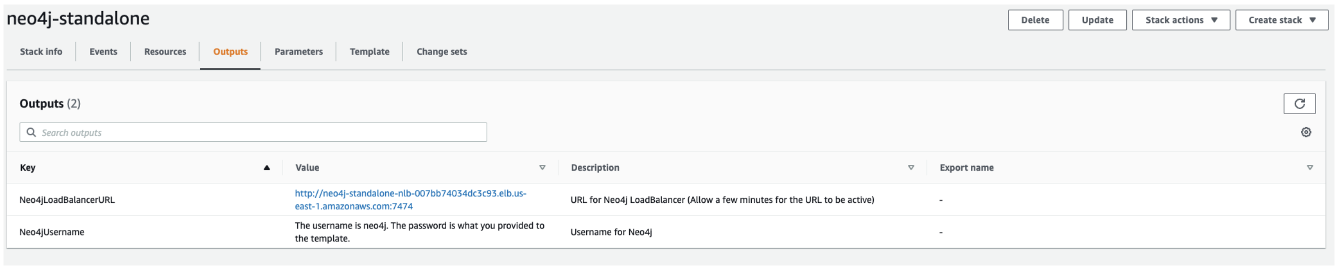
CloudFormation is an Infrastructure as Code (IaC) service that tells AWS how to deploy a set of interrelated resources.

The Neo4j CloudFormation templates have the following properties:

- Optionally install Graph Data Science (GDS). It requires a license key to be provided.
- Optionally install Bloom. It requires a license key to be provided.
- Specify the number of desired Neo4j instances.
- Select the EC2 instance type and disk size.
- Specify SSH CIDR range.

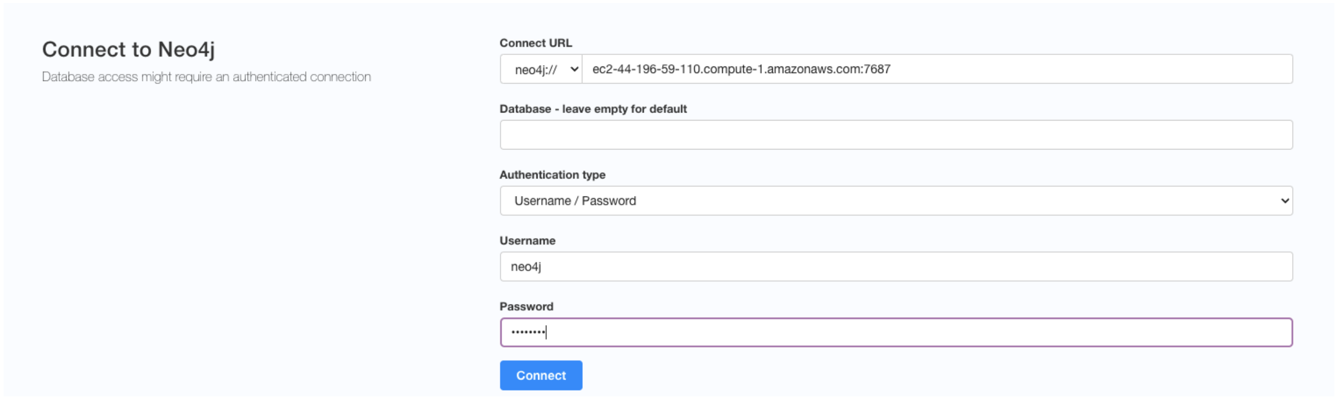
Verify that Neo4j is running

When the CloudFormation stack is created, navigate to the **Outputs** tab and click the **Neo4jLoadBalancerURL**.



Key	Value	Description	Export name
Neo4jLoadBalancerURL	http://neo4j-standalone-nlb-007bb74034dc3c93.elb.us-east-1.amazonaws.com:7474	URL for Neo4j LoadBalancer (Allow a few minutes for the URL to be active)	-
Neo4jUsername	The username is neo4j. The password is what you provided to the template.	Username for Neo4j	-

The URL points to the Neo4j Browser, where the specified credentials can be used to log in to Neo4j.



Connect URL	neo4j:// ec2-44-196-59-110.compute-1.amazonaws.com:7687
Database - leave empty for default	
Authentication type	Username / Password
Username	neo4j
Password	••••••••

Clean up the resources and remove your stack

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

Neo4j on 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.
- Clusters (VM-Based) — Deploying Neo4j on GCP.
- Neo4j deployments automation on GCP – Automating Neo4j deployments on GCP.

Neo4j standalone instance (VM-based)

You can deploy a Neo4j standalone instance using an image from the GCP Marketplace.

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
```

Delete your instance

You can run the following command to delete your instance:

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

Clusters (VM-based)

You can deploy a Neo4j cluster via the GCP Marketplace.

Prerequisites

- You have a Neo4j Enterprise license.
- You are familiar with the 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 deployment 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
```

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.

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 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 (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=5.4.0
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 $?
```

Neo4j on Azure

You can deploy a Neo4j standalone server or a cluster on Azure directly from the Azure Marketplace or by using the Neo4j Azure Resource Manager (ARM) template hosted on GitHub.

The Azure Marketplace represents a straightforward method to deploy Neo4j on a VM instance in Azure.

Both Neo4j Enterprise Edition v5 and v4.4 can be deployed from the Neo4j Enterprise Edition listing on the Azure Marketplace.

In addition to the Azure Marketplace listing, Neo4j provides an ARM template, which can be customized to meet more complex use cases. This template is hosted in a public GitHub repository and can be found at Neo4j ARM template on GitHub.

Neo4j ARM template

ARM is an Infrastructure as Code (IaC) service that tells Azure how to deploy a set of interrelated resources.

The Neo4j ARM template has the following properties:

- **Virtual Machine Size:** Select the VM size for the machine(s) that will host your deployment.
- **Node count:** Specify the number of desired Neo4j servers depending on whether you want to deploy a standalone or a cluster.
- **Disk Size:** The disk size for a Neo4j server.
- Optionally install Graph Data Science (GDS). It requires a license key to be provided.
- Optionally install Bloom. It requires a license key to be provided.

Deploy Neo4j from the Azure Marketplace

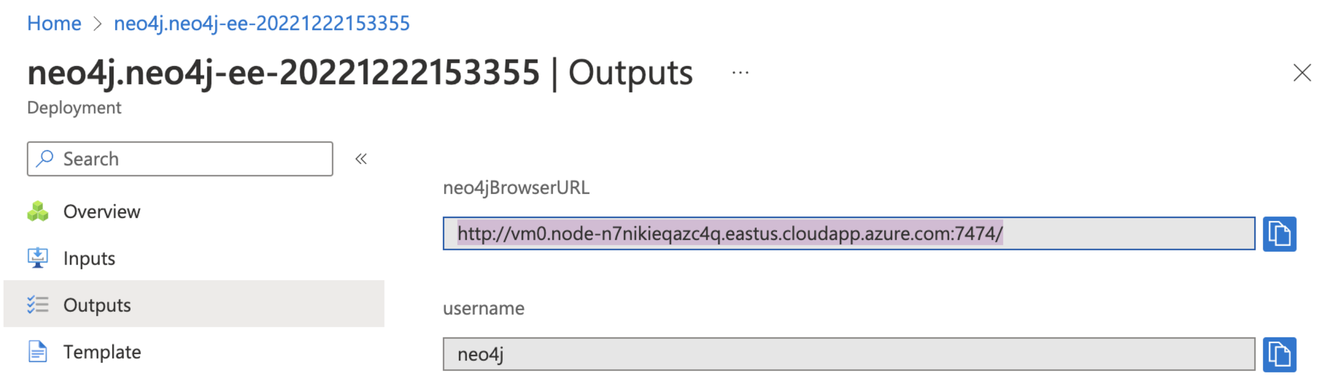
Deploy a Neo4j Enterprise cluster from the Azure Marketplace following the interactive prompts.

It is recommended to create a new resource group to hold the artifacts of your deployment.

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.

Verify that Neo4j is running

When the Neo4j deployment completes, navigate to the **Outputs** tab and copy the Neo4j Browser URL.


Home > neo4j.neo4j-ee-20221222153355

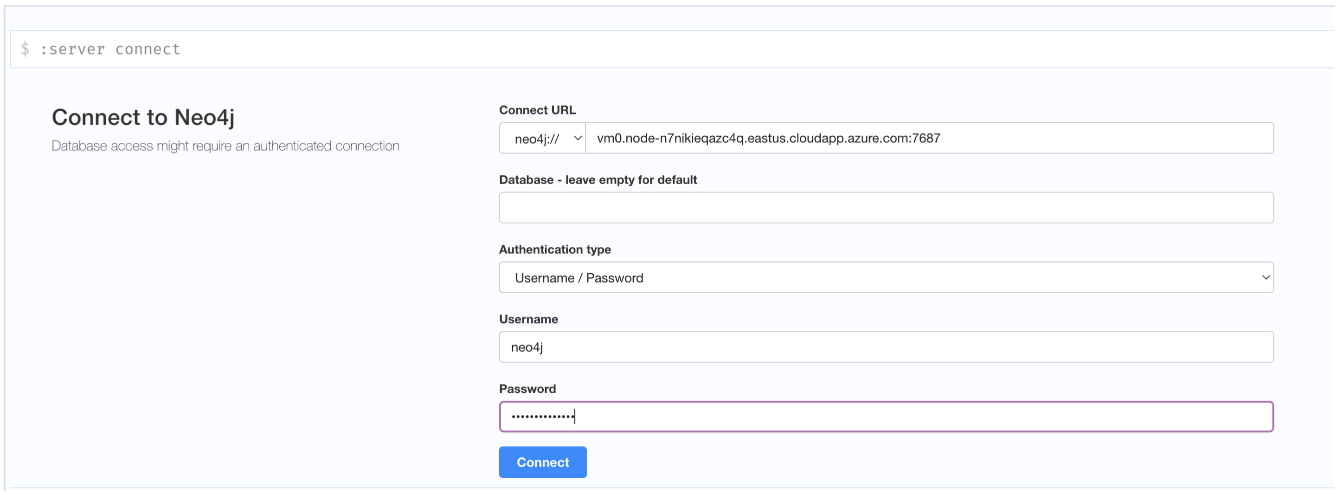
neo4j.neo4j-ee-20221222153355 | Outputs

Deployment

neo4jBrowserURL: http://vm0.node-n7nikieqazc4q.eastus.cloudapp.azure.com:7474/

username: neo4j

Then, in a web browser, paste the URL to open Neo4j Browser, where you can use the credentials that you specified to log into Neo4j.


```
$ :server connect
```

Connect to Neo4j

Database access might require an authenticated connection

Connect URL: neo4j:// vm0.node-n7nikieqazc4q.eastus.cloudapp.azure.com:7687

Database - leave empty for default:

Authentication type: Username / Password

Username: neo4j

Password: ••••••••••••••

Clean up the resources and remove your deployment

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

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

Docker

Neo4j can be run in a Docker container.

This chapter describes the following:

- Introduction — Introduction to running Neo4j in a Docker container.
- Configuration — How to configure Neo4j to run in a Docker container.
- Deploy a Neo4j cluster on Docker — How to set up and deploy a Neo4j cluster on Docker.
- Docker specific operations - Descriptions of various operations that are specific to using Docker.
- Security - Information about using encryption with a Neo4j 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.

Introduction

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

Neo4j editions

Tags are available for both Community Edition and Enterprise Edition. Version-specific Enterprise Edition tags have an `-enterprise` suffix, for example: `neo4j:5.4.0-enterprise`. Community Edition tags have no suffix, for example `neo4j:5.4.0`. The latest Neo4j Enterprise Edition release is available as `neo4j:enterprise`.

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

Neo4j Enterprise Edition license

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

Using the Neo4j Docker image

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

```
docker run \
    --restart always \
    --publish=7474:7474 --publish=7687:7687 \
    neo4j:5.4.0
```

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

The default minimum password length is 8 characters. Use the `dbms.security.auth_minimum_password_length` configuration to change it.

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

Using `NEO4J_AUTH` to set an initial password

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

```
docker run \
    --restart always \
    --publish=7474:7474 --publish=7687:7687 \
    --env NEO4J_AUTH=neo4j/your_password \
    neo4j:5.4.0
```

With no persistent storage for the databases, `NEO4J_ATUH` takes effect each time the container is recreated, even if you have changed the password.

However, with persistent storage for the databases, `NEO4J_AUTH` only takes effect on the initial startup. It is ignored when the container is recreated and cannot override a changed 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`.

Persisting data using Volumes

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

```
docker run \
    --restart always \
    --publish=7474:7474 --publish=7687:7687 \
    --env NEO4J_AUTH=neo4j/your_password \
    --volume=/path/to/your/data:/data \
    --volume=/path/to/your/logs:/logs \
    neo4j:5.4.0
```

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

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

Running Neo4j as a non-root user

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

```
docker run \
    --publish=7474:7474 --publish=7687:7687 \
    --user="$(id -u):$(id -g)" \
    neo4j:5.4.0
```

More useful Docker Run options

This table lists some of the options available:

Table 9. 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`

Option	Description	Example
`-d`	Detach container to run in background	`docker run -d neo4j`
`-v`	Bind mount a volume	`docker run -v $HOME/neo4j/data:/data neo4j`
`--env`	Set config as environment variables for the Neo4j database	`docker run --env NEO4J_AUTH=neo4j/your_password`
`--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.

Offline installation of Neo4j Docker image

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

This is an example command to save the `neo4j:5.4.0` image to a `.tar` file:

```
docker save -o neo4j-5.4.0.tar neo4j:5.4.0
```

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

```
docker load --input neo4j-5.4.0.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.

Configuration

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

There are three ways to modify the configuration:

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

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

Environment variables

Pass environment variables to the container when you run it.

```
docker run \
    --detach \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j/data:/data \
    --volume=$HOME/neo4j/logs:/logs \
    --env NEO4J_dbms_memory_pagecache_size=4G \
    neo4j:5.4.0
```

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

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

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

```
--env NEO4J_db_tx__log_rotation_size
```

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

```
--env NEO4J_server_jvm_additional="-Dcom.sun.management.jmxremote.authenticate=true
-Dcom.sun.management.jmxremote.ssl=false -Dcom.sun.management.jmxremote.password.file=
$HOME/conf/jmx.password -Dcom.sun.management.jmxremote.access.file=$HOME/conf/jmx.access
-Dcom.sun.management.jmxremote.port=3637"
```

Mounting the */conf* volume

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

```
docker run \
    --detach \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j/data:/data \
    --volume=$HOME/neo4j/logs:/logs \
    --volume=$HOME/neo4j/conf:/conf \
    neo4j:5.4.0
```

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

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

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

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

Processes in the Neo4j docker container run under the `neo4j` user by default.

```
docker run --rm \
    --volume=$HOME/neo4j/conf:/conf \
    neo4j:5.4.0 dump-config
```

Customize a Neo4j Docker image

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

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

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

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

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

/example/Dockerfile

FROM neo4j:5.4.0-enterprise
COPY extension_script.sh /extension_script.sh
ENV EXTENSION_SCRIPT=/extension_script.sh

/example/extension_script.sh

echo "extension logic"

# Build the custom image:

docker build --file /example/Dockerfile --tag neo4j:5.4.0-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:5.4.0-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.

Deploy a Neo4j cluster on Docker

Neo4j supports clustering in a containerized environment without an orchestration tool.

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

- Command expansion
- Discovery using DNS with multiple records

Deploy a Neo4j cluster using Docker Compose

You can deploy a Neo4j cluster using Docker Compose. Docker Compose is a management tool for Docker containers. You use a YAML file to define the infrastructure of all your cluster servers in one file. Then, by running the single command `docker-compose up`, you create and start all the members without the need to invoke each of them individually. For more information about Docker Compose, see the Docker Compose official documentation.

Prerequisites

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

Procedure

1. Create a configuration file `neo4j.conf` which will be shared across cluster members and make it readable and writable for the user (eg., `chmod 640 neo4j.conf`)

```
# Setting that specifies how much memory Neo4j is allowed to use for the page cache.
server.memory.pagecache.size=100M

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

# The behavior of the initial discovery is determined by the parameters
`dbms.cluster.discovery.type` and `dbms.cluster.discovery.endpoints`.
# The DNS strategy fetches the IP addresses of the cluster members using the DNS A records.
dbms.cluster.discovery.type=DNS

# The value of `dbms.cluster.discovery.endpoints` should be set to a single domain name and the
port of the discovery service.
# The domain name returns an A record for every server in the cluster when a DNS lookup is
performed.
# Each A record returned by DNS should contain the IP address of the server in the cluster.
# The configured server uses all the IP addresses from the A records to join or form a cluster.
# The discovery port must be the same on all servers when using this configuration.
dbms.cluster.discovery.endpoints=neo4j-network:5000

# Address (the public hostname/IP address of the machine)
# and port setting that specifies where this instance advertises for discovery protocol messages
from other members of the cluster.
server.discovery.advertised_address=$(hostname -i)

# Address (the public hostname/IP address of the machine)
# and port setting that specifies where this instance advertises for Raft messages within the
cluster.
server.cluster.raft.advertised_address=$(hostname)

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

# Enable server-side routing
dbms.routing.enabled=true

# Use server-side routing for neo4j:// protocol connections.
dbms.routing.default_router=SERVER

# The advertised address for the intra-cluster routing connector.
server.routing.advertised_address=$(hostname)
```

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

Example 2. Example docker-compose.yml file

```
version: '3.8'

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

services:

  server1:
    # Docker image to be used
    image: ${NEO4J_DOCKER_IMAGE}

    # Hostname
    hostname: server1

    # Service-level network, which specifies the networks, from the list of the top-level
networks (in this case only neo4j-internal), that the server will connect to.
    # Adds a network alias (used in neo4j.conf when configuring the discovery members)
    networks:
      neo4j-internal:
        aliases:
          - neo4j-network

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

    # Uncomment the volumes to be mounted to make them accessible from outside the container.
    volumes:
      - ./neo4j.conf:/conf/neo4j.conf # This is the main configuration file.
      - ./data/server1:/var/lib/neo4j/data
      - ./logs/server1:/var/lib/neo4j/logs
      - ./conf/server1:/var/lib/neo4j/conf
      - ./import/server1:/var/lib/neo4j/import
      #- ./metrics/server1:/var/lib/neo4j/metrics
      #- ./licenses/server1:/var/lib/neo4j/licenses
      #- ./ssl/server1:/var/lib/neo4j/ssl

    # Passes the following environment variables to the container
    environment:
      - NEO4J_ACCEPT_LICENSE_AGREEMENT
      - NEO4J_AUTH
      - EXTENDED_CONF
      - NEO4J_EDITION
      - NEO4J_initial_server_mode__constraint=PRIMARY

    # Simple check testing whether the port 7474 is opened.
    # If so, the instance running inside the container is considered as "healthy".
    # This status can be checked using the "docker ps" command.
    healthcheck:
      test: ["CMD-SHELL", "wget --no-verbose --tries=1 --spider localhost:7474 || exit 1"]

    # Set up the user
    user: ${USER_ID}:${GROUP_ID}

  server2:
    image: ${NEO4J_DOCKER_IMAGE}
    hostname: server2
    networks:
      neo4j-internal:
        aliases:
          - neo4j-network
    ports:
      - "7475:7474"
      - "7688:7687"
    volumes:
      - ./neo4j.conf:/conf/neo4j.conf
      - ./data/server2:/var/lib/neo4j/data
      - ./logs/server2:/var/lib/neo4j/logs
      - ./conf/server2:/var/lib/neo4j/conf
      - ./import/server2:/var/lib/neo4j/import
      #- ./metrics/server2:/var/lib/neo4j/metrics
      #- ./licenses/server2:/var/lib/neo4j/licenses
      #- ./ssl/server2:/var/lib/neo4j/ssl
    environment:
      - NEO4J_ACCEPT_LICENSE_AGREEMENT
      - NEO4J_AUTH
      - EXTENDED_CONF
      - NEO4J_EDITION
      - NEO4J_initial_server_mode__constraint=PRIMARY
    healthcheck:
      test: ["CMD-SHELL", "wget --no-verbose --tries=1 --spider localhost:7474 || exit 1"]
    user: ${USER_ID}:${GROUP_ID}

  server3:
    image: ${NEO4J_DOCKER_IMAGE}
    hostname: server3
    networks:
      neo4j-internal:
        aliases:
          - neo4j-network
    ports:
      - "7476:7474"
      - "7689:7687"
    volumes:
      - ./neo4j.conf:/conf/neo4j.conf
      - ./data/server3:/var/lib/neo4j/data
      - ./logs/server3:/var/lib/neo4j/logs
      - ./conf/server3:/var/lib/neo4j/conf
      - ./import/server3:/var/lib/neo4j/import
      #- ./metrics/server3:/var/lib/neo4j/metrics
      #- ./licenses/server3:/var/lib/neo4j/licenses
      #- ./ssl/server3:/var/lib/neo4j/ssl
    environment:
      - NEO4J_ACCEPT_LICENSE_AGREEMENT
      - NEO4J_AUTH
      - EXTENDED_CONF
      - NEO4J_EDITION
      - NEO4J_initial_server_mode__constraint=PRIMARY
    healthcheck:
      test: ["CMD-SHELL", "wget --no-verbose --tries=1 --spider localhost:7474 || exit 1"]
    user: ${USER_ID}:${GROUP_ID}

  server4:
    image: ${NEO4J_DOCKER_IMAGE}
    hostname: server4
    networks:
      neo4j-internal:
        aliases:
          - neo4j-network
    ports:
      - "7477:7474"
      - "7690:7687"
    volumes:
      - ./neo4j.conf:/conf/neo4j.conf
      - ./data/server4:/var/lib/neo4j/data
      - ./logs/server4:/var/lib/neo4j/logs
      - ./conf/server4:/var/lib/neo4j/conf
      - ./import/server4:/var/lib/neo4j/import
      #- ./metrics/server4:/var/lib/neo4j/metrics
      #- ./licenses/server4:/var/lib/neo4j/licenses
      #- ./ssl/server4:/var/lib/neo4j/ssl
    environment:
      - NEO4J_ACCEPT_LICENSE_AGREEMENT
      - NEO4J_AUTH
      - EXTENDED_CONF
      - NEO4J_EDITION
      - NEO4J_initial_server_mode__constraint=SECONDARY
    healthcheck:
      test: ["CMD-SHELL", "wget --no-verbose --tries=1 --spider localhost:7474 || exit 1"]
    user: ${USER_ID}:${GROUP_ID}
```

3. Set up the environment variables:

   ◦ `export USER_ID="$(id -u)"`

    ◦ `export GROUP_ID="$(id -g)"`
    ◦ `export NEO4J_DOCKER_IMAGE=neo4j:5-enterprise`
    ◦ `export NEO4J_EDITION=docker_compose`
    ◦ `export EXTENDED_CONF=yes`
    ◦ `export NEO4J_ACCEPT_LICENSE_AGREEMENT=yes`
    ◦ `export NEO4J_AUTH=neo4j/your_password`

4. Deploy your Neo4j cluster by running `docker-compose up` from your project folder.
5. The instance will be available at the following addresses:
    ◦ Neo4j instance *server1* will be available at http://localhost:7474.
    ◦ Neo4j instance *server2* will be available at http://localhost:7475.
    ◦ Neo4j instance *server3* will be available at http://localhost:7476.
    ◦ Neo4j instance *server4* will be available at http://localhost:7477.
6. Authenticate with the default `neo4j/your_password` credentials.
7. Check the status of the cluster by running the following in Neo4j Browser:

```
SHOW SERVERS
```

Example output:

```
╒══════════════════════════════════════╤═════════════════╤═════════╤═══════════╤═════════════════╕
│"name"                                │"address"        │"state"  │"health"   │"hosting"        │
╞══════════════════════════════════════╪═════════════════╪═════════╪═══════════╪═════════════════╡
│"26c4ec22-958c-468d-9735-d74c43165cf4"│"localhost:7687" │"Enabled"│"Available"│["system","neo4j"]│
├──────────────────────────────────────┼─────────────────┼─────────┼───────────┼─────────────────┤
│"4cd6a447-8a16-4ec6-9714-a5b8970d8a80"│"localhost:7687" │"Enabled"│"Available"│["system"]       │
├──────────────────────────────────────┼─────────────────┼─────────┼───────────┼─────────────────┤
│"5d1971e9-a3a9-4945-93f4-34457d918708"│"localhost:7687" │"Enabled"│"Available"│["system"]       │
├──────────────────────────────────────┼─────────────────┼─────────┼───────────┼─────────────────┤
│"8d4c4941-241e-4fac-97b5-c9a9adbc50c3"│"localhost:7687" │"Enabled"│"Available"│["system"]       │
└──────────────────────────────────────┴─────────────────┴─────────┴───────────┴─────────────────┘
```

Deploy a Neo4j Cluster using environment variables

You can set up containers in a cluster to talk to each other using environment variables. Each container must have a network route to each of the others, the `NEO4J_initial_dbms_default__primaries__count`, `NEO4J_initial_dbms_default__secondaries__count`, and `NEO4J_dbms_cluster_discovery_endpoints` environment variables must be set for all servers.

Cluster environment variables Enterprise edition

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

- `NEO4J_initial_server_mode__constraint`: the database mode, defaults to `NONE`, can be set to `PRIMARY` or `SECONDARY`.

- NEO4J_dbms_cluster_discovery_endpoints: a comma-separated list of endpoints, which a server should contact to discover other cluster servers.
- NEO4J_server_discovery_advertised_address: hostname/IP address and port to advertise for member discovery management communication.
- NEO4J_server.cluster.advertised_address: hostname/IP address and port to advertise for transaction handling.
- NEO4J_server.cluster.raft.advertised_address: hostname/IP address and port to advertise for cluster communication.

See Settings reference for more details of Neo4j cluster settings.

Set up a Neo4j Cluster on a single Docker host

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

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

```
docker network create --driver=bridge neo4j-cluster

docker run --name=server1 --detach --network=neo4j-cluster \
    --publish=7474:7474 --publish=7473:7473 --publish=7687:7687 \
    --hostname=server1 \
    --env NEO4J_initial_server_mode__constraint=PRIMARY \
    --env NEO4J_dbms_cluster_discovery_endpoints=server1:5000,server2:5000,server3:5000 \
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
    --env NEO4j_server_bolt_advertised_address=localhost:7687 \
    --env NEO4j_server_http_advertised_address=localhost:7474 \
    --env NEO4J_AUTH=neo4j/mypassword \
    neo4j:5.4.0-enterprise

docker run --name=server2 --detach --network=neo4j-cluster \
    --publish=8474:7474 --publish=8473:7473 --publish=8687:7687 \
    --hostname=server2 \
    --env NEO4J_initial_server_mode__constraint=PRIMARY \
    --env NEO4J_dbms_cluster_discovery_endpoints=server1:5000,server2:5000,server3:5000 \
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
    --env NEO4j_server_bolt_advertised_address=localhost:8687 \
    --env NEO4j_server_http_advertised_address=localhost:8474 \
    --env NEO4J_AUTH=neo4j/mypassword \
    neo4j:5.4.0-enterprise

docker run --name=server3 --detach --network=neo4j-cluster \
    --publish=9474:7474 --publish=9473:7473 --publish=9687:7687 \
    --hostname=server3 \
    --env NEO4J_initial_server_mode__constraint=PRIMARY \
    --env NEO4J_dbms_cluster_discovery_endpoints=server1:5000,server2:5000,server3:5000 \
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
    --env NEO4j_server_bolt_advertised_address=localhost:9687 \
    --env NEO4j_server_http_advertised_address=localhost:9474 \
    --env NEO4J_AUTH=neo4j/mypassword \
    neo4j:5.4.0-enterprise
```

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

Example of a `docker run` command for adding a fourth server with a role `SECONDARY` to the cluster:

```
docker run --name=read-server4 --detach --network=neo4j-cluster \
    --publish=10474:7474 --publish=10473:7473 --publish=10687:7687 \
    --hostname=read-server4 \
    --env NEO4J_initial_server_mode__constraint=SECONDARY \
    --env NEO4J_dbms_cluster_discovery_endpoints=server1:5000,server2:5000,server3:5000 \
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
    --env NEO4j_server_bolt_advertised_address=localhost:10687 \
    --env NEO4j_server_http_advertised_address=localhost:10474 \
    neo4j:5.4.0-enterprise
```

Set up a Neo4j Cluster on multiple Docker hosts

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

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

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

```
docker run --name=server1 --detach \
         --network=host \
         --publish=7474:7474 --publish=7687:7687 \
         --publish=5000:5000 --publish=6000:6000 --publish=7000:7000 \
         --hostname=public-address \
         --env NEO4J_dbms_cluster_discovery_endpoints=server1-public-address:5000,server2-public-
address:5000,server3-public-address:5000 \
         --env NEO4J_server_discovery_advertised_address=public-address:5000 \
         --env NEO4J_server_cluster_advertised_address=public-address:6000 \
         --env NEO4J_server_cluster.raft.advertised_address=public-address:7000 \
         --env NEO4J_server_default_advertised_address=public-address \
         --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
         --env NEO4j_server_bolt_advertised_address=public-address:7687 \
         --env NEO4j_server_http_advertised_address=public-address:7474 \
         neo4j:5.4.0-enterprise
```

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

Docker-specific operations

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

Use Neo4j Admin

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

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

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

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

Use Neo4j Import

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

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

and

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

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

Prerequisites

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

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

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

```
docker run --interactive --tty --rm \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j/data:/data \
    --volume=$HOME/neo4j/import:/import \
    --user="$(id -u):$(id -g)" \
    neo4j:5.4.0 \
neo4j-admin database import full --nodes=Movies=/import/movies_header.csv,/import/movies.csv \
--nodes=Actors=/import/actors_header.csv,/import/actors.csv \
--relationships=ACTED_IN=/import/roles_header.csv,/import/roles.csv
```

Use Neo4j Admin for memory recommendations

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

Example 3. Invoke `neo4j-admin server memory-recommendation --docker`

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

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

Use 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:5.4.0
   ```

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:5.4.0

# 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:5.4.0

# 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.

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:5.4.0
```

Configure Neo4j plugins

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

> This feature is intended to facilitate using Neo4j 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 `NEO4J_PLUGINS` environment variable can be used to specify the plugins to install using this method. This should be set to a JSON-formatted list of supported plugins.

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

```
--env NEO4J_PLUGINS='["apoc"]'
```

and run the following command:

```
docker run -it --rm \
  --publish=7474:7474 --publish=7687:7687 \
  --user="$(id -u):$(id -g)" \
  -e NEO4J_AUTH=none \
  --env NEO4J_PLUGINS='["apoc"]' \
  neo4j:5.4.0
```

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

```
--env NEO4J_PLUGINS='["apoc", "n10s"]'
```

Table 10. Supported Neo4j plugins

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

Name	Key	Further information
Neo Semantics	`n10s`	https://neo4j.com/labs/nsmtx-rdf/

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

SSL encryption in a Neo4j Docker container

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

Set up your certificate folders

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

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

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

2. Create a local folder to store your certificates.

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

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

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

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

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

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

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

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

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

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

Configure SSL via *neo4j.conf*

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

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


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

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:5.4.0
```

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:5.4.0
```

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

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.

Docker maintenance operations

This page describes how to perform basic maintenance operations when running Neo4j in a Docker container.

Dump and load a Neo4j database (offline)

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

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

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

```
docker run --interactive --tty --rm \
   --volume=$HOME/neo4j/data:/data \  ①
   --volume=$HOME/neo4j/backups:/backups \  ②
   neo4j/neo4j-admin:5.4.0 \
neo4j-admin database dump neo4j --to-path=/backups
```

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 database load` *to load your data into the new database.*

```
docker run --interactive --tty --rm \
    --volume=$HOME/neo4j/data:/data \ ①
    --volume=$HOME/neo4j/backups:/backups \ ②
    neo4j/neo4j-admin:5.4.0 \
neo4j-admin database load neo4j --from-path=/backups
```

① The volume that will contain 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 database dump and load` syntax and options, see `neo4j-admin database dump` and `neo4j-admin database load`.
For more information on managing volumes, see the official Docker documentation.

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

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

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

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

Back up a database using `docker exec` 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_server_backup_enabled=true \ ④
    neo4j:5.4.0-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 accepts the Neo4j Enterprise Edition license agreement.

④ The environment variable that enables online backups.

Example 9. Invoke `neo4j-admin database backup` to back up an online database, using `docker exec`:

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

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

Back up a database using `neo4j-admin` image Enterprise edition

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

Example 10. A `docker run` command that starts a database configured for backing up.

```
docker run \
    --detach \
    --publish=7474:7474 \
    --publish=7687:7687 \
    --publish=6362:6362 \ ①
    --volume=$HOME/neo4j-enterprise/data:/data \ ②
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \ ③
    --env NEO4J_server_backup_enabled=true \ ④
    --env NEO4J_server_backup_listen__address=0.0.0.0:6362 \ ⑤
    neo4j:5.4.0-enterprise
```

① The server.backup.listen_address port defined in 5.

② The volume that contains the database that you want to back up.

③ The environment variable that accepts the Neo4j Enterprise Edition license agreement.

④ The environment variable that enables online backups.

⑤ The environment variable that sets the server.backup.listen_address.

Once you have a backup enabled cluster node, the `neo4j/neo4j-admin:5.4.0-enterprise` docker image can be used to backup the database.

Example 11. Invoke `neo4j-admin` docker image to back up your database.

```
docker run --interactive --tty --rm \
   --volume=$HOME/neo4j-enterprise/backups:/backups \  ①
   --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \ ②
   neo4j/neo4j-admin:5.4.0-enterprise \
      neo4j-admin database backup <database name> \
         --to-path=/backups \
         --from=<backup node IP address>:6362 ③
```

① The volume that will be used for the backup database files.

② The environment variable that accepts the Neo4j Enterprise Edition license agreement.

③ The IP address of the backup cluster node and the server.backup.listen_address port.

Restore a database using `docker exec` Enterprise edition

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

Example 12. 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:5.4.0-enterprise
```

① The volume that contains all your databases.

② The volume that contains the database backup.

③ The environment variable that accepts the Neo4j Enterprise Edition license agreement.

Example 13. 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 14. Invoke `neo4j-admin database restore` to restore a database backup.

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

Restore a database using neo4j-admin image Enterprise edition

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

Example 15. A `docker run` command that creates a container to be used for restoring a database backup.

```
docker run --name <container name> \
    --detach \
    --volume=$HOME/neo4j-enterprise/data:/data \ ①
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \ ②
    neo4j:5.4.0-enterprise
```

① The volume that contains, or will contain, all your database data.

② The environment variable accepts the Neo4j Enterprise Edition license agreement.

Example 16. Stop the old database, then restore the backup database using `neo4j/neo4j-admin:5.4.0-enterprise`. Finally start the database again containing the new data.

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

```
docker run --interactive --tty --rm \
   --volume=$HOME/neo4j-enterprise/data:/data \ ①
   --volume=$HOME/neo4j-enterprise/backups:/backups \  ②
   --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \ ③
   neo4j/neo4j-admin:{neo4j-version-exact}-enterprise \
      neo4j-admin database restore \
         --from=/backups/<databasename>-<timestamp>.backup \
         --overwrite-destination \
          <database name> \
```

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

① The volume that contains, or will contain, all your database data. This must be the same data folder that the Neo4j DBMS container is using.

② The volume that contains the database backup.

③ The environment variable that accepts the Neo4j Enterprise Edition license agreement.

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

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

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.

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`
`browser.retain_editor_history`	`NEO4J_browser_retain__editor__history`
`client.allow_telemetry`	`NEO4J_client_allow__telemetry`
`db.checkpoint`	`NEO4J_dbms_checkpoint`
`db.checkpoint.interval.time`	`NEO4J_dbms_checkpoint_interval_time`
`db.checkpoint.interval.tx`	`NEO4J_dbms_checkpoint_interval_tx`
`db.checkpoint.interval.volume`	`NEO4J_db_checkpoint_interval_volume`
`db.checkpoint.iops.limit`	`NEO4J_dbms_checkpoint_iops_limit`
`db.cluster.catchup.pull_interval`	`NEO4J_db_cluster_catchup_pull__interval`
`db.cluster.raft.apply.buffer.max_bytes`	`NEO4J_db_cluster_raft_apply_buffer_max__bytes`
`db.cluster.raft.apply.buffer.max_entries`	`NEO4J_db_cluster_raft_apply_buffer_max__entries`
`db.cluster.raft.in_queue.batch.max_bytes`	`NEO4J_db_cluster_raft_in__queue_batch_max__byte`
`db.cluster.raft.in_queue.max_bytes`	`NEO4J_db_cluster_raft_in__queue_max__bytes`
`db.cluster.raft.leader_transfer.priority_group`	`NEO4J_db_cluster_raft_leader__transfer_priority__group`
`db.cluster.raft.log.prune_strategy`	`NEO4J_db_cluster_raft_log_prune__strategy`
`db.cluster.raft.log_shipping.buffer.max_bytes`	`NEO4J_db_cluster_raft_log__shipping_buffer_max__bytes`

Neo4j format	Docker format
db.cluster.raft.log_shipping.buffer.max_entries	NEO4J_db_cluster_raft_log__shipping_buffer_max__entries
db.filewatcher.enabled	NEO4J_db_filewatcher_enabled
db.format	NEO4J_db_format
db.import.csv.buffer_size	NEO4J_db_import_csv_buffer__size
db.import.csv.legacy_quote_escaping	NEO4J_db_import_csv_legacy__quote__escaping
db.index.fulltext.default_analyzer	NEO4J_db_index_fulltext_default__analyzer
db.index.fulltext.eventually_consistent	NEO4J_db_index_fulltext_eventually__consistent
db.index.fulltext.eventually_consistent_index_update_queue_max_length	NEO4J_db_index_fulltext_eventually__consistent__index__update__queue__max__length
db.index_sampling.background_enabled	NEO4J_db_index__sampling_background__enabled
db.index_sampling.sample_size_limit	NEO4J_db_index__sampling_sample__size__limit
db.index_sampling.update_percentage	NEO4J_db_index__sampling_update__percentage
db.lock.acquisition.timeout	NEO4J_db_lock_acquisition_timeout
db.logs.query.early_raw_logging_enabled	NEO4J_db_logs_query_early__raw__logging__enabled
db.logs.query.enabled	NEO4J_db_logs_query_enabled
db.logs.query.max_parameter_length	NEO4J_db_logs_query_max__parameter__length
db.logs.query.obfuscate_literals	NEO4J_db_logs_query_obfuscate__literals
db.logs.query.parameter_logging_enabled	NEO4J_db_logs_query_parameter__logging__enabled
db.logs.query.plan_description_enabled	NEO4J_db_logs_query_plan__description__enabled
db.logs.query.threshold	NEO4J_db_logs_query_threshold
db.logs.query.transaction.enabled	NEO4J_db_logs_query_transaction_enabled
db.logs.query.transaction.threshold	NEO4J_db_logs_query_transaction_threshold
db.memory.pagecache.warmup.enable	NEO4J_db_memory_pagecache_warmup_enable
db.memory.pagecache.warmup.preload	NEO4J_db_memory_pagecache_warmup_preload
db.memory.pagecache.warmup.preload.allowlist	NEO4J_db_memory_pagecache_warmup_preload_allowlist
db.memory.pagecache.warmup.profile.interval	NEO4J_db_memory_pagecache_warmup_profile_interval
db.memory.transaction.max	NEO4J_db_memory_transaction_max
db.memory.transaction.total.max	NEO4J_db_memory_transaction_total_max
db.recovery.fail_on_missing_files	NEO4J_db_recovery_fail__on__missing__files
db.relationship_grouping_threshold	NEO4J_db_relationship__grouping__threshold
db.shutdown_transaction_end_timeout	NEO4J_db_shutdown__transaction__end__timeout
db.store.files.preallocate	NEO4J_db_store_files_preallocate
db.temporal.timezone	NEO4J_db_temporal_timezone
db.track_query_cpu_time	NEO4J_db_track__query__cpu__time
db.transaction.bookmark_ready_timeout	NEO4J_db_transaction_bookmark__ready__timeout
db.transaction.concurrent.maximum	NEO4J_db_transaction_concurrent_maximum
db.transaction.monitor.check.interval	NEO4J_db_transaction_monitor_check_interval

Neo4j format	Docker format
db.transaction.sampling.percentage	NEO4J_db_transaction_sampling_percentage
db.transaction.timeout	NEO4J_db_transaction_timeout
db.transaction.tracing.level	NEO4J_db_transaction_tracing_level
db.tx_log.buffer.size	NEO4J_db_tx_log_buffer_size
db.tx_log.preallocate	NEO4J_db_tx__log_preallocate
db.tx_log.rotation.retention_policy	NEO4J_db_tx__log_rotation_retention__policy
db.tx_log.rotation.size	NEO4J_db_tx__log_rotation_size
db.tx_state.memory_allocation	NEO4J_db_tx__state_memory__allocation
dbms.cluster.catchup.client_inactivity_timeout	NEO4J_dbms_cluster_catchup_client__inactivity__timeout
dbms.cluster.discovery.endpoints	NEO4J_dbms_cluster_discovery_endpoints
dbms.cluster.discovery.log_level	NEO4J_dbms_cluster_discovery_log__level
dbms.cluster.discovery.type	NEO4J_dbms_cluster_discovery_type
dbms.cluster.minimum_initial_system_primaries_count	NEO4J_dbms_cluster_minimum__initial__system__primaries__count
dbms.cluster.network.handshake_timeout	NEO4J_dbms_cluster_network_handshake__timeout
dbms.cluster.network.max_chunk_size	NEO4J_dbms_cluster_network_max__chunk__size
dbms.cluster.network.supported_compression_algos	NEO4J_dbms_cluster_network_supported__compression__algos
dbms.cluster.raft.binding_timeout	NEO4J_dbms_cluster_raft_binding__timeout
dbms.cluster.raft.client.max_channels	NEO4J_dbms_cluster_raft_client_max__channels
dbms.cluster.raft.election_failure_detection_window	NEO4J_dbms_cluster_raft_election__failure__detection__window
dbms.cluster.raft.leader_failure_detection_window	NEO4J_dbms_cluster_raft_leader__failure__detection__window
dbms.cluster.raft.leader_transfer.balancing_strategy	NEO4J_dbms_cluster_raft_leader__transfer_balancing__strategy
dbms.cluster.raft.log.pruning_frequency	NEO4J_dbms_cluster_raft_log_pruning__frequency
dbms.cluster.raft.log.reader_pool_size	NEO4J_dbms_cluster_raft_log_reader__pool__size
dbms.cluster.raft.log.rotation_size	NEO4J_dbms_cluster_raft_log_rotation__size
dbms.cluster.raft.membership.join_max_lag	NEO4J_dbms_cluster_raft_membership_join__max__lag
dbms.cluster.raft.membership.join_timeout	NEO4J_dbms_cluster_raft_membership_join__timeout
dbms.cluster.store_copy.max_retry_time_per_request	NEO4J_dbms_cluster_store__copy_max__retry__time__per__request
dbms.cypher.forbid_exhaustive_shortestpath	NEO4J_dbms_cypher_forbid__exhaustive__shortestpath
dbms.cypher.hints_error	NEO4J_dbms_cypher_hints__error
dbms.cypher.lenient_create_relationship	NEO4J_dbms_cypher_lenient__create__relationship
dbms.cypher.min_replan_interval	NEO4J_dbms_cypher_min__replan__interval
dbms.cypher.planner	NEO4J_dbms_cypher_planner
dbms.cypher.render_plan_description	NEO4J_dbms_cypher_render__plan__description
dbms.cypher.statistics_divergence_threshold	NEO4J_dbms_cypher_statistics__divergence__threshold

Neo4j format	Docker format
dbms.databases.seed_from_uri_providers	NEO4J_dbms_databases_seed__from__uri__providers
dbms.db.timezone	NEO4J_dbms_db_timezone
dbms.kubernetes.address	NEO4J_dbms_kubernetes_address
dbms.kubernetes.ca_crt	NEO4J_dbms_kubernetes_ca__crt
dbms.kubernetes.cluster_domain	NEO4J_dbms_kubernetes_cluster__domain
dbms.kubernetes.label_selector	NEO4J_dbms_kubernetes_label__selector
dbms.kubernetes.namespace	NEO4J_dbms_kubernetes_namespace
dbms.kubernetes.service_port_name	NEO4J_dbms_kubernetes_service__port__name
dbms.kubernetes.token	NEO4J_dbms_kubernetes_token
dbms.logs.http.enabled	NEO4J_dbms_logs_http_enabled
db.lock.acquisition.timeout	NEO4J_dbms_lock_acquisition_timeout
server.logs.gc.enabled	NEO4J_server_logs_gc_enabled
server.logs.gc.options	NEO4J_server_logs_gc_options
server.logs.gc.rotation.keep_number	NEO4J_server_logs_gc_rotation_keep__number
server.logs.gc.rotation.size	NEO4J_server_logs_gc_rotation_size
dbms.logs.http.enabled	NEO4J_dbms_logs_http_enabled
dbms.memory.tracking.enable	NEO4J_dbms_memory_tracking_enable
dbms.memory.transaction.total.max	NEO4J_dbms_memory_transaction_total_max
dbms.netty.ssl.provider	NEO4J_dbms_netty_ssl_provider
dbms.routing.client_side.enforce_for_domains	NEO4J_dbms_routing_client__side_enforce__for__domains
dbms.routing.default_router	NEO4J_dbms_routing_default__router
dbms.routing.driver.connection.connect_timeout	NEO4J_dbms_routing_driver_connection_connect__timeout
dbms.routing.driver.connection.max_lifetime	NEO4J_dbms_routing_driver_connection_max__lifetime
dbms.routing.driver.connection.pool.acquisition_timeout	NEO4J_dbms_routing_driver_connection_pool_acquisition__timeout
dbms.routing.driver.connection.pool.idle_test	NEO4J_dbms_routing_driver_connection_pool_idle__test
dbms.routing.driver.connection.pool.max_size	NEO4J_dbms_routing_driver_connection_pool_max__size
dbms.routing.driver.logging.level	NEO4J_dbms_routing_driver_logging_level
dbms.routing.enabled	NEO4J_dbms_routing_enabled
dbms.routing.load_balancing.plugin	NEO4J_dbms_routing_load__balancing_plugin
dbms.routing.load_balancing.shuffle_enabled	NEO4J_dbms_routing_load__balancing_shuffle__enabled
dbms.routing.reads_on_primaries_enabled	NEO4J_dbms_routing_reads__on__primaries__enabled
dbms.routing.reads_on_writers_enabled	NEO4J_dbms_routing_reads__on__writers__enabled
dbms.routing_ttl	NEO4J_dbms_routing__ttl
dbms.security.allow_csv_import_from_file_urls	NEO4J_dbms_security_allow__csv__import__from__file__urls
dbms.security.auth_cache_max_capacity	NEO4J_dbms_security_auth__cache__max__capacity
dbms.security.auth_cache_ttl	NEO4J_dbms_security_auth__cache__ttl

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.cluster_status_auth_enabled	NEO4J_dbms_security_cluster__status__auth__enabled
dbms.security.http_access_control_allow_origin	NEO4J_dbms_security_http__access__control__allow__origin
dbms.security.http_auth_allowlist	NEO4J_dbms_security_http__auth__allowlist
dbms.security.http_strict_transport_security	NEO4J_dbms_security_http__strict__transport__security
dbms.security.key.name	NEO4J_dbms_security_key_name
dbms.security.keystore.password	NEO4J_dbms_security_keystore_password
dbms.security.keystore.path	NEO4J_dbms_security_keystore_path
dbms.security.ldap.authentication.attribute	NEO4J_dbms_security_ldap_authentication_attribute
dbms.security.ldap.authentication.cache_enabled	NEO4J_dbms_security_ldap_authentication_cache__enabled
dbms.security.ldap.authentication.mechanism	NEO4J_dbms_security_ldap_authentication_mechanism
dbms.security.ldap.authentication.search_for_attribute	NEO4J_dbms_security_ldap_authentication_search__for__attribute
dbms.security.ldap.authentication.user_dn_template	NEO4J_dbms_security_ldap_authentication_user__dn__template
dbms.security.ldap.authorization.access_permitted_group	NEO4J_dbms_security_ldap_authorization_access__permitted__group
dbms.security.ldap.authorization.group_membership_attributes	NEO4J_dbms_security_ldap_authorization_group__membership__attributes
dbms.security.ldap.authorization.group_to_role_mapping	NEO4J_dbms_security_ldap_authorization_group__to__role__mapping
dbms.security.ldap.authorization.nested_groups_enabled	NEO4J_dbms_security_ldap_authorization_nested__groups__enabled
dbms.security.ldap.authorization.nested_groups_search_filter	NEO4J_dbms_security_ldap_authorization_nested__groups__search__filter
dbms.security.ldap.authorization.system_password	NEO4J_dbms_security_ldap_authorization_system__password
dbms.security.ldap.authorization.system_username	NEO4J_dbms_security_ldap_authorization_system__username
dbms.security.ldap.authorization.use_system_account	NEO4J_dbms_security_ldap_authorization_use__system__account
dbms.security.ldap.authorization.user_search_base	NEO4J_dbms_security_ldap_authorization_user__search__base
dbms.security.ldap.authorization.user_search_filter	NEO4J_dbms_security_ldap_authorization_user__search__filter
dbms.security.ldap.connection_timeout	NEO4J_dbms_security__ldap_connection__timeout
dbms.security.ldap.host	NEO4J_dbms_security__ldap__host

Neo4j format	Docker format
dbms.security.ldap.read_timeout	NEO4J_dbms_security__ldap_read__timeout
dbms.security.ldap.referral	NEO4J_dbms_security__ldap_referral
dbms.security.ldap.use_starttls	NEO4J_dbms_security__ldap_use__starttls
dbms.security.log_successful_authentication	NEO4J_dbms_security_log__successful__authentication
dbms.security.oidc.<provider>.audience	NEO4J_dbms_security_oidc_<provider>_audience
dbms.security.oidc.<provider>.auth_endpoint	NEO4J_dbms_security_oidc_<provider>_auth__endpoint
dbms.security.oidc.<provider>.auth_flow	NEO4J_dbms_security_oidc_<provider>_auth__flow
dbms.security.oidc.<provider>.auth_params	NEO4J_dbms_security_oidc_<provider>_auth__params
dbms.security.oidc.<provider>.authorization.group_to_role_mapping	NEO4J_dbms_security_oidc_<provider>_authorization_group__to__role__mapping
dbms.security.oidc.<provider>.claims.groups	NEO4J_dbms_security_oidc_<provider>_claims_groups
dbms.security.oidc.<provider>.claims.username	NEO4J_dbms_security_oidc_<provider>_claims_username
dbms.security.oidc.<provider>.client_id	NEO4J_dbms_security_oidc_<provider>_client__id
dbms.security.oidc.<provider>.config	NEO4J_dbms_security_oidc_<provider>_config
dbms.security.oidc.<provider>.display_name	NEO4J_dbms_security_oidc_<provider>_display__name
dbms.security.oidc.<provider>.get_groups_from_user_info	NEO4J_dbms_security_oidc_<provider>_get__groups__from__user__info
dbms.security.oidc.<provider>.get_username_from_user_info	NEO4J_dbms_security_oidc_<provider>_get__username__from__user__info
dbms.security.oidc.<provider>.issuer	NEO4J_dbms_security_oidc_<provider>_issuer
dbms.security.oidc.<provider>.jwks_uri	NEO4J_dbms_security_oidc_<provider>_jwks__uri
dbms.security.oidc.<provider>.params	NEO4J_dbms_security_oidc_<provider>_params
dbms.security.oidc.<provider>.token_endpoint	NEO4J_dbms_security_oidc_<provider>_token__endpoint
dbms.security.oidc.<provider>.token_params	NEO4J_dbms_security_oidc_<provider>_token__params
dbms.security.oidc.<provider>.user_info_uri	NEO4J_dbms_security_oidc_<provider>_user__info__uri
dbms.security.oidc.<provider>.well_known_discovery_uri	NEO4J_dbms_security_oidc_<provider>_well__known__discovery__uri
dbms.security.procedures.allowlist	NEO4J_dbms_security_procedures_allowlist
dbms.security.procedures.unrestricted	NEO4J_dbms_security_procedures_unrestricted
initial.dbms.database_allocator	NEO4J_initial_dbms_database__allocator
initial.dbms.default_database	NEO4J_initial_dbms_default__database
initial.dbms.default_primaries_count	NEO4J_initial_dbms_default__primaries__count
initial.dbms.default_secondaries_count	NEO4J_initial_dbms_default__secondaries__count
initial.server.allowed_databases	NEO4J_initial_server_allowed__databases
initial.server.denied_databases	NEO4J_initial_server_denied__databases
initial.server.mode_constraint	NEO4J_initial_server_mode__constraint
server.backup.enabled	NEO4J_server_backup_enabled
server.backup.listen_address	NEO4J_server_backup_listen__address

Neo4j format	Docker format
server.backup.store_copy_max_retry_time_per_request	NEO4J_server_backup_store__copy__max__retry__time__per__request
server.bolt.advertised_address	NEO4J_server_bolt_advertised__address
server.bolt.connection_keep_alive	NEO4J_server_bolt_connection__keep__alive
server.bolt.connection_keep_alive_for_requests	NEO4J_server_bolt_connection__keep__alive__for__requests
server.bolt.connection_keep_alive_probes	NEO4J_server_bolt_connection__keep__alive__probes
server.bolt.connection_keep_alive_streaming_scheduling_interval	NEO4J_server_bolt_connection__keep__alive__streaming__scheduling__interval
server.bolt.enabled	NEO4J_server_bolt_enabled
server.bolt.listen_address	NEO4J_server_bolt_listen__address
server.bolt.ocsp_stapling_enabled	NEO4J_server_bolt_ocsp__stapling__enabled
server.bolt.thread_pool_keep_alive	NEO4J_server_bolt_thread__pool__keep__alive
server.bolt.thread_pool_max_size	NEO4J_server_bolt_thread__pool__max__size
server.bolt.thread_pool_min_size	NEO4J_server_bolt_thread__pool__min__size
server.bolt.tls_level	NEO4J_server_bolt_tls__level
server.cluster.advertised_address	NEO4J_server_cluster_advertised__address
server.cluster.catchup.connect_randomly_to_server_group	NEO4J_server_cluster_catchup_connect__randomly__to__server__group
server.cluster.catchup.upstream_strategy	NEO4J_server_cluster_catchup_upstream__strategy
server.cluster.catchup.user_defined_upstream_strategy	NEO4J_server_cluster_catchup_user__defined__upstream__strategy
server.cluster.listen_address	NEO4J_server_cluster_listen__address
server.cluster.network.native_transport_enabled	NEO4J_server_cluster_network_native__transport__enabled
server.cluster.raft.advertised_address	NEO4J_server_cluster_raft_advertised__address
server.cluster.raft.listen_address	NEO4J_server_cluster_raft_listen__address
server.cluster.system_database_mode	NEO4J_server_cluster_system__database__mode
server.config.strict_validation.enabled	NEO4J_server_config_strict__validation_enabled
server.databases.default_to_read_only	NEO4J_server_databases_default__to__read__only
server.databases.read_only	NEO4J_server_databases_read__only
server.databases.writable	NEO4J_server_databases_writable
server.db.query_cache_size	NEO4J_server_db_query__cache__size
server.default_advertised_address	NEO4J_server_default__advertised__address
server.default_listen_address	NEO4J_server_default__listen__address
server.directories.cluster_state	NEO4J_server_directories_cluster__state
server.directories.data	NEO4J_server_directories_data
server.directories.dumps.root	NEO4J_server_directories_dumps_root
server.directories.import	NEO4J_server_directories_import
server.directories.lib	NEO4J_server_directories_lib

Neo4j format	Docker format
server.directories.licenses	NEO4J_server_directories_licenses
server.directories.logs	NEO4J_server_directories_logs
server.directories.metrics	NEO4J_server_directories_metrics
server.directories.neo4j_home	NEO4J_server_directories_neo4j__home
server.directories.plugins	NEO4J_server_directories_plugins
server.directories.run	NEO4J_server_directories_run
server.directories.script.root	NEO4J_server_directories_script_root
server.directories.transaction.logs.root	NEO4J_server_directories_transaction_logs_root
server.discovery.advertised_address	NEO4J_server_discovery_advertised__address
server.discovery.listen_address	NEO4J_server_discovery_listen__address
server.dynamic.setting.allowlist	NEO4J_server_dynamic_setting_allowlist
server.groups	NEO4J_server_groups
server.http.advertised_address	NEO4J_server_http_advertised__address
server.http.enabled	NEO4J_server_http_enabled
server.http.listen_address	NEO4J_server_http_listen__address
server.http_enabled_modules	NEO4J_server_http__enabled__modules
server.https.advertised_address	NEO4J_server_https_advertised__address
server.https.enabled	NEO4J_server_https_enabled
server.https.listen_address	NEO4J_server_https_listen__address
server.jvm.additional	NEO4J_server_jvm_additional
server.logs.config	NEO4J_server_logs_config
server.logs.debug.enabled	NEO4J_server_logs_debug_enabled
server.logs.gc.enabled	NEO4J_server_logs_gc_enabled
server.logs.gc.options	NEO4J_server_logs_gc_options
server.logs.gc.rotation.keep_number	NEO4J_server_logs.gc_rotation_keep__number
server.logs.gc.rotation.size	NEO4J_server_logs_gc_rotation_size
server.logs.user.config	NEO4J_server_logs_user_config
server.max_databases	NEO4J_server._max__databases
server.memory.heap.initial_size	NEO4J_server_memory_heap_initial__size
server.memory.heap.max_size	NEO4J_server_memory_heap_max__size
server.memory.off_heap.block_cache_size	NEO4J_server_memory_off__heap_block__cache__size
server.memory.off_heap.max_cacheable_block_size	NEO4J_server_memory_off__heap_max__cacheable__block__size
server.memory.off_heap.max_size	NEO4J_server_memory_off__heap_max__size
server.memory.pagecache.directio	NEO4J_server_memory_pagecache_directio
server.memory.pagecache.flush.buffer.enabled	NEO4J_server_memory_pagecache_flush_buffer_enabled
server.memory.pagecache.flush.buffer.size_in_pages	NEO4J_server_memory_pagecache_flush_buffer_size__in__pages

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

Kubernetes

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

This chapter describes the following:

- Introduction — Introduction to running Neo4j on a Kubernetes cluster using Neo4j Helm charts.
- Configure the Neo4j Helm chart repository — Configure the Neo4j Helm chart repository and check for the available charts.
- Quickstart: Deploy a standalone instance — Deploy a Neo4j standalone instance to a cloud (GKE, AWS, AKS) or a local (via Docker Desktop for macOS) Kubernetes cluster.
- Quickstart: Deploy a cluster — Deploy a Neo4j cluster to a cloud (GKE, AWS, AKS) Kubernetes cluster.
- Configure a Neo4j deployment — Configure a Neo4j deployment using a customized *values.yaml* file.
- Persistent volumes — Use persistent volumes with the Neo4j Helm chart and what types Neo4j supports.
- Access a Neo4j DBMS — Access a Neo4j DBMS running on Kubernetes.
- Access a Neo4j cluster — Access a Neo4j cluster running on Kubernetes.
- Import data — Import data into a Neo4j database.
- Monitoring — Monitor a Neo4j deployment running on Kubernetes.
- Operations — Maintain a Neo4j deployment running on Kubernetes.
- Deploy a single Neo4j cluster across multiple AKS clusters — Deploy a single Neo4j cluster with three primary servers running on three different AKS clusters.
- Troubleshooting — Diagnose and troubleshoot a Neo4j deployment running on Kubernetes.

Introduction

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

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

The Neo4j Helm chart repository

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

Using the Neo4j Helm chart repository

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

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

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

Configure the Neo4j Helm chart repository

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

Prerequisites

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

Configure the Neo4j Helm chart repository

1. Add the Neo4j Helm chart repository.

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

2. Update the repository:

```
helm repo update
```

Check for the available Neo4j Helm charts

```
helm search repo neo4j/
```

An example result

```
NAME                                    CHART VERSION   APP VERSION DESCRIPTION
neo4j/neo4j                             5.1.0           5.1.0       Neo4j is the world's leading graph
database
neo4j/neo4j-cluster-core                4.4.13          4.4.13      Neo4j is the world's leading graph
database
neo4j/neo4j-cluster-headless-service    4.4.13          -           Neo4j is the world's leading graph
database
neo4j/neo4j-cluster-loadbalancer        4.4.13          -           Neo4j is the world's leading graph
database
neo4j/neo4j-cluster-read-replica        4.4.13          4.4.13      Neo4j is the world's leading graph
database
neo4j/neo4j-docker-desktop-pv           4.4.13          -           Sets up persistent disks suitable for
simple de...
neo4j/neo4j-gcloud-pv                   4.4.13          -           Sets up persistent disks suitable for
simple de...
neo4j/neo4j-headless-service            5.1.0           -           Neo4j is the world's leading graph
database
neo4j/neo4j-persistent-volume           5.1.0           -           Sets up persistent disks suitable for
a Neo4j H...
neo4j/neo4j-standalone                  4.4.13          4.4.13      Neo4j is the world's leading graph
database
```

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

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

Quickstart: Deploy a standalone instance

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

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

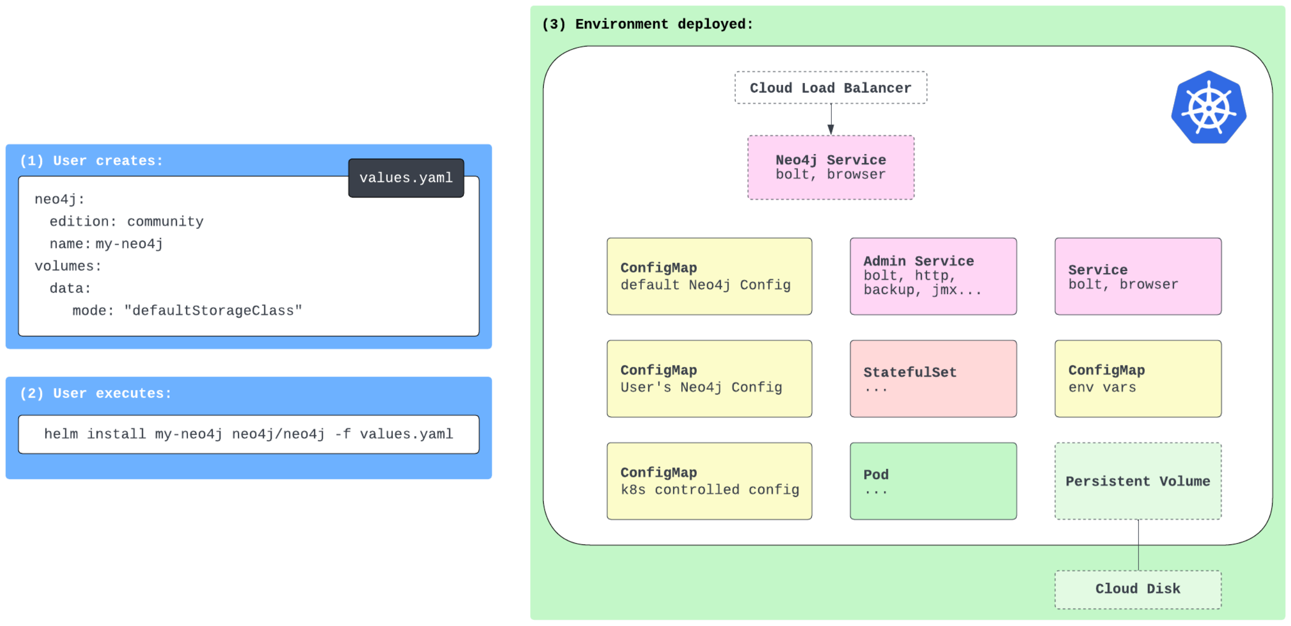
- Neo4j Helm chart for standalone server deployments — A schematic representation of how to use the Neo4j Helm chart for deploying a standalone server.
- Prerequisites — Set up your environment for deploying a Neo4j standalone instance on Kubernetes.
- Create a Helm deployment *values.yaml* file — Create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

Neo4j Helm chart for standalone server deployments

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

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

Neo4j standalone server setup


(1) User creates:
values.yaml
(2) User executes:
(3) Environment deployed:
Cloud Load Balancer
Neo4j Service bolt, browser
ConfigMap default Neo4j Config
Admin Service bolt, http, backup, jmx...
Service bolt, browser
ConfigMap User's Neo4j Config
StatefulSet ...
ConfigMap env vars
ConfigMap k8s controlled config
Pod ...
Persistent Volume
Cloud Disk

```
neo4j:
  edition: community
  name: my-neo4j
volumes:
  data:
    mode: "defaultStorageClass"
```

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

Prerequisites

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

General prerequisites

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

> This guide works with minimum CPU and memory allocated per Neo4j instance. However, the Neo4j system requirements largely depend on the use of the software. Therefore, for running Neo4j in development or production environments, please refer to System requirements.

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

Environment-specific prerequisites

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

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

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

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

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

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

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

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

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

1. Install the `aws` command-line interface (CLI) (https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html). Make sure you complete the AWS configuration step (https://docs.aws.amazon.com/cli/latest/userguide/getting-started-quickstart.html).
2. Install the `eksctl` command-line interface (CLI) (https://docs.aws.amazon.com/eks/latest/userguide/getting-started-eksctl.html).
3. All the shell commands in this guide assume that the AWS region to use has been set using the `AWS_DEFAULT_REGION` environment variable, for example:

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

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

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

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

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

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

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

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

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

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

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

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

1. Install the az command-line interface (CLI) (https://docs.microsoft.com/en-us/cli/azure/install-azure-cli).
2. Verify that you have a Resource Group with:
   - An Azure Kubernetes Service (AKS) cluster.
   - The AKS cluster principal needs to be assigned roles that allow it to manage Microsoft.Compute/disks in the Resource Group.
3. Set the Resource group and the location to use as defaults using:

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

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

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

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

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

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

Create a Helm deployment *values.yaml* file

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

Important configuration parameters

`neo4j.name`

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

`neo4j.resources`

The size of a Neo4j instance is defined by the values of the `neo4j.resources.cpu` and `neo4j.resources.memory` parameters. The minimum is `0.5` CPU and `2GB` memory. If invalid or less than the minimum values are provided, Helm will throw an error, for example:

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

For more information, see Configure resource allocation.

`neo4j.password`

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

You cannot use `neo4j` as the initial password as this is the default password.

`neo4j.edition` and `neo4j.acceptLicenseAgreement`

By default, the standalone Helm chart installs Neo4j Community Edition. If you want to install Neo4j Enterprise Edition, set the configuration parameters `edition: "enterprise"` and acknowledge license compliance by setting `neo4j.acceptLicenseAgreement` to `"yes"`.

`volumes.data`

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

For details of all Neo4j Helm chart configuration options, see Configure a Neo4j Helm deployment.

Create a *values.yaml* file

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Install a Neo4j standalone instance

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

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

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

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

b. Install the Neo4j standalone server:

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

Example output

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

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

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

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

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

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

Graphs are everywhere!
```

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

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

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

Verify the installation

1. Check that the `statefulset` is OK.

```
kubectl get statefulsets
```

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

2. Check that the pod is `Running`:

```
kubectl get pods
```

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

3. Check that the pod logs look OK:

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

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

4. Check that the services look OK:

```
kubectl get services
```

```
NAME                       TYPE        CLUSTER-IP      EXTERNAL-IP    PORT(S)
AGE
my-neo4j-release           ClusterIP   10.28.9.110     <none>
7687/TCP,7474/TCP,7473/TCP              2m24s
my-neo4j-release-admin     ClusterIP   10.28.1.251     <none>
6362/TCP,7687/TCP,7474/TCP,7473/TCP     2m24s
```

   When installing the Community edition, no `LoadBalancer` service is created (Enterprise is required for this feature). To access the Neo4j Browser with the Community edition, forward port `7474`:

```
kubectl port-forward svc/my-neo4j-release 7475:7474 7687:7687
```

5. In a web browser, open the Neo4j Browser at http://localhost:7475/browser.

6. Use the automatically-generated password (as printed in the output of the `helm install` command) or the one you have configured in the *my-neo4j.values.yaml* file.

1. Check that `statefulset` is OK.

```
kubectl get statefulsets
```

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

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

```
kubectl get pvc
```

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

3. Check that the pod is `READY`:

```
kubectl get pods
```

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

4. Check that the pod logs look OK:

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

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

5. Check that the services look OK:

```
kubectl get services
```

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

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

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

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

8. Use the automatically-generated password (as printed in the output of the `helm install` command) or the one you have set up with the `helm install` command.

Uninstall Neo4j and clean up the created resources

Uninstall Neo4j Helm deployment

Uninstall the Neo4j Helm deployment.

```
helm uninstall my-neo4j-release
```

Example output

```
release "my-neo4j-release" uninstalled
```

Fully remove all the data and resources

Uninstalling the Helm release does not remove the created resources and data. Therefore, after uninstalling the helm deployment, you also have to delete all the data and resources.

1. Delete all persistent volume claims in the `neo4j` namespace:

```
kubectl delete pvc --all --namespace neo4j
```

2. Delete the entire Kubernetes cluster in your cloud provider:

```
gcloud container clusters delete my-neo4j-gke-cluster
```

```
eksctl delete cluster --name=my-neo4j-eks-cluster
```

```
az aks delete --name my-neo4j-aks-cluster --resource-group <MyResourceGroup>
```

Quickstart: Deploy a cluster

The quickstart for deploying a Neo4j cluster contains the following:

This guide shows how to deploy a cluster with three servers.

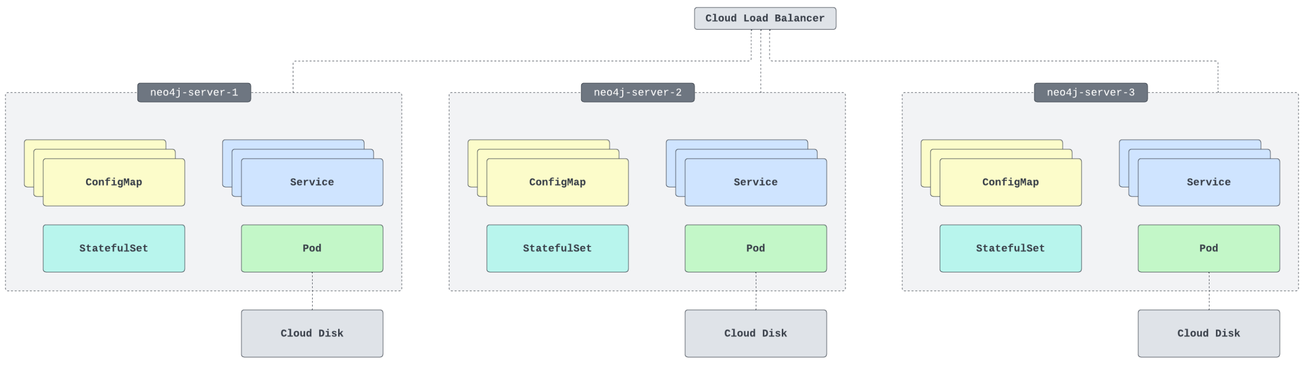
- Neo4j Helm chart for cluster deployments — A schematic representation of how to use the Neo4j Helm chart for deploying a cluster.
- Prerequisites — Set up your environment for deploying a Neo4j cluster on Kubernetes.
- Create Helm deployment values files — Create a Helm deployment *values.yaml* file for each Neo4j cluster member.
- Install Neo4j cluster servers — Install each of your Neo4j cluster servers using its deployment YAML file and the *neo4j/neo4j* Helm chart.
- Verify cluster formation — Verify that the Neo4j servers have formed a cluster.
- Access the Neo4j cluster from inside Kubernetes — Access the Neo4j cluster from inside Kubernetes using a specific server or the headless service.
- Access the Neo4j cluster from outside Kubernetes — Access the Neo4j cluster from outside Kubernetes using a load balancer.
- Uninstall the Neo4j cluster and clean up the created resources — Uninstall all Neo4j Helm deployments and clean up the created resources.

Neo4j Helm chart for cluster deployments

To offer the flexibility required to cluster users, the Helm chart for clusters takes a modular approach: the typical cluster deployment requires several Helm chart installations. For example, a cluster with three servers requires installing the *neo4j* chart three times. By separating the charts up, users can create a different topology of Neo4j and a different topology in their Kubernetes clusters to suit their needs.

The following diagram is a schematic representation of the Helm charts involved and the Kubernetes and Cloud resources they instantiate when installed:

Neo4j cluster setup



The diagram shows an example of a Neo4j cluster setup with three servers. The Kubernetes setup includes a headless service for accessing the cluster from inside Kubernetes and a load-balancer service for accessing the cluster from outside Kubernetes.

Prerequisites

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

General prerequisites

- Configure the Neo4j Helm chart repository.
- Obtain a valid license for Neo4j Enterprise Edition. For more information, see https://neo4j.com/licensing/ or use the form Contact Neo4j.
- Install the Kubernetes client command-line tool `kubectl` (https://kubernetes.io/docs/tasks/tools/).
- Set up a Kubernetes cluster with sufficient CPU and memory for your Neo4j deployment.

  > This guide works with minimum CPU and memory allocated per Neo4j instance. However, the Neo4j system requirements largely depend on the use of the software. Therefore, for running Neo4j in development or production environments, please refer to System requirements.

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

Environment-specific prerequisites

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

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

```
export CLOUDSDK_CORE_PROJECT="neo4j-helm"
export CLOUDSDK_COMPUTE_ZONE="europe-west6-c"
export CLOUDSDK_COMPUTE_REGION="europe-west6"
export CLOUDSDK_CONTAINER_CLUSTER="my-neo4j-gke-cluster"
```

3. If you do not have a Google Kubernetes Engine (GKE) cluster, you can create a multi-node cluster (one node per Neo4j instance) using:

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

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

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

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

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

1. Install the aws command-line interface (CLI) (https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html). Make sure you complete the AWS configuration step (https://docs.aws.amazon.com/cli/latest/userguide/getting-started-quickstart.html).
2. Install the eksctl command-line interface (CLI) (https://docs.aws.amazon.com/eks/latest/userguide/getting-started-eksctl.html).
3. All the shell commands in this guide assume that the AWS region to use has been set using the AWS_DEFAULT_REGION environment variable, for example:

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

4. If you do not have an AWS Elastic Kubernetes Service (EKS) cluster, you can create a multi-node cluster (one node per Neo4j instance) using the following command:

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

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

```
eksctl create cluster --name "my-neo4j-eks-cluster" --region "${AWS_DEFAULT_REGION}"
--nodegroup-name "neo4j-nodes" --nodes-min 1 --nodes-max 4 --node-type c4.xlarge --nodes 4
--node-volume-size 10 --ssh-access --with-oidc
```

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

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

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

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

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

7. Configure kubectl to use your EKS cluster using:

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

1. Install the az command-line interface (CLI) (https://docs.microsoft.com/en-us/cli/azure/install-azure-cli).
2. Verify that you have a Resource Group with:
   - An Azure Kubernetes Service (AKS) cluster.
   - The AKS cluster principal needs to be assigned roles that allow it to manage Microsoft.Compute/disks in the Resource Group.
3. Set the Resource group and the location to use as defaults using:

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

If you do not have an AKS cluster, follow the steps to create a multi-node cluster (one node per Neo4j instance).

1. Create a cluster by running:

```
az aks create --name my-neo4j-aks-cluster --node-count=<num>
```

2. Configure kubectl to use your AKS cluster using:

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

Create Helm deployment values files

You create a Helm deployment YAML file for each Neo4j cluster member with all the configuration settings.

Important configuration parameters

`neo4j.name`

All Neo4j cluster servers are linked together by the value of the parameter `neo4j.name`. When installed via the Neo4j Helm chart, they will join the cluster identified by `neo4j.name`.
`neo4j.name` is different from the Helm *release name*, which is used to reference a cluster server elsewhere in Kubernetes.
All other names and labels of K8s objects created by the Helm chart derive from both `neo4j.name` and *release name*. The `neo4j.name` parameter is mandatory, and the installation will fail if it is not specified.
`neo4j.name` must be unique within a namespace.

If you want to install 2 clusters in the same namespace, they must have different values for `neo4j.name`.

`neo4j.minimumClusterSize`

Starting from Neo4j 5.0.0, standalone servers and cluster servers have no distinction. By default, servers in a cluster can host primary and secondary databases. See the Operational view for more details.
`neo4j.minimumClusterSize` is set to 1 by default, which means the server starts without waiting for the other servers. When installing a cluster, you should set `neo4j.minimumClusterSize` to the number of desired members in the cluster. If you later decide to add an extra cluster server in excess of `neo4j.minimumClusterSize`, you need to manually enable it using the Cypher command `ENABLE SERVER`. For more information on enabling new servers, see Add a server to the cluster.

`neo4j.resources`

The size of a Neo4j cluster member is defined by the values of the `neo4j.resources.cpu` and `neo4j.resources.memory` parameters. The minimum for a Neo4j instance is `0.5` CPU and `2GB` memory. If invalid or less than the minimum values are provided, Helm will throw an error, for example:

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

For more information, see Configure resource allocation.

`neo4j.password`

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

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

`neo4j.edition` and `neo4j.acceptLicenseAgreement`

By default, the Neo4j Helm chart installs Neo4j Enterprise Edition since clusters are not available in Community Edition. This means that you do not have to explicitly set the configuration parameter `edition:` to `"enterprise"`, as you would do for a standalone server of Neo4j Enterprise Edition. However, you must acknowledge license compliance by setting `neo4j.acceptLicenseAgreement` to `"yes"`. For more information on how to obtain a valid license for Neo4j Enterprise Edition, see https://neo4j.com/licensing/ or use the form Contact Neo4j.

`volumes.data`

The `volumes.data` parameter maps the `data` volume mount of each cluster member to the persistent volume for that member. For more information, see Volume mounts and persistent volumes with the Neo4j Helm chart.

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

Create a values.yaml file for each server

Select the tab as per your Kubernetes environment, and using the provided example, create a YAML file for each of your cluster servers with all the configuration settings.

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

The examples use storage classes provided by the vendor. If you want to use a different type of storage class, such as regional disks, consult the cloud vendor documentation on creating new storage classes.

```
neo4j:
  name: "my-cluster"
  minimumClusterSize: 3
  resources:
    cpu: "0.5"
    memory: "2Gi"
  password: "my-password"
  acceptLicenseAgreement: "yes"

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

```
neo4j:
  name: "my-cluster"
  minimumClusterSize: 3
  resources:
    cpu: "0.5"
    memory: "2Gi"
  password: "my-password"
  acceptLicenseAgreement: "yes"

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

```
neo4j:
  name: "my-cluster"
  minimumClusterSize: 3
  resources:
    cpu: "0.5"
    memory: "2Gi"
  password: "my-password"
  acceptLicenseAgreement: "yes"

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

Install Neo4j cluster servers

Getting everything to work in Kubernetes requires that certain K8s objects have specific names that are referenced elsewhere. Each Neo4j instance is a Helm “release” and has a *release name*. Release name must consist of lowercase alphanumeric characters, `-` or `.`, and must start and end with an alphanumeric character.

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

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

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

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

   b. Install `server-1`:

```
helm install server-1 neo4j/neo4j --namespace neo4j -f server-1.values.yaml
```

Example output

```
NAME: server-1
LAST DEPLOYED: Wed Oct 26 11:50:41 2022
NAMESPACE: neo4j
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Thank you for installing neo4j.

Your release "server-1" has been installed in namespace "neo4j".

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

This release creates a single member of a Neo4j cluster. It will not become ready until it is able
to form a working Neo4j cluster by joining other Neo4j servers. To create a working cluster at
least 3 servers are required.

Once you have a working Neo4j cluster, you can access the Neo4j browser using the IP address of
the my-cluster-lb-neo4j service
eg. http://[SERVICE_IP]:7474

Graphs are everywhere!
```

   c. Install `server-2`:

```
helm install server-2 neo4j/neo4j --namespace neo4j -f server-2.values.yaml
```

Example output

```
NAME: server-2
LAST DEPLOYED: Wed Oct 26 11:51:27 2022
NAMESPACE: neo4j
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Thank you for installing neo4j.

Your release "server-2" has been installed in namespace "neo4j".

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

This release creates a single member of a Neo4j cluster. It will not become ready until it is able
to form a working Neo4j cluster by joining other Neo4j servers. To create a working cluster at
least 3 servers are required.

Once you have a working Neo4j cluster, you can access the Neo4j browser using the IP address of
the my-cluster-lb-neo4j service
eg. http://[SERVICE_IP]:7474

Graphs are everywhere!
```

d. Install `server-3`:

```
helm install server-3 neo4j/neo4j --namespace neo4j -f server-3.values.yaml
```

Example output

```
NAME: server-3
LAST DEPLOYED: Wed Oct 26 11:52:02 2022
NAMESPACE: neo4j
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Thank you for installing neo4j.

Your release "server-3" has been installed in namespace "neo4j".

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

This release creates a single member of a Neo4j cluster. It will not become ready until it is able
to form a working Neo4j cluster by joining other Neo4j servers. To create a working cluster at
least 3 servers are required.

Once you have a working Neo4j cluster, you can access the Neo4j browser using the IP address of
the my-cluster-lb-neo4j service
eg. http://[SERVICE_IP]:7474

Graphs are everywhere!
```

If you have not passed a password for the `neo4j` user, the Neo4j Helm chart has automatically generated one for you. The password is the same for all cluster members. You can find it in the Helm install outputs. Make a note of it.

2. Verify that the installed servers have formed a cluster. See the next section Verify the Neo4j cluster formation.

Verify the Neo4j cluster formation

You check the pods and services to verify that the servers have managed to form a cluster.

1. Check that the pods are `READY`.

   Typically, it takes a minute or two after the pods are running. They become `READY` after they form a cluster. You can watch pod status changes by adding the `-w` option to the `kubectl` command (`kubectl get pods -w`).

   ```
   kubectl get pods
   ```

   ```
   NAME        READY   STATUS    RESTARTS   AGE
   server-1-0   1/1     Running   0          147m
   server-2-0   1/1     Running   0          147m
   server-3-0   1/1     Running   0          147m
   ```

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

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

   ```
   2022-10-26 11:37:42.516+0000 INFO  Remote interface available at http://localhost:7474/
   2022-10-26 11:37:42.539+0000 INFO  id:
   72419021DEA149919ABFFD4930EDE02131E52E77F7EADF62C6323BC4F2D01EC5
   2022-10-26 11:37:42.539+0000 INFO  name: system
   2022-10-26 11:37:42.539+0000 INFO  creationDate: 2022-10-26T11:37:28.784Z
   2022-10-26 11:37:42.540+0000 INFO  Started.
   2022-10-26 11:37:47.177+0000 INFO  This instance bootstrapped the 'neo4j' database.
   2022-10-26 11:37:58.644+0000 INFO  Connected to server-3-
   internals.neo4j.svc.cluster.local/10.24.0.131:7000 [RAFT version:1.0]
   2022-10-26 12:01:32.066+0000 INFO  Direct driver instance 577812770 created for server address server-
   1-internals.neo4j.svc.cluster.local:7688
   2022-10-26 12:02:40.172+0000 INFO  Closing driver instance 577812770
   2022-10-26 12:02:40.174+0000 INFO  Closing connection pool towards server-1-
   internals.neo4j.svc.cluster.local:7688
   ```

3. Check that the services look good:

   ```
   kubectl get services
   ```

```
NAME                TYPE           CLUSTER-IP       EXTERNAL-IP     PORT(S)
AGE
my-cluster-lb-neo4j   LoadBalancer   10.28.12.119   35.234.152.117
7474:32169/TCP,7473:32145/TCP,7687:32624/TCP                              148m
server-1              ClusterIP      10.28.13.216   <none>          7687/TCP,7474/TCP,7473/TCP
148m
server-1-admin        ClusterIP      10.28.13.244   <none>
6362/TCP,7687/TCP,7474/TCP,7473/TCP                                       148m
server-1-internals    ClusterIP      None           <none>
6362/TCP,7687/TCP,7474/TCP,7473/TCP,7688/TCP,5000/TCP,7000/TCP,6000/TCP   148m
server-2              ClusterIP      10.28.10.237   <none>          7687/TCP,7474/TCP,7473/TCP
148m
server-2-admin        ClusterIP      10.28.7.113    <none>
6362/TCP,7687/TCP,7474/TCP,7473/TCP                                       148m
server-2-internals    ClusterIP      None           <none>
6362/TCP,7687/TCP,7474/TCP,7473/TCP,7688/TCP,5000/TCP,7000/TCP,6000/TCP   148m
server-3              ClusterIP      10.28.3.164    <none>          7687/TCP,7474/TCP,7473/TCP
148m
server-3-admin        ClusterIP      10.28.14.77    <none>
6362/TCP,7687/TCP,7474/TCP,7473/TCP                                       148m
server-3-internals    ClusterIP      None           <none>
6362/TCP,7687/TCP,7474/TCP,7473/TCP,7688/TCP,5000/TCP,7000/TCP,6000/TCP   148m
```

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

Access the Neo4j cluster from inside Kubernetes

By default, client-side routing is used for accessing a Neo4j cluster from inside Kubernetes.

Access the Neo4j cluster using a specific member

You run `cypher-shell` in a new pod and point it directly to one of the servers.

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

```
kubectl run --rm -it --image "neo4j:5.4.0-enterprise" cypher-shell \
     -- cypher-shell -a "neo4j://server-3.default.svc.cluster.local:7687" -u neo4j -p "my-password"
```

```
If you don't see a command prompt, try pressing enter.


Connected to Neo4j using Bolt protocol version 5 at neo4j://server-3.default.svc.cluster.local:7687 as
user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
```

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

```
SHOW DATABASES;
```

```
+----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+
| name     | type       | aliases | access       | address                           | role
| writer | requestedStatus | currentStatus | statusMessage | default | home  | constituents |
+----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+
| "neo4j"  | "standard" | []      | "read-write" | "server-2.neo4j.svc.cluster.local:7687" | "primary"
| TRUE   | "online"        | "online"      | ""            | TRUE    | TRUE  | []           |
| "neo4j"  | "standard" | []      | "read-write" | "server-1.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""            | TRUE    | TRUE  | []           |
| "neo4j"  | "standard" | []      | "read-write" | "server-3.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""            | TRUE    | TRUE  | []           |
| "system" | "system"   | []      | "read-write" | "server-2.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""            | FALSE   | FALSE | []           |
| "system" | "system"   | []      | "read-write" | "server-1.neo4j.svc.cluster.local:7687" | "primary"
| TRUE   | "online"        | "online"      | ""            | FALSE   | FALSE | []           |
| "system" | "system"   | []      | "read-write" | "server-3.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""            | FALSE   | FALSE | []           |
+----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+

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

3. Run the Cypher command `SHOW SERVERS` to verify that all cluster servers are enabled:

```
SHOW SERVERS;
```

```
+----------------------------------------------------------------------------------------------------
------------------------------+
| name                                    | address                                 | state     |
health      | hosting               |
+----------------------------------------------------------------------------------------------------
------------------------------+
| "ad5c3cf1-541a-44f8-a19b-28bc36030914" | "server-3.neo4j.svc.cluster.local:7687" | "Enabled" |
"Available" | ["system", "neo4j"] |
| "cbdebc59-64c2-4542-a041-24a1f051e64f" | "server-1.neo4j.svc.cluster.local:7687" | "Enabled" |
"Available" | ["system", "neo4j"] |
| "f37e98a7-15ec-4dc4-a6bf-df9e418a7488" | "server-2.neo4j.svc.cluster.local:7687" | "Enabled" |
"Available" | ["system", "neo4j"] |
+----------------------------------------------------------------------------------------------------
------------------------------+

3 rows
ready to start consuming query after 27 ms, results consumed after another 363 ms
```

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

```
:exit;
```

```
Bye!
Session ended, resume using 'kubectl attach cypher-shell -c cypher-shell -i -t' command when the pod
is running
pod "cypher-shell" deleted
```

Access the Neo4j cluster using headless service

To allow for an application running inside Kubernetes to access the Neo4j cluster without using a specific server for bootstrapping, you need to install the *neo4j-cluster-headless-service* Helm chart. This will create a K8s Service with a DNS entry that includes all the Neo4j servers. You can use the created DNS entry to bootstrap drivers connecting to the cluster.

The headless service is a Kubernetes term for a service that has no ClusterIP. For more information, see the Kubernetes official documentation.

1. Install the headless service using the release name `headless`, *neo4j/neo4j-cluster-headless-service* Helm chart, and the name of your cluster as a value of the `neo4j.name` parameter.

   Alternatively, you can create a *values.yaml* file with all the configurations for the service. To see what options are configurable on the *neo4j/neo4j-cluster-headless-service* Helm chart, use `helm show values neo4j/neo4j-headless-service`.

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

```
NAME: headless
LAST DEPLOYED: Wed Oct 26 13:11:14 2022
NAMESPACE: neo4j
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Thank you for installing neo4j-cluster-headless-service.

Your release "headless" has been installed in namespace "neo4j".

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

  $ kubectl run --rm -it --namespace "neo4j" --image "neo4j:5.4.0-enterprise" cypher-shell \
     -- cypher-shell -a "neo4j://headless-neo4j.neo4j.svc.cluster.local:7687"

Graphs are everywhere!
```

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

```
export NEO4J_NAME=my-cluster
kubectl get service ${NEO4J_NAME}-headless
```

```
NAME                  TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)                       AGE
my-cluster-headless   ClusterIP   None         <none>        7474/TCP,7473/TCP,7687/TCP    113s
```

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

```
kubectl describe service ${NEO4J_NAME}-headless
```

```
Name:              my-cluster-headless
Namespace:         neo4j
Labels:            app=my-cluster
                   app.kubernetes.io/managed-by=Helm
                   helm.neo4j.com/neo4j.name=my-cluster
Annotations:       cloud.google.com/neg: {"ingress":true}
                   meta.helm.sh/release-name: headless
                   meta.helm.sh/release-namespace: neo4j
Selector:          app=my-cluster,helm.neo4j.com/neo4j.loadbalancer=include
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                None
IPs:               None
Port:              http  7474/TCP
TargetPort:        7474/TCP
Endpoints:         10.24.0.131:7474,10.24.1.3:7474,10.24.1.67:7474
Port:              https  7473/TCP
TargetPort:        7473/TCP
Endpoints:         10.24.0.131:7473,10.24.1.3:7473,10.24.1.67:7473
Port:              tcp-bolt  7687/TCP
TargetPort:        7687/TCP
Endpoints:         10.24.0.131:7687,10.24.1.3:7687,10.24.1.67:7687
Session Affinity:  None
Events:            <none>
```

You should see three “endpoints” for each port in the service — these are the IP addresses of the three Neo4j servers. These endpoints are contacted to bootstrap the drivers used by applications running in Kubernetes. The drivers will use them to obtain the initial routing table.

3. Run `cypher-shell` in another pod and connect to the cluster servers via the headless service:

```
kubectl run --rm -it --namespace "neo4j" --image "neo4j:5.4.0-enterprise"cypher-shell -- cypher-shell
-a \  "neo4j://my-cluster-headless.neo4j.svc.cluster.local:7687" -u neo4j -p "my-password"
```

```
If you don't see a command prompt, try pressing enter.
Connected to Neo4j using Bolt protocol version 5 at neo4j://headless-
neo4j.default.svc.cluster.local:7687 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
```

4. Run the Cypher command `SHOW DATABASES` to verify that all cluster servers are online.

```
SHOW DATABASES;
```

```
+--------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------+
| name     | type       | aliases | access       | address                                  | role
| writer | requestedStatus | currentStatus | statusMessage | default | home  | constituents |
+--------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------+
| "neo4j"  | "standard" | []      | "read-write" | "server-3.neo4j.svc.cluster.local:7687" | "primary"
| TRUE   | "online"        | "online"      | ""              | TRUE    | TRUE  | []             |
| "neo4j"  | "standard" | []      | "read-write" | "server-2.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""              | TRUE    | TRUE  | []             |
| "neo4j"  | "standard" | []      | "read-write" | "server-1.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""              | TRUE    | TRUE  | []             |
| "system" | "system"   | []      | "read-write" | "server-3.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""              | FALSE   | FALSE | []             |
| "system" | "system"   | []      | "read-write" | "server-2.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""              | FALSE   | FALSE | []             |
| "system" | "system"   | []      | "read-write" | "server-1.neo4j.svc.cluster.local:7687" | "primary"
| TRUE   | "online"        | "online"      | ""              | FALSE   | FALSE | []             |
+--------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------+
6 rows
ready to start consuming query after 4 ms, results consumed after another 42 ms
```

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

```
:exit;
```

```
Bye!
Session ended, resume using 'kubectl attach cypher-shell -c cypher-shell -i -t' command when the pod
is running
pod "cypher-shell" deleted
```

Access the Neo4j cluster from outside Kubernetes

By default, server-side routing is used for accessing a Neo4j cluster from outside Kubernetes.

Access the Neo4j cluster using a load balancer and Cypher Shell

A LoadBalancer service is created to access a Neo4j cluster from outside Kubernetes.

1. Check that the LoadBalancer service is available using the `neo4j.name` used for installation:

```
export NEO4J_NAME=my-cluster
kubectl get service ${NEO4J_NAME}-lb-neo4j
```

```
NAME                  TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)
AGE
my-cluster-lb-neo4j   LoadBalancer   10.28.12.119   82.21.42.42
7474:32169/TCP,7473:32145/TCP,7687:32624/TCP   2m1s
```

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

```
kubectl describe service ${NEO4J_NAME}-lb-neo4j
```

```
Name:                     my-cluster-lb-neo4j
Namespace:                neo4j
Labels:                   app=my-cluster
                          app.kubernetes.io/managed-by=Helm
                          helm.neo4j.com/neo4j.name=my-cluster
                          helm.neo4j.com/service=neo4j
Annotations:              cloud.google.com/neg: {"ingress":true}
                          meta.helm.sh/release-name: server-1
                          meta.helm.sh/release-namespace: neo4j
Selector:                 app=my-
cluster,helm.neo4j.com/clustering=true,helm.neo4j.com/neo4j.loadbalancer=include
Type:                     LoadBalancer
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.28.12.119
IPs:                      10.28.12.119
LoadBalancer Ingress:     82.21.42.42
Port:                     http  7474/TCP
TargetPort:               7474/TCP
NodePort:                 http  32169/TCP
Endpoints:                10.24.0.131:7474,10.24.1.3:7474,10.24.1.67:7474
Port:                     https  7473/TCP
TargetPort:               7473/TCP
NodePort:                 https  32145/TCP
Endpoints:                10.24.0.131:7473,10.24.1.3:7473,10.24.1.67:7473
Port:                     tcp-bolt  7687/TCP
TargetPort:               7687/TCP
NodePort:                 tcp-bolt  32624/TCP
Endpoints:                10.24.0.131:7687,10.24.1.3:7687,10.24.1.67:7687
Session Affinity:         None
External Traffic Policy:  Local
HealthCheck NodePort:     30621
Events:
  Type    Reason                Age    From                Message
  ----    ------                ----   ----                -------
  Normal  EnsuringLoadBalancer  3m11s  service-controller  Ensuring load balancer
  Normal  EnsuredLoadBalancer   2m36s  service-controller  Ensured load balancer
```

The load-balancer service can send requests to all cluster servers.

3. Run `cypher-shell` from the local machine and connect to the `LoadBalancer Ingress` address, in the example `82.21.42.42`:

```
./cypher-shell -a neo4j://82.21.42.42 -u neo4j -p my-password
```

```
If you don't see a command prompt, try pressing enter.
Connected to Neo4j using Bolt protocol version 5 at neo4j://82.21.42.42:7687 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
```

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

```
SHOW DATABASES;
```

```
+-----------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------+
| name     | type       | aliases | access       | address                                 | role
| writer | requestedStatus | currentStatus | statusMessage | default | home  | constituents |
+-----------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------+
| "neo4j"  | "standard" | []      | "read-write" | "server-3.neo4j.svc.cluster.local:7687" | "primary"
| TRUE   | "online"        | "online"      | ""            | TRUE    | TRUE  | []           |
| "neo4j"  | "standard" | []      | "read-write" | "server-2.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""            | TRUE    | TRUE  | []           |
| "neo4j"  | "standard" | []      | "read-write" | "server-1.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""            | TRUE    | TRUE  | []           |
| "system" | "system"   | []      | "read-write" | "server-3.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""            | FALSE   | FALSE | []           |
| "system" | "system"   | []      | "read-write" | "server-2.neo4j.svc.cluster.local:7687" | "primary"
| FALSE  | "online"        | "online"      | ""            | FALSE   | FALSE | []           |
| "system" | "system"   | []      | "read-write" | "server-1.neo4j.svc.cluster.local:7687" | "primary"
| TRUE   | "online"        | "online"      | ""            | FALSE   | FALSE | []           |
+-----------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------+

6 rows
ready to start consuming query after 110 ms, results consumed after another 109 ms
```

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

```
SHOW SERVERS;
```

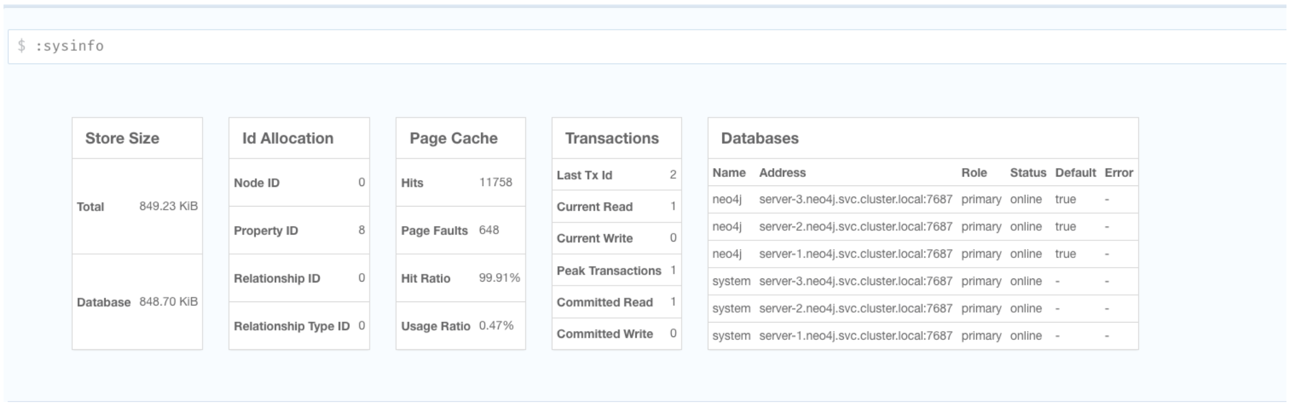
```
+-----------------------------------------------------------------------------------------------------
-----------------------------+
| name                                   | address                                 | state     |
health      | hosting             |
+-----------------------------------------------------------------------------------------------------
-----------------------------+
| "ad5c3cf1-541a-44f8-a19b-28bc36030914" | "server-3.neo4j.svc.cluster.local:7687" | "Enabled" |
"Available" | ["system", "neo4j"] |
| "cbdebc59-64c2-4542-a041-24a1f051e64f" | "server-1.neo4j.svc.cluster.local:7687" | "Enabled" |
"Available" | ["system", "neo4j"] |
| "f37e98a7-15ec-4dc4-a6bf-df9e418a7488" | "server-2.neo4j.svc.cluster.local:7687" | "Enabled" |
"Available" | ["system", "neo4j"] |
+-----------------------------------------------------------------------------------------------------
-----------------------------+

3 rows
ready to start consuming query after 27 ms, results consumed after another 363 ms
```

You can see that the nodes are advertising their internal addresses, but since you connected without using internal addresses, you use server-side routing.

Access the Neo4j cluster using a load balancer and Neo4j Browser

1. Open a web browser and point it to the `LoadBalancer Ingress` address and port `7474`, in this example, http://82.21.42.42:7474/browser.
2. Once connected, verify that all databases are up and running using `:sysinfo` in the Browser Editor:

```
:sysinfo
```



```
$ :sysinfo
```

Store Size

Total	849.23 KiB
Database	848.70 KiB

Id Allocation

Node ID	0
Property ID	8
Relationship ID	0
Relationship Type ID	0

Page Cache

Hits	11758
Page Faults	648
Hit Ratio	99.91%
Usage Ratio	0.47%

Transactions

Last Tx Id	2
Current Read	1
Current Write	0
Peak Transactions	1
Committed Read	1
Committed Write	0

Databases

Name	Address	Role	Status	Default	Error
neo4j	server-3.neo4j.svc.cluster.local:7687	primary	online	true	-
neo4j	server-2.neo4j.svc.cluster.local:7687	primary	online	true	-
neo4j	server-1.neo4j.svc.cluster.local:7687	primary	online	true	-
system	server-3.neo4j.svc.cluster.local:7687	primary	online	-	-
system	server-2.neo4j.svc.cluster.local:7687	primary	online	-	-
system	server-1.neo4j.svc.cluster.local:7687	primary	online	-	-

Uninstall the Neo4j cluster and clean up the created resources

Uninstall all Neo4j Helm deployments

1. Uninstall each of the cluster servers and the services using their Helm release names:

```
helm uninstall server-1 server-2 server-3 headless
```

```
release "server-1" uninstalled
release "server-2" uninstalled
release "server-3" uninstalled
release "headless" uninstalled
```

Fully remove all the data and resources

Uninstalling the Helm releases does not remove the created resources and data. Therefore, after uninstalling the helm deployments, you also have to delete all the data and resources.

1. Delete all persistent volume claims in the `neo4j` namespace:

```
kubectl delete pvc --all --namespace neo4j
```

2. Delete the entire Kubernetes cluster in your cloud provider:

```
gcloud container clusters delete my-neo4j-gke-cluster
```

```
eksctl delete cluster --name=my-neo4j-eks-cluster
```

```
az aks delete --name my-neo4j-aks-cluster --resource-group <MyResourceGroup>
```

Configure a Neo4j Helm deployment

Helm is different from “package managers”, such as `apt`, `yum`, and `npm`, because, in addition to installing applications, Helm allows rich configuration of applications. The customized configuration should be expressed declaratively in a YAML formatted file, and then passed during installation.

For more information, see Helm official documentation.

Create a custom *values.yaml* file

1. Ensure your Neo4j Helm chart repository is up to date and get the latest charts. For more information, see Configure the Neo4j Helm chart repository.
2. To see what options are configurable on the Neo4j helm chart that you want to deploy, use `helm show values` and the Helm chart *neo4j/neo4j*. For example:

```
helm show values neo4j/neo4j
```

```
# Default values for Neo4j.
# This is a YAML-formatted file.

neo4j:
  # Name of your cluster
  name: ""
  # If password is not set or empty a random password will be generated during installation
  password: ""

  # Neo4j Edition to use (community|enterprise)
  edition: "community"

  # Minimum number of machines initially required to form a clustered database. The StatefulSet will
not reach the ready state
  # until at least this many members have discovered each other. The default is 1 (standalone)
  #minimumClusterSize: 1

  # set edition: "enterprise" to use Neo4j Enterprise Edition
  #
  # To use Neo4j Enterprise Edition you must have a Neo4j license agreement.
  #
  # More information is also available at: https://neo4j.com/licensing/
  # Email inquiries can be directed to: licensing@neo4j.com
  #
  # Set acceptLicenseAgreement: "yes" to confirm that you have a Neo4j license agreement.
  acceptLicenseAgreement: "no"
  #
  # set offlineMaintenanceModeEnabled: true to restart the StatefulSet without the Neo4j process
running
  # this can be used to perform tasks that cannot be performed when Neo4j is running such as `neo4j-
admin dump`
  offlineMaintenanceModeEnabled: false
  #
  # set resources for the Neo4j Container. The values set will be used for both "requests" and
"limit".
  resources:
    cpu: "1000m"
    memory: "2Gi"

  #add labels if required
  labels:

# Volumes for Neo4j
volumes:
  data:
    # REQUIRED: specify a volume mode to use for data
    # Valid values are share|selector|defaultStorageClass|volume|volumeClaimTemplate|dynamic
    # To get up-and-running quickly, for development or testing, use "defaultStorageClass" for a
dynamically provisioned volume of the default storage class.
    mode: ""

    # Only used if mode is set to "selector"
    # Will attach to existing volumes that match the selector
    selector:
      storageClassName: "manual"
      accessModes:
        - ReadWriteOnce
      requests:
        storage: 100Gi
      # A helm template to generate a label selector to match existing volumes n.b. both
storageClassName and label selector must match existing volumes
      selectorTemplate:
        matchLabels:
          app: "{{ .Values.neo4j.name }}"
          helm.neo4j.com/volume-role: "data"

    # Only used if mode is set to "defaultStorageClass"
    # Dynamic provisioning using the default storageClass
    defaultStorageClass:
      accessModes:
        - ReadWriteOnce
      requests:
        storage: 10Gi

    # Only used if mode is set to "dynamic"
    # Dynamic provisioning using the provided storageClass
    dynamic:
      storageClassName: "neo4j"
      accessModes:
        - ReadWriteOnce
      requests:
        storage: 100Gi

    # Only used if mode is set to "volume"
    # Provide an explicit volume to use
    volume:
      # If set an init container (running as root) will be added that runs:
      #   `chown -R <securityContext.fsUser>:<securityContext.fsGroup>` AND `chmod -R g+rwx`
      # on the volume. This is useful for some filesystems (e.g. NFS) where Kubernetes fsUser or
fsGroup settings are not respected
      setOwnerAndGroupWritableFilePermissions: false

      # Example (using a specific Persistent Volume Claim)
      # persistentVolumeClaim:
      #   claimName: my-neo4j-pvc

    # Only used if mode is set to "volumeClaimTemplate"
    # Provide an explicit volumeClaimTemplate to use
    volumeClaimTemplate: {}

  # provide a volume to use for backups
  # n.b. backups will be written to /backups on the volume
  # any of the volume modes shown above for data can be used for backups
  backups:
    mode: "share" # share an existing volume (e.g. the data volume)
    share:
      name: "data"

  # provide a volume to use for logs
  # n.b. logs will be written to /logs/$(POD_NAME) on the volume
  # any of the volume modes shown above for data can be used for logs
  logs:
    mode: "share" # share an existing volume (e.g. the data volume)
    share:
      name: "data"

  # provide a volume to use for csv metrics (csv metrics are only available in Neo4j Enterprise
Edition)
  # n.b. metrics will be written to /metrics/$(POD_NAME) on the volume
  # any of the volume modes shown above for data can be used for metrics
  metrics:
    mode: "share" # share an existing volume (e.g. the data volume)
    share:
      name: "data"
  # provide a volume to use for import storage
  # n.b. import will be mounted to /import on the underlying volume
  # any of the volume modes shown above for data can be used for import
  import:
    mode: "share" # share an existing volume (e.g. the data volume)
    share:
      name: "data"

  # provide a volume to use for licenses
  # n.b. licenses will be mounted to /licenses on the underlying volume
  # any of the volume modes shown above for data can be used for licenses
  licenses:
    mode: "share" # share an existing volume (e.g. the data volume)
    share:
      name: "data"

#add additional volumes and their respective mounts
additionalVolumes: []
#  - name: neo4j1-conf
#    emptyDir: {}
additionalVolumeMounts: []
#  - mountPath: "/config/neo4j1.conf"
#    name: neo4j1-conf


#nodeSelector labels
#please ensure the respective labels are present on one of the cluster nodes or else helm charts will
throw an error
nodeSelector: {}
#  "label1: "value1"
#  "label2: "value2"

# Services for Neo4j
services:
  # A ClusterIP service with the same name as the Helm Release name should be used for Neo4j Driver
connections originating inside the
  # Kubernetes cluster.
  default:
    # Annotations for the K8s Service object
    annotations: { }

  # A LoadBalancer Service for external Neo4j driver applications and Neo4j Browser
  neo4j:
    enabled: true

    # Annotations for the K8s Service object
    annotations: { }

    spec:
      # Type of service.
      type: LoadBalancer

      # in most cloud environments LoadBalancer type will receive an ephemeral public IP address
automatically. If you need to specify a static ip here use:
      # loadBalancerIP: ...

    # ports to include in neo4j service
    ports:
      http:
        enabled: true #Set this to false to remove HTTP from this service (this does not affect
whether http is enabled for the neo4j process)
        # uncomment to publish http on port 80 (neo4j default is 7474)
        # port: 80
      https:
        enabled: true #Set this to false to remove HTTPS from this service (this does not affect
whether https is enabled for the neo4j process)
        # uncomment to publish http on port 443 (neo4j default is 7474)
        # port: 443
      bolt:
        enabled: true #Set this to false to remove BOLT from this service (this does not affect
whether https is enabled for the neo4j process)
        # Uncomment to explicitly specify the port to publish Neo4j Bolt (7687 is the default)
        # port: 7687
      backup:
        enabled: false #Set this to true to expose backup port externally (n.b. this could have
security implications. Backup is not authenticated by default)
```

```
        # Uncomment to explicitly specify the port to publish Neo4j Backup (6362 is the default)
        # port: 6362

    selector:
      "helm.neo4j.com/neo4j.loadbalancer": "include"
      # By default the load balancer will match all Neo4j instance types.
      # When Neo4j drivers connect from outside K8s using the load balancer they will not fetch a
routing table.
      # In this case drivers can only use instances included in the load balancer.
      # To only include Neo4j Core instances uncomment the setting below.
      # To only route to Neo4j Read Replicas uncomment the setting and change the value to
"READ_REPLICA"
      # "helm.neo4j.com/clustering": "false"

    #this flag allows you to open internal neo4j ports necessary in multi zone /region neo4j cluster
scenario
    multiCluster: false

  # A service for admin/ops tasks including taking backups
  # This service is available even if the deployment is not "ready"
  admin:
    enabled: true
    # Annotations for the admin service
    annotations: { }
    spec:
      type: ClusterIP
    # n.b. there is no ports object for this service. Ports are autogenerated based on the neo4j
configuration

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


# Neo4j Configuration (yaml format)
config:
  server.config.strict_validation.enabled: "false"
#  dbms.cluster.minimum_initial_system_primaries_count: "3"
  # The amount of memory to use for mapping the store files.
  # The default page cache memory assumes the machine is dedicated to running
  # Neo4j, and is heuristically set to 50% of RAM minus the Java heap size.
  #dbms.memory.pagecache.size: "74m"

  #The number of Cypher query execution plans that are cached.
  #dbms.query_cache_size: "10"

  # Java Heap Size: by default the Java heap size is dynamically calculated based
  # on available system resources. Uncomment these lines to set specific initial
  # and maximum heap size.
  #dbms.memory.heap.initial_size: "317m"
  #dbms.memory.heap.max_size: "317m"

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

# securityContext defines privilege and access control settings for a Pod or Container. Making sure
that we dont run Neo4j as root user.
securityContext:
  runAsNonRoot: true
  runAsUser: 7474
  runAsGroup: 7474
  fsGroup: 7474
  fsGroupChangePolicy: "Always"

# Readiness probes are set to know when a container is ready to be used.
# Because Neo4j uses Java these values are large to distinguish between long Garbage Collection pauses
(which don't require a restart) and an actual failure.
# These values should mark Neo4j as not ready after at most 5 minutes of problems (20 attempts * max
15 seconds between probes)
readinessProbe:
  failureThreshold: 20
```

```yaml
  timeoutSeconds: 10
  periodSeconds: 5

# Liveness probes are set to know when to restart a container.
# Because Neo4j uses Java these values are large to distinguish between long Garbage Collection pauses
(which don't require a restart) and an actual failure.
# These values should trigger a restart after at most 10 minutes of problems (40 attempts * max 15
seconds between probes)
livenessProbe:
  failureThreshold: 40
  timeoutSeconds: 10
  periodSeconds: 5

# Startup probes are used to know when a container application has started.
# If such a probe is configured, it disables liveness and readiness checks until it succeeds
# When restoring Neo4j from a backup it's important that startup probe gives time for Neo4j to recover
and/or upgrade store files
# When using Neo4j clusters it's important that startup probe give the Neo4j cluster time to form
startupProbe:
  failureThreshold: 1000
  periodSeconds: 5

# top level setting called ssl to match the "ssl" from "dbms.ssl.policy"
ssl:
  # setting per "connector" matching neo4j config
  bolt:
    privateKey:
      secretName:  # we set up the template to grab `private.key` from this secret
      subPath:  # we specify the privateKey value name to get from the secret
    publicCertificate:
      secretName:  # we set up the template to grab `public.crt` from this secret
      subPath:  # we specify the publicCertificate value name to get from the secret
    trustedCerts:
      sources: [ ] # a sources array for a projected volume - this allows someone to (relatively)
easily mount multiple public certs from multiple secrets for example.
    revokedCerts:
      sources: [ ]  # a sources array for a projected volume
  https:
    privateKey:
      secretName:
      subPath:
    publicCertificate:
      secretName:
      subPath:
    trustedCerts:
      sources: [ ]
    revokedCerts:
      sources: [ ]

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

# Override image settings in Neo4j pod
image:
  imagePullPolicy: IfNotPresent
  # set a customImage if you want to use your own docker image
#  customImage: eu.gcr.io/neo4j-helm/neo4j:v5

  #imagePullSecrets list
  #  imagePullSecrets:
  #    - "demo"

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

statefulset:
  metadata:
    #Annotations for Neo4j StatefulSet
    annotations:
#      imageregistry: "https://hub.docker.com/"
#      demo: alpha

# additional environment variables for the Neo4j Container
env: {}

# Other K8s configuration to apply to the Neo4j pod
podSpec:

  #Annotations for Neo4j pod
  annotations: {}
#   imageregistry: "https://hub.docker.com/"
#   demo: alpha

  nodeAffinity: {}
#    requiredDuringSchedulingIgnoredDuringExecution:
#      nodeSelectorTerms:
#        - matchExpressions:
#            - key: topology.kubernetes.io/zone
#              operator: In
#              values:
#                - antarctica-east1
#                - antarctica-west1
#    preferredDuringSchedulingIgnoredDuringExecution:
#      - weight: 1
#        preference:
#          matchExpressions:
#            - key: another-node-label-key
#              operator: In
#              values:
#                - another-node-label-value

  # Anti Affinity
  # If set to true then an anti-affinity rule is applied to prevent database pods with the same
`neo4j.name` running on a single Kubernetes node.
  # If set to false then no anti-affinity rules are applied
  # If set to an object then that object is used for the Neo4j podAntiAffinity
  podAntiAffinity: true

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

  #Priority indicates the importance of a Pod relative to other Pods.
  # More Information : https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-
preemption/
  priorityClassName: ""

  #This indicates that the neo4j instance be included to the loadbalancer. Can be set to exclude to
not add the stateful set to loadbalancer
  loadbalancer: "include"

  # Name of service account to use for the Neo4j Pod (optional)
  # this is useful if you want to use Workload Identity to grant permissions to access cloud resources
e.g. cloud object storage (AWS S3 etc.)
  serviceAccountName: ""

  # How long the Neo4j pod is permitted to keep running after it has been signalled by Kubernetes to
stop. Once this timeout elapses the Neo4j process is forcibly terminated.
  # A large value is used because Neo4j takes time to flush in-memory data to disk on shutdown.
  terminationGracePeriodSeconds: 3600

  # initContainers for the Neo4j pod
  initContainers: [ ]

  # additional runtime containers for the Neo4j pod
  containers: [ ]
```

```
# print the neo4j user password set during install to the `helm install` log
logInitialPassword: true

# Jvm configuration for Neo4j
jvm:
  # If true any additional arguments are added after the Neo4j default jvm arguments.
  # If false Neo4j default jvm arguments are not used.
  useNeo4jDefaultJvmArguments: true
  # additionalJvmArguments is a list of strings. Each jvm argument should be a separate element:
  additionalJvmArguments: []
  # - "-XX:+HeapDumpOnOutOfMemoryError"
  # - "-XX:HeapDumpPath=/logs/neo4j.hprof"
  # - "-XX:MaxMetaspaceSize=180m"
  # - "-XX:ReservedCodeCacheSize=40m"

logging:
  serverLogsXml: |-
#     <?xml version="1.0" encoding="UTF-8"?>
#     <!-- Example JSON logging configuration -->
#     <Configuration status="ERROR" monitorInterval="30" packages="org.neo4j.logging.log4j">
#         <Appenders>
#             <!-- Default debug.log, please keep -->
#             <RollingRandomAccessFile name="DebugLog"
fileName="${config:server.directories.logs}/debug.log"
#                                      filePattern="$${config:server.directories.logs}/debug.log.%02i">
#                 <JsonTemplateLayout
eventTemplateUri="classpath:org/neo4j/logging/StructuredLayoutWithMessage.json"/>
#                 <Policies>
#                     <SizeBasedTriggeringPolicy size="20 MB"/>
#                 </Policies>
#                 <DefaultRolloverStrategy fileIndex="min" max="7"/>
#             </RollingRandomAccessFile>
#
#             <RollingRandomAccessFile name="HttpLog"
fileName="${config:server.directories.logs}/http.log"
#                                      filePattern="$${config:server.directories.logs}/http.log.%02i">
#                 <JsonTemplateLayout
eventTemplateUri="classpath:org/neo4j/logging/StructuredLayoutWithMessage.json"/>
#                 <Policies>
#                     <SizeBasedTriggeringPolicy size="20 MB"/>
#                 </Policies>
#                 <DefaultRolloverStrategy fileIndex="min" max="5"/>
#             </RollingRandomAccessFile>
#
#             <RollingRandomAccessFile name="QueryLog"
fileName="${config:server.directories.logs}/query.log"
#                                      filePattern="$${config:server.directories.logs}/query.log.%02i">
#                 <JsonTemplateLayout
eventTemplateUri="classpath:org/neo4j/logging/QueryLogJsonLayout.json"/>
#                 <Policies>
#                     <SizeBasedTriggeringPolicy size="20 MB"/>
#                 </Policies>
#                 <DefaultRolloverStrategy fileIndex="min" max="7"/>
#             </RollingRandomAccessFile>
#
#             <RollingRandomAccessFile name="SecurityLog"
fileName="${config:server.directories.logs}/security.log"
#
filePattern="$${config:server.directories.logs}/security.log.%02i">
#                 <JsonTemplateLayout
eventTemplateUri="classpath:org/neo4j/logging/StructuredLayoutWithMessage.json"/>
#                 <Policies>
#                     <SizeBasedTriggeringPolicy size="20 MB"/>
#                 </Policies>
#                 <DefaultRolloverStrategy fileIndex="min" max="7"/>
#             </RollingRandomAccessFile>
#         </Appenders>
#
#         <Loggers>
#             <!-- Log levels. One of DEBUG, INFO, WARN, ERROR or OFF -->
#
#             <!-- The debug log is used as the root logger to catch everything -->
#             <Root level="INFO">
#                 <AppenderRef ref="DebugLog"/> <!-- Keep this -->
#             </Root>
#             <!-- The query log, must be named "QueryLogger" -->
#             <Logger name="QueryLogger" level="INFO" additivity="false">
#                <AppenderRef ref="QueryLog"/>
#            </Logger>
#            <!-- The http request log, must be named "HttpLogger" -->
#            <Logger name="HttpLogger" level="INFO" additivity="false">
#                <AppenderRef ref="HttpLog"/>
#            </Logger>
#            <!-- The security log, must be named "SecurityLogger" -->
#            <Logger name="SecurityLogger" level="INFO" additivity="false">
#                <AppenderRef ref="SecurityLog"/>
#            </Logger>
#        </Loggers>
#    </Configuration>
  userLogsXml: |-
#    <?xml version="1.0" encoding="UTF-8"?>
#    <!-- Example JSON logging configuration -->
#    <Configuration status="ERROR" monitorInterval="30" packages="org.neo4j.logging.log4j">
#    <Appenders>
#        <RollingRandomAccessFile name="Neo4jLog"
fileName="${config:server.directories.logs}/neo4j.log"
#                                 filePattern="$${config:server.directories.logs}/neo4j.log.%02i">
#            <JsonTemplateLayout
eventTemplateUri="classpath:org/neo4j/logging/StructuredLayoutWithMessage.json"/>
#            <Policies>
#                <SizeBasedTriggeringPolicy size="20 MB"/>
#            </Policies>
#            <DefaultRolloverStrategy fileIndex="min" max="7"/>
#        </RollingRandomAccessFile>
#        <!-- Only used by "neo4j console", will be ignored otherwise -->
#        <Console name="ConsoleAppender" target="SYSTEM_OUT">
#            <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSSZ}{GMT+0} %-5p %m%n"/>
#        </Console>
#    </Appenders>
#    <Loggers>
#        <!-- Log level for the neo4j log. One of DEBUG, INFO, WARN, ERROR or OFF -->
#        <Root level="INFO">
#            <AppenderRef ref="Neo4jLog"/>
#            <AppenderRef ref="ConsoleAppender"/>
#        </Root>
#    </Loggers>
#    </Configuration>
```

3. Pass the *neo4j-values.yaml* file during installation. The `neo4j.name` parameter is mandatory and can be supplied either in `neo4j-values.yaml` or by using the `--set` argument.

```
helm install <release-name> neo4j/neo4j --set "neo4j.name=my-neo4j-db" -f neo4j-values.yaml
```

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

Some examples of possible K8s configurations

- Configure (or disable completely) the Kubernetes LoadBalancer that exposes Neo4j outside the Kubernetes cluster by modifying the `externalService` object in the *values.yml* file.
- Set the `securityContext` used by Neo4j Pods by modifying the `securityContext` object in the *values.yml* file.
- Configure manual persistent volume provisioning or set the `StorageClass` to be used as the Neo4j persistent storage.

Some examples of possible Neo4j configurations

- All Neo4j configuration (*neo4j.conf*) settings can be set directly on the `config` object in the *values.yaml* file.
- Neo4j can be configured to use SSL certificates contained in Kubernetes Secrets by modifying

the ssl object in the values file.

Set neo4j.name parameter

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

Set neo4j.minimumClusterSize parameter

By default, servers in a cluster can host primary and secondary databases. See the clustering documentation for more details.
neo4j.minimumClusterSize is set to 1 by default, which means the server starts without waiting for the other servers. When installing a cluster, you should set neo4j.minimumClusterSize to the number of desired members in the cluster. If you later decide to add an extra cluster server in excess of neo4j.minimumClusterSize, you need to manually enable it using the Cypher command ENABLE SERVER. For more information on enabling new servers, see Add a server to the cluster.

Set Neo4j configuration

The Neo4j Helm chart does not use a neo4j.conf file. Instead, the Neo4j configuration is set in the Helm deployment's *values.yaml* file under the config object.

The config object should contain a string map of *neo4j.conf* setting name to value. For example, this config object configures the Neo4j metrics:

```
# Neo4j configuration (yaml format)
config:
  server.metrics.enabled: "true"
  server.metrics.csv.interval: "10s"
  server.metrics.csv.rotation.keep_number: "2"
  server.metrics.csv.rotation.compression: "NONE"
```

All Neo4j config values must be YAML strings. It is important to put quotes around the values, such as "true", "false", and "2", so that they are handled correctly as strings.

All *neo4j.conf* settings are supported except for server.jvm.additional. Additional JVM settings can be set on the jvm object in the Helm deployment *values.yaml* file, as shown in the example:

```
# Jvm configuration for Neo4j
jvm:
  additionalJvmArguments:
  - "-XX:+HeapDumpOnOutOfMemoryError"
  - "-XX:HeapDumpPath=/logs/neo4j.hprof"
```

To find out more about configuring Neo4j and the *neo4j.conf* file, see Configuration and The neo4j.conf file.

Set an initial password

You can set an initial password for accessing Neo4j in the *values.yaml* file. If no initial password is set, the Neo4j helm chart will automatically generate one. In cluster deployments, the same password must be set for all cluster members.

```
neo4j:
 # If not set or empty a random password will be generated
 password: ""
```

The password will be printed out in the Helm install output, unless `--set logInitialPassword=false` is used.

The initial Neo4j password is stored in a *Kubernetes Secret*. The password can be extracted from the *Secret* using this command:

```
kubectl get secret <release-name>-auth -oyaml | yq -r '.data.NEO4J_AUTH' | base64 -d
```

> To change the initial password, follow the steps in Operations - Reset the Neo4j user password.
>
> Once you change the password in Neo4j, the password stored in *Kubernetes Secrets* will still exist but will no longer be valid.

Configure SSL

The Neo4j SSL Framework can be used with Neo4j Helm charts. You can specify SSL policy objects for `bolt`, `https`, `cluster`, and `backup`. SSL public certificates and private keys to use with a Neo4j Helm deployment must be stored in *Kubernetes Secrets*.

To enable Neo4j SSL policies, configure the `ssl.<policy name>` object in the Neo4j Helm deployment's *values.yaml* file to reference the *Kubernetes Secrets* containing the SSL certificates and keys to use. This example shows how to configure the `bolt` SSL policy:

```
ssl:
 bolt:
   privateKey:
     secretName: bolt-cert
     subPath: private.key
   publicCertificate:
     secretName: bolt-cert
     subPath: public.crt
```

When a private key is specified in the *values.yaml* file, the Neo4j `ssl` policy is enabled automatically. To disable a policy, add `dbms.ssl.policy.{{ $name }}.enabled: "false"` to the `config` object.

> Unencrypted `http` is not disabled automatically when `https` is enabled. If `https` is enabled, add `server.http.enabled: "false"` to the `config` object to disable `http`.

For more information on configuring SSL policies, see SSL Framework configuration.

The following examples show how to deploy a Neo4j cluster with configured SSL policies.

Create a self-signed certificate

If you do not have a self-signed certificate to use, follow the steps to create one:

1. Create a new folder for the self-signed certificate. This example uses the */neo4j-ssl* folder.

```
mkdir neo4j-ssl
cd neo4j-ssl
```

2. Create the `private.key` and `public.crt` for the self-signed certificate by using the `openssl` command and passing all the values in the `subj` argument:

```
openssl req -newkey rsa:2048 -nodes -keyout private.key -x509 -days 365 -out public.crt -subj
"/C=GB/ST=London/L=London/O=Neo4j/OU=IT Department"
```

3. Verify that the *private.key* and *public.crt* files are created:

```
ls -lst
```

Example output

```
-rw-r--r--  1 user  staff  1679  28 Dec 15:00 private.key
-rw-r--r--  1 user  staff  1679  28 Dec 15:00 public.crt
```

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

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

Configure an SSL policy using a `tls` Kubernetes secret

This example shows how to configure an SSL policy for intra-cluster communication using a self-signed certificate stored in a `tls` Kubernetes secret.

1. Create a Kubernetes TSL secret using the *public.crt* and *private.key* files:

You must have a Kubernetes cluster running and the `kubectl` command installed. For more information, see Prerequisites.

a. To create a TLS secret, use the `tls` option and a secret name, e.g., `neo4j-tls`:

```
kubectl create secret tls neo4j-tls --cert=/path/to/neo4j-ssl/public.crt --key=/path/to/neo4j-
ssl/private.key
```

b. Verify that the secret is created:

```
kubectl get secret
```

Example output

```
NAME                   TYPE                                  DATA   AGE
neo4j-tls              kubernetes.io/tls                     2      4s
```

c. Verify that the secret contains the *public.crt* and *private.key* files:

```
kubectl get secret neo4j-tls -o yaml
```

Example output

```
apiVersion: v1
data:
  tls.crt:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURLakNDQWhJQ0NRRGRYYVg1Y29mczdEQU5CZ2txaGtpRzl3MEJBUXNGQU
RCWE1Rc3dDUVlEVlFRR0V3SkgKUWpFUE1BMEdBMVVFQ0F3R1RHOXVaRzl1TVE4d0RRWURWUVFIREFaTWIyNWtiMjR4RGpBTUJn
TlZCQW9NQlU1bApielJxTVJZd0ZBWURWUVFMREExSlZDQkVaWEJoY25SdFpXNTBNQjRYRFRJeU1USXlPREl4TURjeU5sb1hEVE
l6Ck1USXlPREl4TURjeU5sb3dWekVMTUFrR0ExVUVCaE1DUjBJeER6QU5CZ05WQkFnTUJreHZibVJ2YmpFUE1BMEcKQTFVRUJ3
d0dURzl1Wkc5dU1RNHdEQVlEVlFRS0RBVk9aVzgwYWpFV01CUUdBMVVFQ3d3TlNWUWdSR1Z3WVhKMApiV1Z1ZERDQ0FTSXdEUV
lKS29aSWh2Y05BUUVCQlFBRGdnRVBBRENDQVFvQ2dnRUJBTDBRc0c2Ukwrd3hxZSt3CjJGSWljZldVaUFtdmNqeVdlS0lKaThu
T2tBSGIvSTYzUUU2L3ZpR3RNeEI3S28xdUJLNlVPZXBaeU91UzE2bUMKaitpMDAwbmFnWkR3RGNyRXd3UUE1cTBGMC90VXB5UH
BaL1p3clhEaGFDOXhzVnFnVms0TXl5aUtTNzRIOUc2UgprUUV4dHBaNFArcTlaRHVFVk1KVGVaL2pQNGZoTkg2MUpSTVdORTJ3
NjNUWkx2ZGMyUitXL2U5N3h2TGQ5Y0FnCjlqTm9FMHo5UHRmczB2L2lyUGhuUHpzWHQ5bzE0MWlnOVFZNjNtMzBxQ0NaYnpMRl
R6WFgvdTUvTSsycFB3WXoKcUNOTUZYYW1ITlAxdlRPWFlRTG1iYW1JdVplYnVPNEVlUHZ6WUVXSmEyUi9oTmhtUDNvM2tRVFAz
dmF1UEFjZQpSQlJZS09NQ0F3RUFBVEFOQmdrcWhraUc5dzBCQVFzRkFBT0NBUUVBWVh6SkIzNU55cExxDUEdvQXVJdGt5dHBFCk
1ZeSs5YnlrV3BEVnhRaXZLUHQ3d1NUaWZjNU1QdW5NUy9xYmxnaWprWm5IaWVLeEdoU3lQL283QndtMzJDSnAKQUFsSjJ3RjhI
VXVSSGpYcUU5dkNQeFdtVlVJS2ExOWN5V0tUYWhySWU1eWZkQWNkbUJmRzJNWnY0dEdFeWxsUgo0Vk81STdRNjVWZDlGQnB0U3
JjS3R1WUtBUzg2RTBHZmlmMWxCakdUZTFZbkhvK1RZTVpoVEUvN3RlNHZ1M251CjA4Y1BmbS9RYThSNFBXZDZNbXVDaTJYcDdu
WVlEMmp3WklCSENtMUU3U1RrdS9JRk5kOWFWRW91VG5KR1pCWFcKeWVzWG9OMXhOb3kvMXZFdElhV2xXZW1GcGo4clJ6VGJQek
Q1TEpiNDBSRFVOTXN3NytLUXczV3BBMjVKUHc9PQotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
  tls.key:
LS0tLS1CRUdJTiBQUklWQVRFIEtFWS0tLS0tCk1JSUV2QUlCQURBTkJna3Foa2lHOXcwQkFRRUZBQVNDQktZd2dnU2lBZ0VBQW
9JQkFRQzlFTEJ1a1Mvc01hbnYKc05oU0luSDFsSWdKcjNJOGxuaWlDWXZKenBBQjIveU90MEJPdjc0aHJUTVFleXFOYmdTdWxE
bnFXY2pya3RlcApnby9vdE5OSjJvR1E4QTNLeE1NRUFPYXRCZFA3VktjajZXZjJjSzF3NFdndmNiRmFvRlpPRE1zb2lrdStCL1
J1CmtaRUJNYmFXZUQvcXZXUTdoRlRDVTNtZjR6K0g0VFIrdFNVVEZqUk5zT3QwMlM3M1hOa2ZsdjN2ZThieTNmWEEKSVBZemFC
Tk0vVDdYN05MLzRxejRaejg3RjdmYU5lTllvUFVHT3Q1dDlLZ2dtVzh5eFU4MTEvN3VmelB0cVQ4RwpNNmdqVEJWMnBoelQ5Yj
B6bDJFQzVtMnBpTG1YbTdqdUJIajc4MkJGaVd0a2Y0VFlaajk2TjVFRXo5NzJyandICkhrUVVXQ2pqQWdNQkFBRUNnZ0VBTmF6
OVNnYXlJazVmUG90b2Zya0V2WUh6dFR3NEpIZGJ2RFVWbUsrcU5yenIKME9DNXd5R3dxd0x2RW1qRlJlM01Lbnd1alJmOGNOVD
VvVWhON3ZVWFgwcEhxb3hjZmdxcWl3SnVld1RDa0FJUwppYUdFUUhUdzZMRTEwUEpvTmFCN29DRUZ0SGErMWk2UCtLd2ZETVcr
WHEyNUI3M0pMUlIrczhUYkxNZHBpL3VvCjRmTFNJV0xDV09MZThUTlU0ck5vVDQ0enY0eUhUOXAyV3liSUNrL3F5bVV3bTlhRH
FnYzRJRzk4YXVVNG5JYVQKenk2T3NBODdONW9FME0rcHlUdEFJcmxRZFBXUzBBZ28xZUJCcWplL1I3MTI4TmdHVzhOTzVMWDBt
Nit2YzhyVgpaTHh0N1d0NThucXR2WlI3QTF5SU9lbWtocHl2Q3hrNVRxSmZQRlJxRVFLQmdRRGlOL3NBZncrZ1dvTFpLbTNyCm
50WVkyRW9TOTBkQ0wwd293SGFGa0lsL2hIWXduQi9qaWlnMU5ZbEhDNzNPVDdDc2UycS9tS0xhMzZBVHlpTHcKZjN1T0J3NmNF
Z2RJZlU5aDBtZjJCNFZXdEVEeDJMSU94MEtZU2VrYldTTVZQZ2w2SkhNb3hLdjNMbEx5R1RiMApZQmtKVmpRdkVLS1dpa1FLMU
dPYnZtdzFWUUtCZ1FEVjlJLzc5WFJuN1EzZ3M4Z2JqZWhDejRqNHdtNWdpNFM2CkVsVzVJWkFidDh3QWZPdVIzUm4wQW41NFl0
ZW1HUk1seDF0L1ZUM1IzK0J5bmVTZEgzbUJ6eVJEQysvRGhBTlYKNVZPckk5SFhnVTRMSElVMmNwVVZxYVo3N1J1b2JnTmlDen
BmOVZPVkNadzdmQzRPYkFqcTMzQ3RtT2taR0hRbAo2dkJtNm1ubFZ3S0JnQ2FnOW95TUplZjA3TGtXcExxTQ1ovN1FHRDRLMmJF
MGtHVzVEOFFZL1ZHNEZkS1JKbVRkCmQ2WTJZUjJ2cEpheFJ2TDlGQ3BwYncyKzkvL0pHWlJGd0p4dEdoS09oWTNjVUF6ZE9BRn
NJVm0vNkFNa1JLdC8KWFNEU0ppc1VXb2hMRXFVM3lpNWcveGh6WVppVHM2MmhKMFZQNGhOVFhPQWw5aDUvVEE4UlFqc05Bb0dB
Tm84Twp5R2xuTGJrOWVMZGZwK2NmK3ltQS9DNVloellNdW9aQ1pkc3hMR0JLSFRXOXZJeHRPZFFJL0JuNGM5cWhEMWt1CjgrR0
F5aXdVeUNXTFRxWGdEa0lNTlN5dUQyVnlsRXpPY1MzSkxQTkVPNEVpVnlnUTdGMCtud3R2cWh1anNUUzcKeGd5Qks5Z3ZodHU3
d3VHNXhHc0dDTDZkY2xEU0RYbERwSHJTVmpFQ2dZQWx0STNjMzJxaG5KU2xHSGhjdW1wRwpReGpvYnJBUUxUa3dyOWk2TkNuS0
EyNVR1SVFXa1NiN2JUWWtuUi80WDhOT2w2U2EvYm9QK2dncWNJM0haSk05CkxJRnpPUTFWT1luQ2ZYZVd0SmlHQklwUExadFdo
bnA3NGVhdmJKYW9udlhVVGNZcm5qcytIWGhpaFhjOUhENWsKeEJEaWJKYUlEbXg2T1FpVWI2RndJZz09Ci0tLS0tRU5EIFBSSV
ZBVEUgS0VZLS0tLS0K
kind: Secret
metadata:
  creationTimestamp: "2023-01-04T13:53:14Z"
  managedFields:
  - apiVersion: v1
    fieldsType: FieldsV1
    fieldsV1:
      f:data:
        .: {}
        f:tls.crt: {}
        f:tls.key: {}
      f:type: {}
    manager: kubectl
    operation: Update
    time: "2023-01-04T13:53:14Z"
  name: neo4j-tls
  namespace: neo4j
  resourceVersion: "212009"
  uid: b1be45dd-4cbe-41c9-a6e5-c814c5e39c25
type: kubernetes.io/tls
```

2. Configure `ssl` object in the *ssl-values.yaml* file using the created secret:

```
ssl:
# setting per "connector" matching neo4j config
  bolt:
    privateKey:
      secretName: neo4j-tls
      subPath: tls.key
    publicCertificate:
      secretName: neo4j-tls
      subPath: tls.crt
  https:
    privateKey:
      secretName: neo4j-tls
      subPath: tls.key
    publicCertificate:
      secretName: neo4j-tls
      subPath: tls.crt
    trustedCerts:
      sources:
      - secret:
          name: neo4j-tls
          items:
          - key: tls.crt
            path: public.crt
  cluster:
    privateKey:
      secretName: neo4j-tls
      subPath: tls.key
    publicCertificate:
      secretName: neo4j-tls
      subPath: tls.crt
    trustedCerts:
      sources:
      - secret:
          name: neo4j-tls
          items:
          - key: tls.crt
            path: public.crt
    revokedCerts:
      sources: [ ]
```

Now you are ready to deploy the Neo4j cluster using the configured *ssl-values.yaml* file and the Neo4j Helm charts.

Configure an SSL policy using a `generic` Kubernetes secret

This example shows how to configure an SSL policy for intra-cluster communication using a self-signed certificate stored in a `generic` Kubernetes secret.

1. Create a Kubernetes `generic` secret using the *public.crt* and *private.key* files:

   You must have a Kubernetes cluster running and the `kubectl` command installed. For more information, see Prerequisites.

   a. Get the Base64-encoded value of your *public.crt* and *private.key*:

   ```
   cat public.crt| base64
   ```

   ```
   cat private.key| base64
   ```

   b. Using the Base64-encoded values of your *public.crt* and *private.key*, create a *secret.yaml* file:

```
apiVersion: v1
data:
  public.crt:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURLakNDQWhJQ0NRRGRYYVg1Y29mczdEQU5CZ2txaGtpRzl3MEJBUXNGQU
RCWE1Rc3dDUVlEVlFRR0V3SkgKUWpFUE1BMEdBMVVFQ0F3R1RHOXVaRzl1TVE4d0RRWURWUVFIREFaTWIyNWtiMjR4RGpBTUJn
TlZCQW9NQlU1bApielJxTVJZd0ZBWURWUVFMREExSlZDQkVaWEJoY25SdFpXNTBNQjRYRFRJeU1USXlPREl4TURjeU5sb1hEVE
l6Ck1USXlPREl4TURjeU5sb3dWekVMTUFrR0ExVUVCaE1DUjBJeER6QU5CZ05WQkFnTUJreHZibVJ2YmpFUE1BMEcKQTFVRUJ3
d0dURzl1Wkc5dU1RNHdEQVlEVlFRS0RBVk9aVzgwYWpFV01CUUdBMVVFQ3d3TlNWUWdSR1Z3WVhKMApiV1Z1ZERDQ0FTSXdEUV
lKS29aSWh2Y05BUUVCQlFBRGdnRVBBRENDQVFvQ2dnRUJBTDBRc0c2Ukwrd3hxZSt3CjJGSWljZldVaUFtdmNqeVdlS0lKaThu
T2tBSGIvSTYzUUU2L3ZpR3RNeEI3S28xdUJLNlVPZXBaeU91UzE2bUMKaitpMDAwbmFnWkR3RGNyRXd3UUE1cTBGMC90VXB5UH
BaL1p3clhEaGFDOXhzVnFnVms0TXl5aUtTNzRIOUc2UgprUUV4dHBaNFArcTlaRHVFVk1KVGVaL2pQNGZoTkg2MUpSTVdORTJ3
NjNUWkx2ZGMyUitXL2U5N3h2TGQ5Y0FnCjlqTm9FMHo5UHRmczB2L2lyUGhuUHpzWHQ5bzE0MWlnOVFZNjNtMzBxQ0NaYnpMRl
R6WFgvdTUvTSsycFB3WXoKcUNOTUZYYW1ITlAxdlRPWFlRTG1iYW1JdVplYnVPNEVlUHZ6WUVXSmEyUi9oTmhtUDNvM2tRVFAz
dmF1UEFjZQpSQlJZS09NQ0F3RUFBVEFOQmdrcWhraUc5dzBCQVFzRkFBT0NBUUVBWVh6SkIzNU55cExxDUEdvQXVJdGt5dHBFCk
1ZeSs5YnlrV3BEVnhRaXZLUHQ3d1NUaWZjNU1QdW5NUy9xYmxnaWprWm5IaWVLeEdoU3lQL283QndtMzJDSnAKQUFsSjJ3RjhI
VXVSSGpYcUU5dkNQeFdtVlVJS2ExOWN5V0tUYWhySWU1eWZkQWNkbUJmRzJNWnY0dEdFeWxsUgo0Vk81STdRNjVWZDlGQnB0U3
JjS3R1WUtBUzg2RTBHZmlmMWxCakdUZTFZbkhvK1RZTVpoVEUvN3RlNHZ1M251CjA4Y1BmbS9RYThSNFBXZDZNbXVDaTJYcDdu
WVlEMmp3WklCSENtMUU3U1RrdS9JRk5kOWFWRW91VG5KR1pCWFcKeWVzWG9OMXhOb3kvMXZFdElhV2xXZW1GcGo4clJ6VGJQek
Q1TEpiNDBSRFVOTXN3NytLUXczV3BBMjVKUHc9PQotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
  private.key:
LS0tLS1CRUdJTiBQUklWQVRFIEtFWS0tLS0tCk1JSUV2QUlCQURBTkJna3Foa2lHOXcwQkFRRUZBQVNDQktZd2dnU2lBZ0VBQW
9JQkFRQzlFTEJ1a1Mvc01hbnYKc05oU0luSDFsSWdKcjNJOGxuaWlDWXZKenBBQjIveU90MEJPdjc0aHJUTVFleXFOYmdTdWxE
bnFXY2pya3RlcApnby9vdE5OSjJvR1E4QTNLeE1NRUFPYXRCZFA3VktjajZXZjJjSzF3NFdndmNiRmFvRlpPRE1zb2lrdStCL1
J1CmtaRUJNYmFXZUQvcXZXUTdoRlRDVTNtZjR6K0g0VFIrdFNVVEZqUk5zT3QwMlM3M1hOa2ZsdjN2ZThieTNmWEEKSVBZemFC
Tk0vVDdYN05MLzRxejRaejg3RjdmYU5lTllvUFVHT3Q1dDlLZ2dtVzh5eFU4MTEvN3VmelB0cVQ4RwpNNmdqVEJWMnBoelQ5Yj
B6bDJFQzVtMnBpTG1YbTdqdUJIajc4MkJGaVd0a2Y0VFlaajk2TjVFRXo5NzJyandICkhrUVVXQ2pqQWdNQkFBRUNnZ0VBTmF6
OVNnYXlJazVmUG90b2Zya0V2WUh6dFR3NEpIZGJ2RFVWbUsrcU5yenIKME9DNXd5R3dxd0x2RW1qRlJ1M01Lbnd1alJmOGNOVD
VvVWhON3ZVWFgwcEhxb3hjZmdxcWl3SnVld1RDa0FJUwppYUdFUUhUdzZMRTEwUEpvTmFCN29DRUZ0SGErMWk2UCtLd2ZETVcr
WHEyNUI3M0pMUlIrczhUYkxNZHBpL3VvCjRmTFNJV0xDV09MZThUTlU0ck5vVDQ0enY0eUhUOXAyV3liSUNrL3F5bVV3bTlhRH
FnYzRJRzk4YXVVNG5JYVQKenk2T3NBODdONW9FME0rcHlUdEFJcmxRZFBXUzBBZ28xZUJCcWplL1I3MTI4TmdHVzhOTzVMWDBt
Nit2YzhyVgpaTHh0N1d0NThucXR2WlI3QTF5SU9lbWtocHl2Q3hrNVRxSmZQRlJxRVFLQmdRRGlOL3NBZncrZ1dvTFpLbTNyCm
50WVkyRW9TOTBkQ0wwd293SGFGa0lsL2hIWXduQi9qaWlnMU5ZbEhDNzNPVDdDc2UycS9tS0xhMzZBVHlpTHcKZjN1T0J3NmNF
Z2RJZlU5aDBtZjJCNFZXdEVEeDJMSU94MEtZU2VrYldTTVZQZ2w2SkhNb3hLdjNMbEx5R1RiMApZQmtKVmpRdkVLS1dpa1FLMU
dPYnZtdzFWUUtCZ1FFVjlJLzc5WFJuN1EzZ3M4Z2JqZWhDejRqNHdtNWdpNFM2CkVsVzVJWkFidDh3QWZPdVIzUm4wQW41NFl0
ZW1HUk1seDF0L1ZUM1IzK0J5bmVTZEgzbUJ6eVJEQysvRGhBTlYKNVZPckk5SFhnVTRMSElVMmNwVVZxYVo3N1J1b2JnTmlDen
BmOVZPVkNadzdmQzRPYkFqcTMzQ3RtT2taR0hRbAo2dkJtNm1ubFZ3S0JnQ2FnOW95TUplZjA3TGtXcExxTQ1ovN1FHRDRLMmJF
MGtHVzVEOFFZL1ZHNEZkS1JKbVRkCmQ2WTJZUjJ2cEpheFJ2TDlGQ3BwYncyKzkvL0pHWlJGd0p4dEdoS09oWTNjVUF6ZE9BRn
NJVm0vNkFNa1JLdC8KWFNEU0ppc1VXb2hMRXFVM1lpNWcveGh6WVppVHM2MmhKMFZQNGhOVFhPQWw5aDUvVEE4UlFqc05Bb0dB
Tm84Twp5R2xuTGJrOWVMZGZwK2NmK3ltQS9DNVloellNdW9aQ1pkc3hMR0JLSFRXOXZJeHRPZFFJL0JuNGM5cWhEMWt1CjgrR0
F5aXdVeUNXTFRxWGdEa0lNTlN5dUQyVnlsRXpPY1MzSkxQTkVPNEVpVnlnUTdGMCtud3R2cWh1anNUUzcKeGd5Qks5Z3ZodHU3
d3VHNXhHc0dDTDZkY2xEU0RYbERwSHJTVmpFQ2dZQWx0STNjMzJxaG5KU2xHSGhjdW1wRwpReGpvYnJBUUxUa3dyOWk2TkNuS0
EyNVR1SVFXa1NiN2JUWWtuUi80WDhOT2w2U2EvYm9QK2dncWNJM0haSk05CkxJRnpPUTFWT1luQ2ZYZVd0SmlHQklwUExadFdo
bnA3NGVhdmJKYW9udlhVVGNZcm5qcytIWGhpaFhjOUhENWsKeEJEaWJKYUlEbXg2T1FpVWI2RndJZz09Ci0tLS0tRU5EIFBSSV
ZBVEUgS0VZLS0tLS0K
kind: Secret
metadata:
  name: neo4j-tls
  namespace: neo4j
type: Opaque
```

c. Create the generic secret using the `kubectl create` command and the *secret.yaml* file:

```
kubectl create -f /path/to/secret.yaml
```

Example output

```
secret/neo4j-tls created
```

d. Verify that the secret is created:

```
kubectl get secret
```

Example output

```
NAME        TYPE      DATA   AGE
neo4j-tls   Opaque    2      85s
```

2. Configure the `ssl` object in the *ssl-values.yaml* file using the created secret:

```
ssl:
# setting per "connector" matching neo4j config
  bolt:
    privateKey:
      secretName: neo4j-tls
      subPath: private.key
    publicCertificate:
      secretName: neo4j-tls
      subPath: public.crt
  https:
    privateKey:
      secretName: neo4j-tls
      subPath: private.key
    publicCertificate:
      secretName: neo4j-tls
      subPath: public.crt
    trustedCerts:
      sources:
      - secret:
          name: neo4j-tls
          items:
          - key: public.crt
            path: public.crt
  cluster:
    privateKey:
      secretName: neo4j-tls
      subPath: private.key
    publicCertificate:
      secretName: neo4j-tls
      subPath: public.crt
    trustedCerts:
      sources:
      - secret:
          name: neo4j-tls
          items:
          - key: public.crt
            path: public.crt
    revokedCerts:
      sources: [ ]
```

Now you are ready to deploy the Neo4j cluster using the *ssl-values.yaml* file and the Neo4j Helm charts.

Deploy a Neo4j cluster with SSL certificates

Deploy a Neo4j cluster using the Neo4j Helm chart and the *ssl-values.yaml* file.

1. Install server-1:

```
helm install server-1 neo4j/neo4j --namespace neo4j --set neo4j.acceptLicenseAgreement=yes --set
neo4j.password=my-password --set neo4j.name="my-cluster" --set neo4j.minimumClusterSize=3 --set
neo4j.edition="enterprise" --set volumes.data.mode=defaultStorageClass -f ~/Documents/neo4j-ssl/ssl-
values.yaml
```

2. Repeat the command from the previous step for `server-2` and `server-3`.

3. Verify that the Neo4j cluster is running:

```
kubectl get pods
```

Example output

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

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

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

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

Example output

```
total 12
4 drwxr-xr-x 2 root root 4096 Jan  4 13:55 bolt
4 drwxr-xr-x 3 root root 4096 Jan  4 13:55 cluster
4 drwxr-xr-x 3 root root 4096 Jan  4 13:55 https
```

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

Example output

```
total 8
0 drwxrwsrwt 3 root neo4j  100 Jan  4 13:56 trusted
4 -rw-r--r-- 1 root neo4j 1704 Jan  4 13:56 private.key
4 -rw-r--r-- 1 root neo4j 1159 Jan  4 13:56 public.crt
```

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

Example output

```
total 0
0 lrwxrwxrwx 1 root neo4j 17 Jan  4 13:56 public.crt -> ..data/public.crt
```

5. Exit the pod:

```
exit
```

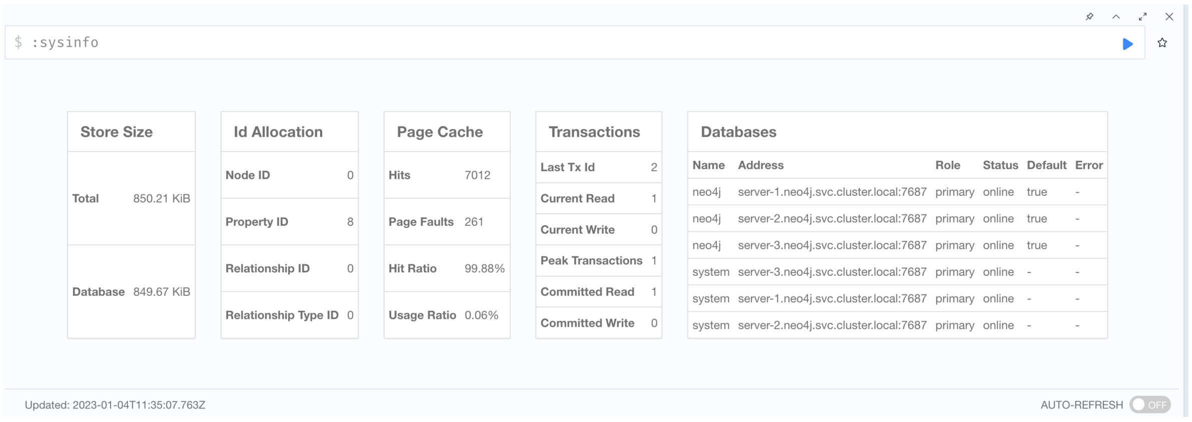
6. Check that the LoadBalancer service is available using the `neo4j.name` used for the installation:

```
export NEO4J_NAME=my-cluster
kubectl get service ${NEO4J_NAME}-lb-neo4j
```

Example output

```
NAME                  TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)
AGE
my-cluster-lb-neo4j   LoadBalancer   10.0.134.210   20.237.50.207
7474:31168/TCP,7473:31045/TCP,7687:32708/TCP   3m30s
```

7. Connect to the Neo4j cluster using one of the following options:
   - Neo4j Browser:
     a. Open a web browser and type https://lb-EXTERNAL_IP:7473 (in this example, https://20.237.50.207:7473/browser/). You should see the Neo4j browser.
     b. Authenticate using the user `neo4j` and the password you set when deploying the cores, in this example, `my-password`.
     c. Verify that the cluster is online by running `:sysinfo` or `SHOW SERVERS`:



```
$ :sysinfo
```

Store Size	
Total	850.21 KiB
Database	849.67 KiB

Id Allocation	
Node ID	0
Property ID	8
Relationship ID	0
Relationship Type ID	0

Page Cache	
Hits	7012
Page Faults	261
Hit Ratio	99.88%
Usage Ratio	0.06%

Transactions	
Last Tx Id	2
Current Read	1
Current Write	0
Peak Transactions	1
Committed Read	1
Committed Write	0

Databases

Name	Address	Role	Status	Default	Error
neo4j	server-1.neo4j.svc.cluster.local:7687	primary	online	true	-
neo4j	server-2.neo4j.svc.cluster.local:7687	primary	online	true	-
neo4j	server-3.neo4j.svc.cluster.local:7687	primary	online	true	-
system	server-3.neo4j.svc.cluster.local:7687	primary	online	-	-
system	server-1.neo4j.svc.cluster.local:7687	primary	online	-	-
system	server-2.neo4j.svc.cluster.local:7687	primary	online	-	-

Updated: 2023-01-04T11:35:07.763Z

     d. Run `CALL dbms.listConfig('ssl') YIELD name, value` to verify that the configuration is deployed as expected.
   - Cypher Shell:
     a. Open a terminal and connect to one of the cluster pods:

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

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

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

Example output

```
Connected to Neo4j using Bolt protocol version 5.3 at neo4j+ssc://server-
1.neo4j.svc.cluster.local:7687 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
neo4j@neo4j>
```

     c. Verify that the cluster is online by running `SHOW SERVERS`:

```
neo4j@server-1-0:~/bin$ SHOW SERVERS;
```

Example output

```
+-----------------------------------------------------------------------------------------------
--------------------------------------+
| name                                       | address                                     | state
| health       | hosting              |
+-----------------------------------------------------------------------------------------------
--------------------------------------+
| "1c5946b1-0eb5-43b9-a549-5601087c57f2" | "server-3.neo4j.svc.cluster.local:7687" | "Enabled"
| "Available" | ["neo4j", "system"]  |
| "ba63cd32-3e7d-4042-9935-c8eba925a98f" | "server-1.neo4j.svc.cluster.local:7687"  |
"Enabled" | "Available" | ["neo4j", "system"] |
| "cbad7ed6-0c13-4ba7-b6a1-f20c5552dfcd" | "server-2.neo4j.svc.cluster.local:7687" | "Enabled"
| "Available" | ["neo4j", "system"]  |
+-----------------------------------------------------------------------------------------------
--------------------------------------+
```

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

Example output

```
+-----------------------------------------------------------------------------------------------
-------+
| name                                         | value
|
+-----------------------------------------------------------------------------------------------
-------+
| "dbms.netty.ssl.provider"                    | "JDK"
|
| "dbms.ssl.policy.bolt.base_directory"        | "/var/lib/neo4j/certificates/bolt"
|
| "dbms.ssl.policy.bolt.ciphers"               | "No Value"
|
| "dbms.ssl.policy.bolt.client_auth"           | "NONE"
|
| "dbms.ssl.policy.bolt.enabled"               | "true"
|
| "dbms.ssl.policy.bolt.private_key"           |
"/var/lib/neo4j/certificates/bolt/private.key"     |
| "dbms.ssl.policy.bolt.private_key_password"  | "No Value"
|
| "dbms.ssl.policy.bolt.public_certificate"    |
"/var/lib/neo4j/certificates/bolt/public.crt"     |
| "dbms.ssl.policy.bolt.revoked_dir"           | "/var/lib/neo4j/certificates/bolt/revoked"
|
| "dbms.ssl.policy.bolt.tls_versions"          | "TLSv1.2"
|
| "dbms.ssl.policy.bolt.trust_all"             | "false"
|
| "dbms.ssl.policy.bolt.trusted_dir"           | "/var/lib/neo4j/certificates/bolt/trusted"
|
| "dbms.ssl.policy.bolt.verify_hostname"       | "false"
|
| "dbms.ssl.policy.cluster.base_directory"     | "/var/lib/neo4j/certificates/cluster"
|
| "dbms.ssl.policy.cluster.ciphers"            | "No Value"
|
| "dbms.ssl.policy.cluster.client_auth"        | "REQUIRE"
|
| "dbms.ssl.policy.cluster.enabled"            | "true"
|
| "dbms.ssl.policy.cluster.private_key"        |
"/var/lib/neo4j/certificates/cluster/private.key" |
| "dbms.ssl.policy.cluster.private_key_password" | "No Value"
|
| "dbms.ssl.policy.cluster.public_certificate"   |
"/var/lib/neo4j/certificates/cluster/public.crt"  |
| "dbms.ssl.policy.cluster.revoked_dir"          |
"/var/lib/neo4j/certificates/cluster/revoked"      |
| "dbms.ssl.policy.cluster.tls_versions"         | "TLSv1.2"
|
| "dbms.ssl.policy.cluster.trust_all"            | "false"
|
| "dbms.ssl.policy.cluster.trusted_dir"          |
"/var/lib/neo4j/certificates/cluster/trusted"      |
| "dbms.ssl.policy.cluster.verify_hostname"      | "false"
|
| "dbms.ssl.policy.https.base_directory"         | "/var/lib/neo4j/certificates/https"
|
| "dbms.ssl.policy.https.ciphers"                | "No Value"
|
| "dbms.ssl.policy.https.client_auth"            | "NONE"
|
| "dbms.ssl.policy.https.enabled"                | "true"
|
| "dbms.ssl.policy.https.private_key"            |
"/var/lib/neo4j/certificates/https/private.key"    |
| "dbms.ssl.policy.https.private_key_password"   | "No Value"
|
| "dbms.ssl.policy.https.public_certificate"     |
"/var/lib/neo4j/certificates/https/public.crt"     |
| "dbms.ssl.policy.https.revoked_dir"            | "/var/lib/neo4j/certificates/https/revoked"
|
| "dbms.ssl.policy.https.tls_versions"           | "TLSv1.2"
|
| "dbms.ssl.policy.https.trust_all"              | "false"
|
| "dbms.ssl.policy.https.trusted_dir"            | "/var/lib/neo4j/certificates/https/trusted"
|
| "dbms.ssl.policy.https.verify_hostname"        | "false"
|
+----------------------------------------------------------------------------------------------
-------+
```

```
37 rows
ready to start consuming query after 212 ms, results consumed after another 11 ms
```

Configure SSO

Neo4j supports SSO authentication and authorization through identity providers implementing the OpenID Connect (OIDC) standard.

To configure the Neo4j helm deployment to use SSO authentication, first, you need to configure your identity provider for authentication and authorization using ID tokens. And then, you configure the Neo4j helm deployment to use that identity provider for authentication by adding all the SSO configurations to the *values.yaml* file.

For more information on how to configure your identity provider and what settings you should define, see Neo4j Single Sign-On (SSO) configuration.

An example of configuring Neo4j to use Azure SSO for authentication

```
config:
  server.security.oidc.azure.audience: "00f3a7d3-d855-4849-9e3c-57d7b6e12794"
  server.security.oidc.azure.params: "client_id=00f3a7d3-d855-4849-9e3c-
57d7b6e12794;response_type=code;scope=openid profile email"
  server.security.oidc.azure.well_known_discovery_uri: "https://login.microsoftonline.com/da501982-4ca7-
420c-8926-1e65b5bf565f/v2.0/.well-known/openid-configuration"
  server.security.authorization_providers: "oidc-azure,native"
  server.security.authentication_providers: "oidc-azure,native"
  server.security.oidc.azure.display_name: "Azure SSO on K8s"
  server.security.oidc.azure.auth_flow: "pkce"
  server_type_principal=id_token;token_type_authentication=id_token"
  server.security.oidc.azure.config: "principal=unique_name;code_challenge_method=S256;
  server.security.oidc.azure.claims.username: "sub"
  server.security.oidc.azure.claims.groups: "groups"
  server.security.oidc.azure.authorization.group_to_role_mapping: "e197354c-bd75-4524-abbc-
d44325904567=editor;fa31ce67-9e4d-4999-bf6d-25c55258d116=publisher"
```

sub is the only claim guaranteed to be unique and stable. Other claims, such as email or preferred_username, may change over time and should **not** be used for authentication. Neo4j may assign permissions to a user based on this username value in a hybrid authorization configuration. Thus, changing the username claim from sub is not recommended. For details, see Microsoft documentation as well as the OpenId spec.

Configure resource allocation

CPU and memory

The resources (CPU, memory) for the Neo4j container are configured by setting neo4j.resources object in the *values.yaml* file. In the resource *requests*, you can specify how much CPU and memory the Neo4j container needs, while in the resource *limits*, you can set a limit on these resources in case the container tries to use more resources than its *requests* allow.

```
neo4j:
  resources:
    requests:
     cpu: "1000m"
     memory: "2Gi"
    limits:
     cpu: "2000m"
     memory: "4Gi"
```

If no resource *requests* and resource *limits* are specified, the values set in the resources object are used for both the Neo4j container's resource *requests* and resource *limits*.

```
neo4j:
  resources:
    cpu: "2"
    memory: "5Gi"
```

The minimum for a Neo4j instance is `0.5` CPU and `2GB` memory.
If invalid or less than the minimum values are provided, Helm will throw an error, for example:

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

JVM heap and page cache

You configure Neo4j to use the memory provided to the container by setting the parameters `server.memory.heap.initial_size` and `server.memory.pagecache.size`. Combined, they must not exceed the memory configuration of the Neo4j container.
In Kubernetes, running processes in the Neo4j container, which exceed the configured memory limit are killed by the underlying operating system. Therefore, it is recommended to allow an additional 1GB of memory headroom so that `heap + pagecache + 1GB < available memory`.

For example, a 5GB container could be configured like this:

```
neo4j:
  resources:
    cpu: "2"
    memory: "5Gi"

# Neo4j configuration (yaml format)
config:
  server.memory.heap.initial_size: "3G"
  server.memory.heap.max_size: "3G"
  server.memory.pagecache.size: "1G"
```

`server.memory.pagecache.size` and `server.memory.heap.initial_size` are not the only settings available in Neo4j to manage memory usage. For full details of how to configure memory usage in Neo4j, see Performance - Memory Configuration.

Configure a service account

In some deployment situations, it may be desirable to assign a Kubernetes Service Account to the Neo4j pod. For example, if processes in the pod want to connect to services that require Service Account authorization. To configure the Neo4j pod to use a Kubernetes service account, set `podSpec.serviceAccountName` to the name of the service account to use.

For example:

```
# neo4j-values.yaml
neo4j:
  password: "my-password"

podSpec:
  serviceAccountName: "neo4j-service-account"
```

The service account must already exist. The Neo4j Helm chart does not create or configure Service Accounts.

Configure a custom container image

The helm chart uses the official Neo4j Docker image that matches the version of the Helm chart. To configure the helm chart to use a different container image, set the `image.customImage` property in the *values.yaml* file.

This can be necessary when public container repositories are not accessible for security reasons. For example, this *values.yaml* file configures Neo4j to use `my-container-repository.io` as the container repository:

```
# neo4j-values.yaml
neo4j:
  password: "my-password"

image:
  customImage: "my-container-repository.io/neo4j:5-enterprise"
```

Configure and install APOC core only

APOC core is shipped with Neo4j, but it is not installed in the Neo4j *plugins* directory. If APOC core is the *only* plugin that you want to add to Neo4j, it is not necessary to perform plugin installation as described in Install Plugins. Instead, you can configure the helm deployment to use APOC core by upgrading the deployment with this additional setting in the *values.yaml* file:

1. Configure APOC core:

```
config:
  server.directories.plugins: "/var/lib/neo4j/labs"
  dbms.security.procedures.unrestricted: "apoc.*"
  server.config.strict_validation.enabled: "false"
  dbms.security.procedures.allowlist: "gds.*,apoc.*"
```

2. Under `apoc_config`, configure the APOC settings you want, for example:

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

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

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

4. After the Helm upgrade rollout is complete, verify that APOC core has been configured by running the following Cypher query using `cypher-shell` or Neo4j Browser:

```
RETURN apoc.version()
```

Install Plugins

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

- an automatic plugin download
- an init container
- a custom container image.
- a `plugins` volume.

Add plugins using an automatic plugin download

You can configure the Neo4j deployment to automatically download and install plugins. If licenses are required for the plugins, you must provide the licenses in a secret.

Install GDS Community Edition (CE)

GDS Community Edition does not require a license. To add the GSD CE, configure the Neo4j *values.yaml* and set the `env` to download the plugins:

```
neo4j:
  name: licenses
  acceptLicenseAgreement: "yes"
  edition: enterprise
volumes:
  data:
    mode: defaultStorageClass
env:
  NEO4J_PLUGINS: '["graph-data-science"]'
config:
  dbms.security.procedures.unrestricted: "gds.*,apoc.*"
```

Install GDS Enterprise Edition (EE) and Bloom plugins

To install GDS EE and Bloom, you must provide a license for each plugin. You provide the licenses in a secret.

1. Create a secret containing the licenses:

   ```
   kubectl create secret  generic --from-file=gds.license,bloom.license gds-bloom-license
   ```

2. Configure the Neo4j *values.yaml* file using the secret as the */licenses* volume mount, and set the `env` to download the plugins:

```
neo4j:
  name: licenses
  acceptLicenseAgreement: "yes"
  edition: enterprise
volumes:
  data:
    mode: defaultStorageClass
  licenses:
    mode: volume
    volume:
      secret:
        secretName: gds-bloom-license
        items:
          - key: gds.license
            path: gds.license
          - key: bloom.license
            path: bloom.license
env:
  NEO4J_PLUGINS: '["graph-data-science", "bloom"]'
config:
  gds.enterprise.license_file: "/licenses/gds.license"
  dbms.security.procedures.unrestricted: "gds.*,apoc.*,bloom.*"
  server.unmanaged_extension_classes: "com.neo4j.bloom.server=/bloom,semantics.extension=/rdf"
  dbms.security.http_auth_allowlist: "/,/browser.*,/bloom.*"
  dbms.bloom.license_file: "/licenses/bloom.license"
```

Add plugins using an init container

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

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

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

   Some Neo4j plugins, such as Bloom and GDS Enterprise, require a license activation key, which needs to be placed in a directory accessible by the Neo4j Docker container, for example, mounted to */licenses* (default). To obtain a valid license, reach out to your Neo4j account representative or write to licensing@neo4j.com.

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

  password: "password"

  edition: "enterprise"
  acceptLicenseAgreement: "yes"


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

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

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

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

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

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

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

Example output

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

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

4. Using the `<EXTERNAL-IP>` of the LoadBalancer service, open the Neo4j Browser at *http://<EXTERNAL-IP>:7474/browser*.

5. Use the password that you have configured in the *plugin_initContainer.yaml* file.
6. Run a query to verify that the Apoc plugin is installed and configured. For example, you can use the following query to get the value of your jdbc URL:

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

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

Add plugins using a custom container image

The best method for adding plugins to Neo4j running in Kubernetes is to create a new Docker container image that contains both Neo4j and the Neo4j plugins. This way, you can ensure when building the container that the correct plugin version for the Neo4j version of the container is used and that the resulting image encapsulates all Neo4j runtime dependencies.

The Neo4j Bloom plugin (https://neo4j.com/download-center/#bloom) requires a license activation key, which needs to be placed in a directory accessible by the Neo4j Docker container, for example, mounted to */licenses* (default). To obtain a valid license, reach out to your Neo4j account representative or use the form Contact Neo4j.

Building a Docker container image that is based on the official Neo4j Docker image and does not override the official image's `ENTRYPOINT` and `COMMAND` is the recommended method to use with the Neo4j Helm chart, as shown in this example Dockerfile:

```
ARG  NEO4J_VERSION
FROM neo4j:{NEO4J_VERSION}

# copy my-plugins into the Docker image
COPY my-plugins/ /var/lib/neo4j/plugins

# install the apoc core plugin that is shipped with Neo4j
RUN cp /var/lib/neo4j/labs/apoc-* /var/lib/neo4j/plugins
```

Once the docker image has been built, push it to a container repository that is accessible to your Kubernetes cluster.

```
CONTAINER_REPOSITORY="my-container-repository.io"
IMAGE_NAME="my-neo4j"

# export this so that it's accessible as a docker build arg
export NEO4J_VERSION=5.4.0-enterprise

docker build --build-arg NEO4J_VERSION --tag ${CONTAINER_REPOSITORY}/${IMAGE_NAME}:${NEO4J_VERSION} .
docker push ${CONTAINER_REPOSITORY}/${IMAGE_NAME}:${NEO4J_VERSION}
```

To use the image that you have created, in the Neo4j Helm deployment's *values.yaml* file, set `image.customImage` to use the image. For more details, see Configure a custom container image.

Many plugins require additional Neo4j configuration to work correctly. Plugin configuration should be set on the `config` object in the Helm deployment's *values.yaml* file. In some cases, plugin configuration can cause Neo4j's strict config validation to fail. Strict config validation can be disabled by setting `server.config.strict_validation.enabled: "false"`.

Add plugins using a plugins volume

An alternative method for adding Neo4j plugins to a Neo4j Helm deployment uses a `plugins` volume mount. With this method, the plugin jar files are stored on a Persistent Volume that is mounted to the `/plugins` directory of the Neo4j container.

The Neo4j Bloom plugin (https://neo4j.com/download-center/#bloom) requires a license activation key, which needs to be placed in a directory accessible by the Neo4j Docker container, for example, mounted to */licenses* (default). To obtain a valid license, reach out to your Neo4j account representative or use the form Contact Neo4j.

The simplest way to set up a persistent */plugins* volume is to share the Persistent Volume that is used for storing Neo4j data. This example shows how to configure that in the Neo4j Helm deployment *values.yaml* file:

```yaml
# neo4j-values.yaml
volumes:
  data:
    # your data volume configuration
    ...

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

Details of different ways to configure volume mounts are covered in Mapping volume mounts to persistent volumes.

The Neo4j container now has an empty */plugins* directory backed by a persistent volume. Plugin jar files can be copied onto the volume using `kubectl cp`. Because it is backed by a persistent volume, plugin files will persist even if the Neo4j pod is restarted or moved.

Neo4j loads plugins only on startup. Therefore, you must restart the Neo4j pod to load them once all plugins are in place.

For example:

```
# Copy plugin files into the Neo4j container:

kubectl cp my-plugins/* <namespace>/<neo4j-pod-name>:/plugins/

# Restart Neo4j
kubectl rollout restart statefulset/<neo4j-statefulset-name>

# Verify plugins are still present after the restart:

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

Volume mounts and persistent volumes with the Neo4j Helm chart

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

Volume mounts

A *volume mount* is part of a Kubernetes Pod spec that describes how and where a volume is mounted within a container.

The Neo4j Helm chart creates the following volume mounts:

- `backups` mounted at */backups*
- `data` mounted at */data*
- `import` mounted at */import*
- `licenses` mounted at */licenses*
- `logs` mounted at */logs*
- `metrics` mounted at */metrics* (Neo4j Community Edition does not generate `metrics`.)

It is also possible to specify a `plugins` volume mount (mounted at */plugins*), but this is not created by the default Helm chart. For more information, see Add plugins using a plugins volume.

Persistent volumes

`PersistentVolume` (PV) is a storage resource in the Kubernetes cluster that has a lifecycle independent of any individual pod that uses the PV.
`PersistentVolumeClaim` (PVC) is a request for a storage resource by a user. PVCs consume PV resources. For more information about what PVs are and how they work, see the Kubernetes official documentation.

The type of PV used and its configuration can have a significant effect on the performance of Neo4j. Some PV types are not suitable for use with Neo4j at all.

The volume type used for the `data` volume mount is particularly important. Neo4j supports the following PV types for the `data` volume mount:

- `persistentVolumeClaim`

- `hostPath` when using Docker Desktop [3].

Neo4j `data` volume mount does not support `azureFile` and `nfs`.

> `awsElasticBlockStore`, `azureDisk`, `gcePersistentDisk` are now deprecated volume types in Kubernetes and their use is no longer supported by the Neo4j Helm chart. If you currently use one of these volume types, consult your Kubernetes vendor's documentation on migrating to Container Storage Interface (CSI) driver-based storage.

For volume mounts other than the `data` volume mount, generally, all PV types are presumed to work.

> `hostPath`, `local`, and `emptyDir` types are expected to perform well, provided suitable underlying storage, such as SSD, is used. However, these volume types have operational limitations and are not recommended.
>
> It is also not recommended to use an HDD or cloud storage, such as AWS S3 mounted as a drive.

Mapping volume mounts to persistent volumes

By default, the Neo4j Helm chart uses a single PV, named `data`, to support volume mounts.

The volume used for each volume mount can be changed by modifying the `volumes.<volume name>` object in the Helm chart values.

The Neo4j Helm chart `volumes` object supports different modes:

`mode: dynamic` Recommended

Description

Dynamic volumes are recommended for most production workloads due to ease of management. The volume mount is backed by a PV that Kubernetes dynamically provisions using a dedicated `StorageClass`. The `StorageClass` is specified in the `storageClassName` field.

Example

The data volume uses a dedicated storage class:

storage-class-values.yaml

```yaml
neo4j:
  name: standalone-with-storage-class
volumes:
  data:
    mode: dynamic
    dynamic:
      storageClassName: "neo4j-data"
      requests:
        storage: 10Gi
```

See Provision a PV using a dedicated `StorageClass` Recommended for more information.

`mode: share`

Description

The volume mount shares the underlying volume from one of the other volume objects.

Example

The `logs` volume mount uses the `data` volume (this is the default behavior).

```
volumes:
  logs:
    mode: "share"
    share:
      name: "data"
```

`mode: defaultStorageClass`

Description

The volume mount is backed by a PV that Kubernetes dynamically provisions using the default `StorageClass`.

Example

A dynamically provisioned `data` volume with a size of `10Gi`.

```
volumes:
  data:
    mode: "defaultStorageClass"
    defaultStorageClass:
      requests:
        storage: 10Gi
```

For the `data` volume, if `requests.storage` is not set, `defaultStorageClass` defaults to a `10Gi` volume. For all other volumes, `defaultStorageClass.requests.storage` must be set explicitly when using `defaultStorageClass` mode.

`mode: volume`

Description

A complete Kubernetes `volume` object can be specified for the volume mount. Generally, volumes specified in this way have to be manually provisioned.

`volume` can be any valid Kubernetes volume type. This mode is typically used to mount a pre-existing Persistent Volume Claim (PVC).

For details on how to specify `volume` objects, see the Kubernetes documentation.

Set file permissions on mounted volumes

The Neo4j Helm chart supports an additional field not present in normal Kubernetes `volume` objects: `setOwnerAndGroupWritableFilePermissions: true|false`. If set to `true`, an `initContainer` will be run to modify the file permissions of the mounted volume, so that the contents can be written and read by the Neo4j process. This is to help with certain volume implementations that are not aware of the `SecurityContext` set on pods using them.

Example - reference an existing PersistentVolume

The `backups` volume mount is backed by the specified PVC. When this method is used, the `persistentVolumeClaim` object must already exist.

```yaml
volumes:
  backups:
    mode: volume
    volume:
      persistentVolumeClaim:
        claimName: my-neo4j-pvc
```

`mode: selector`

Description

The volume to use is chosen from the existing PVs based on the provided `selector` object and a PVC that is dynamically generated.

If no matching PVs exist, the Neo4j pod will be unable to start. To match, a PV must have the specified `StorageClass`, match the label `selectorTemplate`, and have sufficient storage capacity to meet the requested storage amount.

Example

The `data` volume is chosen from the available volumes with the `neo4j` storage class and the label `developer: alice`.

```yaml
volumes:
  import:
    mode: selector
    selector:
      storageClassName: "neo4j"
      requests:
        storage: 128Gi
      selectorTemplate:
        matchLabels:
          developer: "alice"
```

For the `data` volume, if `requests.storage` is not set, `selector` defaults to a `100Gi` volume. For all other volumes, `selector.requests.storage` must be set explicitly when using `selector` mode.

`mode: volumeClaimTemplate`

Description

A complete Kubernetes `volumeClaimTemplate` object is specified for the volume mount. Volumes specified in this way are dynamically provisioned.

Example - provision Neo4j storage using a volume claim template

The data volume uses a dynamically provisioned PVC from the `default` storage class.

```yaml
volumes:
  data:
    mode: volumeClaimTemplate
    volumeClaimTemplate:
      storageClassName: "default"
      accessModes:
        - ReadWriteOnce
      resources:
        requests:
          storage: 10Gi
```

In all cases, do not forget to set the `mode` field when customizing the volumes object. If not set, the default `mode` is used, regardless of the other properties set on the `volume` object.

Provision persistent volumes with Neo4j Helm chart

Provision persistent volumes dynamically

With the Neo4j Helm chart, you can provision a PV dynamically using the default or a custom `StorageClass`. To see a list of available storage classes in your Kubernetes cluster, run the following command:

```
kubectl get storageclass
```

Provision a PV using a dedicated `StorageClass` Recommended

For production workloads, it is recommended to create a dedicated storage class for Neo4j, which uses the `Retain` reclaim policy. This is to avoid data loss when disks are deleted after removing the persistent volume resource.

Example: Deploy Neo4j using a dedicated `StorageClass`

The following example shows how to deploy a Neo4j server with a dynamically provisioned PV that uses a dedicated `storageClass`.

1. Create a dedicated storage class that uses the `Retain` reclaim policy:

1. Create a storage class in GKE that uses the `Retain` reclaim policy and `pd-ssd` high-performance SSD disks:

   ```
   cat <<EOF | kubectl apply -f -
   apiVersion: storage.k8s.io/v1
   kind: StorageClass
   metadata:
     name: neo4j-data
   provisioner: pd.csi.storage.gke.io
   parameters:
     type: pd-ssd
   reclaimPolicy: Retain
   volumeBindingMode: WaitForFirstConsumer
   allowVolumeExpansion: true
   EOF
   ```

2. Check the storage class is created:

   ```
   kubectl get storageclass neo4j-data
   ```

   ```
   NAME                     PROVISIONER             RECLAIMPOLICY    VOLUMEBINDINGMODE
   ALLOWVOLUMEEXPANSION   AGE
   neo4j-data               pd.csi.storage.gke.io   Retain           WaitForFirstConsumer
   true                     7s
   ```

1. Create a storage class in EKS that uses the `Retain` reclaim policy and `gp3` high-performance SSD disks:

   > The EBS CSI Driver addon is required to provision EBS disks in EKS clusters. See the AWS documentation for instructions on installing the driver.

   ```
   cat <<EOF | kubectl apply -f -
   kind: StorageClass
   apiVersion: storage.k8s.io/v1
   metadata:
     name: neo4j-data
   provisioner: ebs.csi.aws.com
   parameters:
     type: gp3
   reclaimPolicy: Retain
   allowVolumeExpansion: true
   volumeBindingMode: WaitForFirstConsumer
   EOF
   ```

2. Check the storage class is created:

   ```
   kubectl get storageclass neo4j-data
   ```

   ```
   NAME            PROVISIONER            RECLAIMPOLICY   VOLUMEBINDINGMODE
   ALLOWVOLUMEEXPANSION   AGE
   neo4j-data      ebs.csi.aws.com        Retain          WaitForFirstConsumer   true
   2m41s
   ```

1. Create a storage class in GKE that uses the `Retain` reclaim policy and `pd-ssd` high-performance SSD disks:

   ```
   cat <<EOF | kubectl apply -f -
   apiVersion: storage.k8s.io/v1
   kind: StorageClass
   metadata:
     name: neo4j-data
   provisioner: disk.csi.azure.com
   parameters:
     skuName: Premium_LRS
   reclaimPolicy: Retain
   volumeBindingMode: WaitForFirstConsumer
   allowVolumeExpansion: true
   EOF
   ```

2. Check the storage class is created:

   ```
   kubectl get storageclass neo4j-data
   ```

   ```
   NAME                     PROVISIONER             RECLAIMPOLICY    VOLUMEBINDINGMODE
   ALLOWVOLUMEEXPANSION   AGE
   neo4j-data               disk.csi.azure.com      Retain           WaitForFirstConsumer
   true                     7s
   ```

2. Install a Neo4j server with a data volume that uses the new storage class:

   a. Create a file *storage-class-values.yaml* that configures the data volume to use the new storage class:

   storage-class-values.yaml

   ```
   neo4j:
     name: standalone-with-storage-class
   volumes:
     data:
       mode: dynamic
       dynamic:
         storageClassName: "neo4j-data"
         requests:
           storage: 10Gi
   ```

   b. Install a single Neo4j server:

   ```
   helm install standalone-with-storage-class neo4j -f storage-class-values.yaml
   ```

   c. When the installation completes, verify that a PVC has been created:

   ```
   kubectl get pvc
   ```

```
NAME                                   STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS   AGE
data-standalone-with-storage-class-0   Bound    pvc-5d400f06-f99f-43ac-bf37-6079d692eaac   10Gi
RWO            neo4j-data     23m
```

3. Clean up the resources:

   The storage class uses the `Retain` retention policy, meaning the disk will **not** be deleted after removing the PVC. To delete the disk, patch the PVC to use the `Delete` retention policy and delete the PVC:

```
export pv_name=$(kubectl get pvc data-standalone-with-storage-class-0 -o jsonpath
='{.spec.volumeName}')

kubectl patch pv $pv_name -p '{"spec":{"persistentVolumeReclaimPolicy": "Delete"}}'

kubectl delete pvc data-standalone-with-storage-class-0
```

For the `data` volume, if `requests.storage` is not set, `dynamic` defaults to a `100Gi` volume. For all other volumes, `dynamic.requests.storage` must be set explicitly when using `dynamic` mode.

Provision a PV using `defaultStorageClass`

Using the default `StorageClass` of the running Kubernetes cluster is the quickest way to spin up and run Neo4j for simple tests, handling small amounts of data. However, it is not recommended for large amounts of data, as it may lead to performance issues.

Example: Deploy Neo4j using `defaultStorageClass`

The following example shows how to deploy a Neo4j server with a dynamically provisioned PV that uses the default `StorageClass`.

1. Create a file *default-storage-class-values.yaml* that configures the data volume to use the default `StorageClass` and a storage size `100Gi`:

   storage-class-values.yaml

```
volumes:
  data:
    mode: "defaultStorageClass"
    defaultStorageClass:
      requests:
        storage: 100Gi
```

2. Install a single Neo4j server:

```
helm install standalone-with-default-storage-class neo4j -f default-storage-class-values.yaml
```

Provision persistent volumes manually

Optionally, the Helm chart can use manually created disks for Neo4j storage. This installation option has more steps than using dynamic volumes, but it does provide more control over how disks are provisioned.

The instructions for the manual provisioning of PVs vary according to the type of PV being used and the underlying infrastructure. In general, there are two steps:

1. Create the disk/volume to be used for storage in the underlying infrastructure. For example:
   - If using a `csi` volume — create the Persistent Disk using the cloud provider CLI or console.
   - If using a `hostPath` volume — on the host node, create the path (directory).
2. Create a PV in Kubernetes that references the underlying resource created in step 1.
   a. Ensure that the created PV's `app` label matches the name of the Neo4j Helm release.
   b. Ensure that the created PV's `capacity.storage` matches the storage available on the underlying infrastructure.

If no suitable PV or PVC exists, the Neo4j pod will not start.

Provision a PV for Neo4j Storage using a PV selector

The Neo4j StatefulSet can select a persistent volume to use based on its labels. A Neo4j Helm release uses only manually provisioned PVs that have:

- `storageClassName` that uses the provisioner `kubernetes.io/no-provisioner`.
- An `app` label — set in their metadata, which matches the name of the `neo4j.name` value of the Helm installation.
- Sufficient storage capacity — the PV capacity must be greater than or equal to the value of `volumes.data.selector.requests.storage` set for the Neo4j Helm release (default is `100Gi`).

> The neo4j/neo4j-persistent-volume Helm chart provides a convenient way to provision the persistent volume.

Example: Deploy Neo4j using a selector volume

The following example shows how to deploy Neo4j using a selector volume.

1. Create a file *persistent-volume-selector.yaml* that configures the data volume to use a selector:

   storage-class-values.yaml

   ```
   neo4j:
     name: volume-selector
   volumes:
     data:
       mode: selector
       selector:
         storageClassName: "manual"
         accessModes:
           - ReadWriteOnce
         requests:
           storage: 10Gi
   ```

2. Export environment variables to be used by the commands:

   ```
   export RELEASE_NAME=volume-selector
   export GCP_ZONE="$(gcloud config get compute/zone)"
   export GCP_PROJECT="$(gcloud config get project)"
   ```

3. Create the disks to be used by the persistent volume:

   ```
   gcloud compute disks create --size 10Gi --type pd-ssd "${RELEASE_NAME}"
   ```

4. Use the *neo4j/neo4j-persistent-volume* chart to configure the persistent volume. This command will create a persistent volume and a manual storage class that uses the `kubernetes.io/no-provisioner` provisioner.

   ```
   helm install "${RELEASE_NAME}"-disk neo4j/neo4j-persistent-volume \
          --set neo4j.name="${RELEASE_NAME}" \
          --set data.driver=pd.csi.storage.gke.io \
          --set data.storageClassName="manual" \
          --set data.reclaimPolicy="Delete" \
          --set data.createPvc=false \
          --set data.createStorageClass=true \
          --set data.volumeHandle="projects/${GCP_PROJECT}/zones/${GCP_ZONE}/disks/
   ${RELEASE_NAME}" \
          --set data.capacity.storage=10Gi
   ```

5. Now install Neo4j using the `persistent-volume-selector.yaml` created earlier:

   ```
   helm install "${RELEASE_NAME}" neo4j/neo4j -f persistent-volume-selector.yaml
   ```

6. Clean up the helm installation and disks created for the example:

   ```
   helm uninstall ${RELEASE_NAME} ${RELEASE_NAME}-disk
   kubectl delete pvc data-${RELEASE_NAME}-0
   gcloud compute disks delete ${RELEASE_NAME} --quiet
   ```

The EBS CSI Driver addon is required to provision EBS disks in EKS clusters. You can run the command `kubectl get daemonset ebs-csi-node -n kube-system` to check if it is installed See the AWS Documentation for instructions on installing the driver.

1. Create a file `persistent-volume-selector.yaml` that configures the data volume to use a selector:

   storage-class-values.yaml

```
neo4j:
  name: volume-selector
volumes:
  data:
    mode: selector
    selector:
      storageClassName: "manual"
      accessModes:
        - ReadWriteOnce
      requests:
        storage: 10Gi
```

2. Export environment variables to be used by the commands:

```
readonly RELEASE_NAME=volume-selector
readonly AWS_ZONE={availability zone of EKS cluster}
```

3. Create the disks to be used by the persistent volume:

```
export volumeId=$(aws ec2 create-volume \
                    --availability-zone="${AWS_ZONE}" \
                    --size=10 \
                    --volume-type=gp3 \
                    --tag-specifications 'ResourceType=volume,Tags=[{Key=volume,Value='
"${RELEASE_NAME}"'}]' \
                    --no-cli-pager \
                    --output text \
                    --query VolumeId)
```

4. Use the *neo4j/neo4j-persistent-volume* chart to configure the persistent volume. This command will create a persistent volume and a manual storage class that uses the `kubernetes.io/no-provisioner` provisioner.

```
helm install "${RELEASE_NAME}"-disk neo4j-persistent-volume \
    --set neo4j.name="${RELEASE_NAME}" \
    --set data.driver=ebs.csi.aws.com \
    --set data.reclaimPolicy="Delete" \
    --set data.createPvc=false \
    --set data.createStorageClass=true \
    --set data.volumeHandle="${volumeId}" \
    --set data.capacity.storage=10Gi
```

5. Now install Neo4j using the `persistent-volume-selector.yaml` created earlier:

```
helm install "${RELEASE_NAME}" neo4j/neo4j -f persistent-volume-selector.yaml
```

6. Clean up the helm installation and disks created for the example:

```
helm uninstall ${RELEASE_NAME} ${RELEASE_NAME}-disk
    kubectl delete pvc data-${RELEASE_NAME}-0
    aws ec2 delete-volume --volume-id ${volumeId}
```

1. Create a file `persistent-volume-selector.yaml` that configures the data volume to use a selector:

   storage-class-values.yaml

   ```
   neo4j:
     name: volume-selector
   volumes:
     data:
       mode: selector
       selector:
         storageClassName: "manual"
         accessModes:
           - ReadWriteOnce
         requests:
           storage: 10Gi
   ```

2. Export environment variables to be used by the commands:

   ```
   readonly AKS_CLUSTER_NAME={AKS Cluster name}
   readonly AZ_RESOURCE_GROUP={Resource group of cluster}
   readonly AZ_LOCATION={Location of cluster}
   ```

3. Create the disks to be used by the persistent volume:

   ```
   export node_resource_group=$(az aks show --resource-group "${AZ_RESOURCE_GROUP}" --name
   "${AKS_CLUSTER_NAME}" --query nodeResourceGroup -o tsv)
   export disk_id=$(az disk create --name "${RELEASE_NAME}" --size-gb "10" --max-shares 1
   --resource-group "${node_resource_group}" --location ${AZ_LOCATION} --output tsv --query
   id)
   ```

4. Use the *neo4j/neo4j-persistent-volume* chart to configure the persistent volume. This command will create a persistent volume and a manual storage class that uses the `kubernetes.io/no-provisioner` provisioner.

   ```
   helm install "${RELEASE_NAME}"-disk neo4j-persistent-volume \
       --set neo4j.name="${RELEASE_NAME}" \
       --set data.driver=disk.csi.azure.com \
       --set data.storageClassName="manual" \
       --set data.reclaimPolicy="Delete" \
       --set data.createPvc=false \
       --set data.createStorageClass=true \
       --set data.volumeHandle="${disk_id}" \
       --set data.capacity.storage=10Gi
   ```

5. Now install Neo4j using the `persistent-volume-selector.yaml` created earlier:

   ```
   helm install "${RELEASE_NAME}" neo4j/neo4j -f persistent-volume-selector.yaml
   ```

6. Clean up the helm installation and disks created for the example:

   ```
   helm uninstall ${RELEASE_NAME} ${RELEASE_NAME}-disk
   kubectl delete pvc data-${RELEASE_NAME}-0
   az disk delete --name ${RELEASE_NAME} -y
   ```

Provision a PVC for Neo4j Storage

An alternative method for manual provisioning is to use a manually provisioned PVC. This is supported by the Neo4j Helm chart using the `volume` mode.

The *neo4j/neo4j-persistent-volume* Helm chart can be used to create a PV and PVC for a manually provisioned disk. A full example can be found in the Neo4j GitHub repository For example, to use a pre-existing PVC called `my-neo4j-pvc` set these values:

```yaml
volumes:
  data:
    mode: "volume"
    volume:
      persistentVolumeClaim:
        claimName: my-neo4j-pvc
```

Reuse a persistent volume

After uninstalling the Neo4j Helm chart, both the PVC and the PV remain and can be reused by a new install of the Helm chart. If you delete the PVC, the PV moves into a `Released` status and will not be reusable.

To be able to reuse the PV by a new install of the Neo4j Helm chart, remove its connection to the previous PVC:

1. Edit the PV by running the following command:

   ```
   kubectl edit pv <pv-name>
   ```

2. Remove the section `spec.claimRef`.
   The PV goes back to the `Available` status and can be reused by a new install of the Neo4j Helm chart.

> The performance of Neo4j is very dependent on the latency, IOPS capacity, and throughput of the storage it is using. For the best performance of Neo4j, use the best available disks (e.g., SSD) and set IOPS throttling/quotas to high values. For some cloud providers, IOPS throttling is proportional to the size of the volume. In these cases, the best performance is achieved by setting the size of the volume based on the desired IOPS rather than the amount required for data storage.

Access a Neo4j standalone server

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

Supported Kubernetes services

The Neo4j Helm chart publishes three K8s services:

- **Default Service** — a ClusterIP service for application `neo4j`/`bolt` and `http(s)` connections to the Neo4j database, originating from inside the Kubernetes cluster.
- **Admin Service** — a "Headless" (DNS only) service that includes all Neo4j ports. It is only available inside the Kubernetes cluster and access to it should be guarded. The Admin service can be used for Neo4j DBMS administration, performing backups, and collecting metrics.
- **Neo4j** — a LoadBalancer service for application `neo4j`/`bolt` and `http(s)` connections originating from outside the Kubernetes cluster.

Table 11. K8s services per Neo4j interface

Neo4j Interface	Default Port	Default Service	Admin Service	Neo4j Service
Bolt (`neo4j://` and `bolt://` protocols)	`7687`	Yes	Yes*	Yes
Neo4j Browser HTTP	`7474`	Yes	Yes*	Yes
Neo4j Browser HTTPS	`7473`	Yes	Yes*	Yes
Neo4j Cypher HTTP API	`7474`	Yes	Yes*	Yes
Neo4j Cypher HTTPS API	`7473`	Yes	Yes*	Yes
Neo4j Backup	`6362`	No	Yes	No but configurable
Graphite Monitoring	`2003`	No	Yes	No
Prometheus Metrics	`2004`	No	Yes	No
Java Management Extensions (JMX)	`3637`	No	No but configurable	No

*The Admin service bypasses health checks. This allows it to be used to make connections for administrative purposes when the database is in an unhealthy state. However, you must not use it to connect from applications that require the database to be in a healthy state.

Applications accessing Neo4j from inside Kubernetes

Access Neo4j using DNS

To access Neo4j from an application in the same Kubernetes cluster use the Neo4j service DNS address `<release-name>.<namespace>.svc.<cluster domain>`.

The default cluster domain is `cluster.local` and the default namespace is `default`. Generally, the Neo4j service DNS address is *<release-name>.default.svc.cluster.local*.

For example, if using the release name `my-release` in the `default` namespace, the cluster's DNS address would be `my-release.default.svc.cluster.local`, and the `bolt` address for use with Neo4j drivers would be *neo4j://my-release.default.svc.cluster.local:7687*.

Access Neo4j using K8s label selector

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

For example:

```
# install neo4j
helm install "my-release" …
# lookup installed service
kubectl get service -l helm.neo4j.com/service=default,helm.neo4j.com/instance=my-release
```

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

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

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

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

Applications accessing Neo4j from outside Kubernetes Enterprise edition

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

- The service name is based on the value of the `neo4j.name` — `<my-neo4j-name>-lb-neo4j`:

  ```
  kubectl get service `<my-neo4j-name>-lb-neo4j` -ocustom-columns=ip:.status.loadBalancer.ingress[].ip
  ```

- Using a label selector:

  ```
  kubectl get service -l helm.neo4j.com/service=neo4j,helm.neo4j.com/name=<release-name> -ocustom
  -columns=ip:.status.loadBalancer.ingress[].ip
  ```

If the Kubernetes `LoadBalancer` implementation that you are using supports setting a static IP, the IP address of the `LoadBalancer` can be configured in the Neo4j Helm release by setting `externalService.loadBalancerIP`. If a static IP address is not explicitly set, then Kubernetes does not guarantee that a dynamically assigned IP address will not change.

When exposing a Neo4j database on the Internet, it is recommended to use a static IP and configure SSL on the exposed services. For more information, see Configure SSL.

If you have static IPs, you can associate DNS with them and obtain trusted certificates.

The ports that are exposed on the external service can be configured in the Helm release by changing the `services.neo4j` object. The default values are:

```yaml
services:
  neo4j:
     annotations: { }
     loadBalancerIP: NULL
     ports:
       http:
         enabled: true
       https:
         enabled: true
       bolt:
         enabled: true
       backup:
         enabled: false
```

Disabling/enabling a port on the `services.neo4j` object removes it from the load balancer but does not affect whether it is disabled/enabled in Neo4j.

Backup is not secure unless SSL-with-client-auth is enforced in the Neo4j configuration.

Customizing Kubernetes Resources

The Neo4j Helm chart creates various Kubernetes resources. Some of them can be customized by adding extra configuration to the helm deployment values file.

Table 12. Supported K8s resources customizations

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

Accessing Neo4j for DBMS administration and monitoring

The Neo4j Helm chart creates the admin service for the purposes of Neo4j administration. The admin service is a “Headless” service in Kubernetes and does not depend on Neo4j health checks. Therefore, it permits connections to Neo4j even if Neo4j is not healthy. In general, that is not desirable for applications but can be useful for administration and debugging.

Access Neo4j using DNS

To access the admin service inside Kubernetes use the DNS address *<release-name>-admin.<namespace>.svc.<cluster domain>*.

For example, if using the release name `my-release` in the `default` namespace, the cluster's DNS address would be `my-release-admin.default.svc.cluster.local`.

The admin service can be used to access a range of Neo4j interfaces:

- Neo4j Bolt for Neo4j administration via Cypher commands
- Neo4j Backup for taking database backups
- Graphite for metrics collection
- Prometheus for metrics collection
- Java Management Extensions (JMX) for metrics collection and JVM administration

Access Neo4j using `kubectl` for troubleshooting

To get an interactive `cypher-shell` console for troubleshooting, use this command:

```
kubectl run -it --rm --image neo4j:5.4.0 cypher-shell -- cypher-shell -a bolt://my-release-
admin.default.svc.cluster.local
```

Generally, the `neo4j://` protocol is used for connecting to Neo4j. For troubleshooting, though, the direct `bolt://` protocol is used because it allows a connection in some situations where a `neo4j://` connection will not succeed.

Access a Neo4j cluster

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

Supported Kubernetes services

The Neo4j Helm chart publishes four K8s services:

- **Default Service** — a ClusterIP service for application `neo4j`/`bolt` and `http(s)` connections to the Neo4j database, originating from inside the Kubernetes cluster.
- **Admin Service** — a "Headless" (DNS only) service that includes all Neo4j ports for admin connections to Neo4j inside Kubernetes. It is only available inside the Kubernetes cluster and access to it should be guarded. The Admin service can be used for Neo4j DBMS administration, performing backups, and collecting metrics.
- **Internal Service** — a "Headless" (DNS only) internal service that includes all Neo4j ports required for causal clustering.
- **Neo4j** — a LoadBalancer service for application `neo4j`/`bolt` and `http(s)` connections originating from outside the Kubernetes cluster.

Table 13. K8s services per Neo4j interface

Neo4j Interface	Default Port	Default Service	Admin Service	Internal Service	Neo4j Service
Bolt (`neo4j://` and `bolt://` protocols)	`7687`	Yes	Yes*	Yes	Yes
Neo4j Browser HTTP	`7474`	Yes	Yes*	Yes	Yes
Neo4j Browser HTTPS	`7473`	Yes	Yes*	Yes	Yes
Neo4j Cypher HTTP API	`7474`	Yes	Yes*	Yes	Yes
Neo4j Cypher HTTPS API	`7473`	Yes	Yes*	Yes	Yes
Neo4j Backup	`6362`	No	Yes	Yes	No
Graphite Monitoring	`2003`	No	No but configurable	No but configurable	No
Prometheus Metrics	`2004`	No	No but configurable	No but configurable	No
Java Management Extensions (JMX)	`3637`	No	No but configurable	No but configurable	No
Cluster discovery management	`5000`	No	No	Yes	No
Cluster transaction	`6000`	No	No	Yes	No
Cluster RAFT	`7000`	No	No	Yes	No
Cluster routing connector	`7688`	No	No	Yes	No

*The Admin service bypasses health checks. This allows it to be used to make connections for administrative purposes when the database is in an unhealthy state. However, you must not use it to connect from applications that require the database to be in a healthy state.

Applications accessing Neo4j from inside Kubernetes

Access Neo4j using DNS

To access Neo4j from an application in the same Kubernetes cluster use the Neo4j service DNS address `<release-name>.<namespace>.svc.<cluster domain>`.

The default cluster domain is `cluster.local` and the default namespace is `default`. Generally, the Neo4j service DNS address is *<release-name>.default.svc.cluster.local*.

For example, if using the release name `my-release` in the `default` namespace, the cluster's DNS address would be `my-release.default.svc.cluster.local`, and the `bolt` address for use with Neo4j drivers would be *neo4j://my-release.default.svc.cluster.local:7687*.

To allow for an application running inside Kubernetes to access the Neo4j cluster, you can also use the

Neo4j headless service that is installed via the *neo4j/neo4j-cluster-headless-service* Helm chart. For more information and a detailed example, see Access the Neo4j cluster using headless service.

Access Neo4j using K8s label selector

Alternatively, the Neo4j service (default) in Kubernetes can be located using Kubernetes service discovery by searching with the label selector: `helm.neo4j.com/service=default/admin/internals,helm.neo4j.com/instance=<release-name>`.

The following is an example of how to look up the installed services:

```
# Neo4j service:
kubectl get service -l helm.neo4j.com/service=default,helm.neo4j.com/instance=my-release
# Admin service:
kubectl get service -l helm.neo4j.com/service=admin,helm.neo4j.com/instance=my-release
# internals service:
kubectl get service -l helm.neo4j.com/service=internals,helm.neo4j.com/instance=my-release
```

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

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

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

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

Applications accessing Neo4j from outside Kubernetes

To access a Neo4j cluster from an application outside the Kubernetes cluster, you can use a LoadBalancer service. For more information, see Access the Neo4j cluster from outside Kubernetes.

Customizing Kubernetes Resources

The Neo4j Helm chart creates various Kubernetes resources. Some of them can be customized by adding extra configuration to the helm deployment values file.

Table 14. Supported K8s resources customizations

Customization	*values.yaml* **field**	**Type**
Setting a pod securityContext for the Neo4j Pod	`securityContext`	`PodSecurityContext`

Customization	*values.yaml* field	Type
Adding annotations to Services	`services.neo4j.annotations`	Annotations object for `ClusterIP` service.
	`services.admin.annotations`	Annotations object for headless (DNS) service.
	`services.internal.annotations`	Annotations object for internal service.
Adding annotations to Load Balancer Service	`annotations`	Annotations object for `LoadBalancer` service.

Accessing Neo4j for DBMS administration and monitoring

The Neo4j Helm chart creates the admin service for the purposes of Neo4j administration. The admin service is a “Headless” service in Kubernetes and does not depend on Neo4j health checks. Therefore, it permits connections to Neo4j even if Neo4j is not healthy. In general, that is not desirable for applications but can be useful for administration and debugging.

Access Neo4j using DNS

To access the admin service inside Kubernetes use the DNS address *<release-name>-admin.<namespace>.svc.<cluster domain>*.

For example, if using the release name `my-release` in the `default` namespace, the cluster’s DNS address would be `my-release-admin.default.svc.cluster.local`.

The admin service can be used to access a range of Neo4j interfaces:

- Neo4j Bolt for Neo4j administration via Cypher commands.
- Neo4j Backup for taking database backups.
- Graphite for metrics collection.
- Prometheus for metrics collection.
- Java Management Extensions (JMX) for metrics collection and JVM administration.

Access Neo4j using `kubectl` for troubleshooting

To get an interactive `cypher-shell` console for troubleshooting, use this command:

```
kubectl run -it --rm --image neo4j:5.4.0 cypher-shell -- cypher-shell -a bolt://my-release-
admin.default.svc.cluster.local
```

Generally, the `neo4j://` protocol is used for connecting to Neo4j. For troubleshooting, though, the direct `bolt://` protocol is used because it allows a connection in some situations where a `neo4j://` connection will not succeed.

Import Data

There is a wide range of ways to import data from files into a Neo4j instance. This page describes the most common ways to import data into a Neo4j instance running on a Kubernetes cluster.

Importing data into Neo4j on Kubernetes

The Neo4j Helm chart configures a volume mount at */import* as the Neo4j *import* directory, as described in Default file locations. You place all the files that you want to import in this volume.

To import data from CSV files into Neo4j, use the command `neo4j-admin database import` or the Cypher query `LOAD CSV`.

- The `neo4j-admin database import` command can be used to do batch imports of large amounts of data into a previously unused database and can only be performed once per database.
- `LOAD CSV` Cypher statement can be used to import small to medium-sized CSV files into an existing database. `LOAD CSV` can be run as many times as needed and does not require an empty database. For a simple example, see Getting Started Guide → Import data.

> Depending on your Neo4j configuration, some methods support fetching data to import from a remote location (e.g., using HTTP or fetching from cloud object storage). Therefore, it is not always necessary to place the source data files in the Neo4j *import* directory.

Configure the import volume mount

The default configuration of the `/import` volume mount is to share the `/data` volume mount. Generally, this is sufficient, and it is unnecessary to explicitly configure an *import* volume in the Helm deployment's *values.yaml* file. For the full details of configuring volume mounts for a Neo4j Helm deployment, see Volume mounts and persistent volumes.

This example shows how to configure `/import` to use a dynamically provisioned Persistent Volume of the default `StorageClass`:

```yaml
volumes:
  import:
    mode: "defaultStorageClass"
    defaultStorageClass:
      requests:
        storage: 100Gi
```

Copy files to the *import* volume using `kubectl cp`

Files can be copied to the *import* volume using `kubectl cp`. This example shows how to copy a local directory `my-files/` to `/import/files-1` to a Neo4j instance with the release name `my-graph-db` in the namespace `default`.

```
kubectl cp my-files/ default/my-graph-db-0:/import/files-1

# Validate: list the contents of /import/files-1
kubectl exec my-graph-db-0 -- ls /import/files-1
```

Instead of using `kubectl cp`, data can also be loaded onto the `/import` directory by:

- using an additional container or `initContainer` to load data.
- using `kubectl exec` to run commands to load data.
- mounting a volume that is already populated with data.

> Data must be placed in the volume's `/import` directory.

Use `neo4j-admin database import`

The simplest way to run `neo4j-admin database import` is to use `kubectl exec` to run it in the Neo4j container. However, running `neo4j-admin database import` to perform a large import in the same container as the Neo4j process may cause resource contention problems, including causing either or both processes to be OOM Killed by the node operating system. To avoid this, either use a separate container or `initContainer` or place the Neo4j Helm deployment in offline maintenance mode to run `neo4j-admin database import`.

`neo4j-admin database import` cannot be used to replace an existing database while Neo4j is running. To replace an existing database, either `DROP` the database or put the Neo4j Helm deployment into offline maintenance mode before running `neo4j-admin database import`.

Alternative approach

An alternative approach to importing data into Neo4j is to run a separate Neo4j standalone instance outside Kubernetes, perform the import on that Neo4j instance, and then copy the resulting database into the Kubernetes-based Neo4j instance using the backup and restore or dump and load procedures.

Monitoring

You can monitor a Neo4j DBMS running on Kubernetes using the same mechanisms as you would for a Neoj4 running on-prem.

Logging

When using the Helm chart, Neo4j logging output is written to files in the `/logs` directory. This directory is mounted on a `PersistentVolume` so that logs are persisted if the pod is moved or restarted. For full details of Neo4j logging, see Neo4j logging.

- To view the Neo4j user log (*neo4j.log*), use the command `kubectl exec`:

Follow neo4j.log

```
kubectl exec <neo4j-pod-name> -- tail -f /logs/neo4j.log
```

- To copy the log files from a Neo4j instance, use `kubectl cp`:

  Copy all logs

```
$ kubectl cp <neo4j-pod-name>:/logs neo4j-logs/
$ ls neo4j-logs
debug.log       neo4j.log       query.log       security.log
```

Log collection

The Neo4j log output can be collected from the log files and sent to a unified location using tools, such as Fluentd (https://www.fluentd.org) or Logstash (https://www.elastic.co/logstash). We recommend running these either as "sidecar" containers in the Neo4j pods or as separate DaemonSets.

- For more information about Pods and the sidecar pattern, see Kubernetes Pod documentation.
- For more information about DaemonSets, see Kubernetes DaemonSet documentation.
- For more information and examples of these logging patterns, see Kubernetes cluster administration documentation.

Metrics

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

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

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

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

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

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

For more information and examples, see Neo4j metrics.

Operations

Neo4j supports two maintenance modes: online and offline, which you can use to perform different maintenance tasks.

Online Maintenance

Online maintenance does not require stopping the `neo4j` process. It is performed using the command `kubectl exec`.

To directly run tasks:

```
kubectl exec <release-name>-0 -- neo4j-admin database info --from-path=/var/lib/neo4j/data/databases
--expand-commands
```

All `neo4j-admin` commands need the `--expand-commands` flag to run in the Neo4j container. This is because the Neo4j Helm chart defines the Neo4j configuration using command expansion to dynamically resolve some configuration parameters at runtime.

To run a series of commands, use an interactive shell:

```
kubectl exec -it <release-name>-0 -- bash
```

Processes executed using `kubectl exec` count towards the Neo4j container's memory allocation. Therefore, running tasks that use a significant amount of memory or running Neo4j in an extremely memory-constrained configuration could cause the Neo4j container to be terminated by the underlying Operating System.

Offline Maintenance

You use the Neo4j offline maintenance mode to perform maintenance tasks that require Neo4j to be offline. In this mode, the `neo4j` process is not running. However, the Neo4j Pod does run, but it never reaches the status `READY`.

Put the Neo4j instance in offline mode

1. To put the Neo4j instance in offline maintenance mode, you set the `offlineMaintenanceModeEnabled: true` and upgrade the helm release.
    - You can do that by using the *values.yaml* file:

a. Open your *values.yaml* file and add `offlineMaintenanceModeEnabled: true` to the `neo4j` object:

```
neo4j:
 offlineMaintenanceModeEnabled: true
```

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

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

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

```
helm upgrade <release-name> neo4j/neo4j --version=5.4.0 --reuse-values --set
neo4j.offlineMaintenanceModeEnabled=true
```

2. Poll `kubectl get pods` until the pod has restarted (`STATUS`=`Running`).

```
kubectl get pod <release-name>-0
```

3. Connect to the pod with an interactive shell:

```
kubectl exec -it "<release-name>-0" -- bash
```

4. View running java processes:

```
jps
```

```
19 Jps
```

The result shows no running java process other than `jps` itself.

Run task in offline mode

Offline maintenance tasks are performed using the command `kubectl exec`.

- To directly run tasks:

```
kubectl exec <release-name>-0 -- neo4j-admin database info --from-path=/var/lib/neo4j/data/databases
--expand-commands
```

- To run a series of commands, use an interactive shell:

```
kubectl exec -it <release-name>-0 -- bash
```

- For long-running commands, use a shell and run tasks using `nohup` so they continue if the `kubectl exec` connection is lost:

```
kubectl exec -it <release-name>-0 -- bash
  $ nohup neo4j-admin database check neo4j --expand-commands &>job.out </dev/null &
  $ tail -f job.out
```

Put the Neo4j DBMS in online mode

When you finish with the maintenance tasks, return the Neo4j instance to normal operation:

- You can do that by using the *values.yaml* file:

  1. Open your *values.yaml* file and add `offlineMaintenanceModeEnabled: false` to the `neo4j` object:

     ```
     neo4j:
      offlineMaintenanceModeEnabled: false
     ```

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

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

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

```
helm upgrade <release-name> neo4j/neo4j-standalone --version=5.4.0 --reuse-values --set
neo4j.offlineMaintenanceModeEnabled=false
```

Reset the `neo4j` user password

You reset the `neo4j` user password by disabling authentication and then re-enabling it.

1. In the *values.yaml* file, set `dbms.security.auth_enabled:` to `false` to disable the authentication:

   All Neo4j `config` values must be YAML strings, not YAML booleans. Therefore, make sure you put quotes around values, such as `"true"` or `"false"`, so that they are handled correctly by Kubernetes.

   ```
   # Neo4j Configuration (yaml format)
   config:
     dbms.security.auth_enabled: "false"
   ```

2. Run the following command to apply the changes:

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

   Authentication is now disabled.

3. Connect with `cypher-shell` and set the desired password:

   ```
   ALTER USER neo4j SET PASSWORD '<new-password>'
   ```

4. Update the Neo4j configuration to enable authentication:

```
# Neo4j Configuration (yaml format)
config:
  dbms.security.auth_enabled: "true"
```

5. Run the following command to apply the update and re-enable authentication:

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

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

Dump and load databases (offline)

You can use the `neo4j-admin database dump` command to make a full backup (an archive) of an **offline** database(s) and `neo4j-admin database load` to load it back into a Neo4j deployment. These operations are performed in offline maintenance mode.

Dump the `neo4j` and `system` databases

1. Put the Neo4j instance in offline mode.
2. Dump `neo4j` and `system` databases:

```
neo4j-admin database dump --expand-commands system --to-path=/backups && neo4j-admin database dump
--expand-commands neo4j --to-path=/backups
```

3. Put the Neo4j DBMS in online mode.
4. Verify that Neo4j is working by refreshing Neo4j Browser.

For information about the command syntax, options, and usage, see Back up an offline database.

Load the `neo4j` and `system` databases

1. Put the Neo4j instance in offline mode.
2. Run `neo4j-admin database load` commands:

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

   For information about the command syntax, options, and usage, see Restore a database dump.

3. Put the Neo4j DBMS in online mode.
4. Verify that Neo4j is working by refreshing Neo4j Browser.

Back up and restore a single database (online) Enterprise edition

You can use the `neo4j-admin database backup` command to make a full or differential backup of an **online** database(s) and `neo4j-admin database restore` to restore it in a live Neo4j DBMS or cluster. These operations are performed in online maintenance mode.

For performing backups, Neo4j uses the *Admin Service*, which is only available inside the Kubernetes cluster and access to it should be guarded. For more information, see Access the Neo4j cluster from inside Kubernetes and Access the Neo4j cluster from outside Kubernetes.

Back up a single database

The `neo4j-admin database backup` command can be run both from the same and a separate pod. However, it uses resources (CPU, RAM) in the Neo4j container (competing with Neo4j itself), because it checks the database consistency at the end of every backup operation. Therefore, it is recommended to run the operation in a separate pod.

In the Neo4j Helm chart, the backup configurations are set by default to `server.backup.enabled=true` and `server.backup.listen_address=0.0.0.0:6362`.

Note that the default for Neo4j on-site installations is to listen only on 127.0.0.1, which will not work from other containers, since they would not be able to access the backup port.

Back up a database from a separate pod

1. Create a Neo4j instance pod to get access to the `neo4j-admin` command:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: neo4j-backup
spec:
  containers:
    - name: neo4j-backup
      image: neo4j:5.4.0-enterprise
      imagePullPolicy: IfNotPresent
      env:
        - name: NEO4J_ACCEPT_LICENSE_AGREEMENT
          value: "yes"
      volumeMounts:
        - mountPath: /backup
          name: backups
  volumes:
    - name: backups
      emptyDir: {}
EOF
```

2. Run the following command to back up the database you want. In this example, this is the `neo4j` database. The command is the same for standalone instances and Neo4j cluster members.

```
kubectl exec -ti neo4j-backup -- neo4j-admin database backup --from=my-neo4j-release-admin:6362 --to
-path=/backups --expand-commands neo4j
```

3. Finally, copy the backup from the Pods volume mount to your local machine so that it can be moved to a resilient backup location such as S3:

```
kubectl cp neo4j-backup:/backup /tmp/backup
```

Restore a single database

To restore a single offline database or a database backup, you first need to delete the database that you want to replace unless you want to restore the backup as an additional database in your DBMS. Then, use the restore command of `neo4j-admin` to restore the database backup. Finally, use the Cypher command `CREATE DATABASE name` to create the restored database in the `system` database.

Delete the database that you want to replace

Before you restore the database backup, you have to delete the database that you want to replace with that backup using the Cypher command `DROP DATABASE name` against the `system` database. If you want to restore the backup as an additional database in your DBMS, then you can proceed to the next section.

For Neo4j cluster deployments, you run the Cypher command `DROP DATABASE name` only on one of the cluster servers. The command is automatically routed from there to the other cluster members.

1. Connect to the Neo4j DBMS:

```
kubectl exec -it <release-name>-0 -- bash
```

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

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

3. Drop the database you want to replace with the backup:

```
DROP DATABASE neo4j;
```

4. Exit the Cypher Shell command-line console:

```
:exit;
```

Restore the database backup

You use the `neo4j-admin database restore` command to restore the database backup, and then the Cypher command `CREATE DATABASE name` to create the restored database in the `system` database. For information about the command syntax, options, and usage, see Restore a database backup.

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

1. Run the `neo4j-admin database restore` command to restore the database backup:

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

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

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

3. Create the `neo4j` database.

   For Neo4j cluster deployments, you run the Cypher command `CREATE DATABASE name` only on one of the cluster servers.

   ```
   CREATE DATABASE neo4j;
   ```

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

5. Execute a Cypher command against the `neo4j` database, for example:

   ```
   MATCH (n) RETURN n
   ```

   If you have backed up your database with the option `--include-metadata`, you can manually restore the users and roles metadata. For more information, see Restore a database backup → Example.

To restore the `system` database, follow the steps described in Dump and load databases (offline).

Upgrade Neo4j Community to Enterprise edition

To upgrade from Neo4j Community to Enterprise edition, run:

```
helm upgrade <release-name> neo4j/neo4j --reuse-values --set neo4j.edition=enterprise --set
neo4j.acceptLicenseAgreement=yes
```

To upgrade to the next patch release of Neo4j, update your Neo4j *values.yaml* file and upgrade the helm release.

1. Open the *values.yaml* file, using the code editor of your choice, and add the following line to the `image` object:

   ```
   image:
     customImage: neo4j:5.4.0
   ```

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

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

Migrate Neo4j from the Labs Helm charts to the Neo4j Helm charts (offline)

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

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

Single instance

- From the Labs Helm chart 3.5 or earlier to either the Neo4j Helm chart 4.3 or 4.4 — upgrade your Neo4j deployment to whichever version you want to move to, using the steps in the https://neo4j.com/labs/neo4j-helm/1.0.0/ and then migrate from the Labs Helm chart (4.3 or 4.4) to the Neo4j Helm chart 4.3 or 4.4 using the steps described here.
- From the Labs Helm chart 4.3 to the Neo4j Helm chart 4.3 — follow the steps described here.
- From the Labs Helm chart 4.3 to the Neo4j Helm chart 4.4 — follow the steps described here.

Cluster

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

Back up a Neo4j deployment created with the Labs Helm chart

To back up your Neo4j deployment created with the Labs Helm chart, follow the steps in the Neo4j-Helm User Guide → Backing up Neo4j Containers.

Restore your backup into a standalone or a cluster created with the Neo4j Helm chart

If the backup exists on a cloud provider, you can take one of the following approaches:

Approach 1

1. Create a standalone or a cluster using the Neo4j Helm chart with a custom Neo4j image that has all the cloud provider utilities to download the backup from the respective cloud provider storage to your specific mount.
2. Restore the backup following the steps described in Restore a single database.

Approach 2

1. Get the backup on your local machine.
2. Copy the backup to the respective mount in your new cluster created using the Neo4j Helm chart, using the command `kubectl cp <local-path> <pod>:<path>`. For example,

   ```
   kubectl cp /Users/username/Desktop/backup/4.3.3/neo4j standalone-0:/tmp/
   ```

   where the */tmp* directory refers to the mount.
3. Restore the backup following the steps described in Restore a single database.

Scale a Neo4j deployment

Neo4j supports both vertical and horizontal scaling.

Vertical scaling

To increase or decrease the resources (CPU, memory) available to a Neo4j instance, change the `neo4j.resources` object in the *values.yaml* file to set the desired resource usage, and then perform a helm upgrade.

> If you change the memory allocated to the Neo4j container, you should also change the Neo4j's memory configuration (`server.memory.heap.initial_size` and `server.memory.pagecache.size` in particular). See Configure Resource Allocation for more details.

For example, if your running Neo4j instance has the following allocated resources:

```
# values.yaml
neo4j:
  resources:
    cpu: "1"
    memory: "3Gi"

# Neo4j Configuration (yaml format)
config:
  server.memory.heap.initial_size: "2G"
  server.memory.heap.initial_size: "2G"
  server.memory.pagecache.size: "500m"
```

And, you want to increase them to 2 CPUs and 4 GB of memory (allocating additional memory to the pagecache).

1. Modify the *values.yaml* file to set the desired resource usage:

```
# values.yaml
neo4j:
  resources:
    cpu: "2"
    memory: "4Gi"

# Neo4j Configuration (yaml format)
config:
  server.memory.heap.initial_size: "2G"
  server.memory.heap.initial_size: "2G"
  server.memory.pagecache.size: "1G"
```

2. Run `helm upgrade` with the modified deployment *values.yaml* file and the Neo4j Helm chart to apply the changes. For example:

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

Horizontal scaling Enterprise edition

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

Example — add a new server to an existing cluster

The following example assumes that you have a cluster with 3 servers.

1. In the Kubernetes cluster, verify that you have a node that you can use for the new server, `server4`.
2. Install `server4` using the same value for `neo4j.name` as your existing cluster:

   ```
   helm install server4 neo4j --set neo4j.edition=enterprise --set neo4j.acceptLicenseAgreement=yes
   --set volumes.data.mode=defaultStorageClass --set neo4j.password="password" --set
   neo4j.minimumClusterSize=3 --set neo4j.name=my-cluster
   ```

   Alternatively, you can use a *values.yaml* file to set the values for the new server and the *neo4j/neo4j* Helm chart to install the new server. For more information, see Create Helm deployment values files and Install Neo4j cluster servers.

   When the new server joins the cluster, it will initially be in the `Free` state.

3. Enable `server4` to be able to host databases by using `cypher-shell` (or Neo4j Browser) to connect to one of the existing servers:

   a. Access the cypher-shell on `server1`:

   ```
   kubectl exec -ti server1-0 -- cypher-shell -u neo4j -p password -d neo4j
   ```

   b. When the `cypher-shell` prompt is ready, verify that `server4` is in the `Free` state, and take a note of its name:

   ```
   SHOW SERVERS;
   ```

```
+---------------------------------------------------------------------------------------------------
-----------------------------------+
| name                                 | address                                 | state     |
health       | hosting              |
+---------------------------------------------------------------------------------------------------
-----------------------------------+
| "0908819d-238a-473d-9877-5cc406050ea2" | "server4.neo4j.svc.cluster.local:7687" | "Free"    |
"Available" | ["system"]           |
| "19817354-5cd1-4579-8c45-8b897808fdb4" | "server2.neo4j.svc.cluster.local:7687" | "Enabled" |
"Available" | ["system", "neo4j"] |
| "b3c91592-1806-41d0-9355-8fc6ba236043" | "server3.neo4j.svc.cluster.local:7687" | "Enabled" |
"Available" | ["system", "neo4j"] |
| "eefd7216-6096-46f5-9c41-a74f79684172" | "server1.neo4j.svc.cluster.local:7687" | "Enabled" |
"Available" | ["system", "neo4j"] |
+---------------------------------------------------------------------------------------------------
-----------------------------------+
```

4. Using its name, enable `server4` to use it in the cluster:

```
ENABLE SERVER "0908819d-238a-473d-9877-5cc406050ea2";
```

5. Run `SHOW SERVERS;` again to verify that `server4` is enabled:

```
SHOW SERVERS;
```

```
+---------------------------------------------------------------------------------------------------
-------------------------------+
| name                                 | address                                 | state     |
health       | hosting              |
+---------------------------------------------------------------------------------------------------
-------------------------------+
| "0908819d-238a-473d-9877-5cc406050ea2" | "server4.neo4j.svc.cluster.local:7687" | "Enabled" |
"Available" | ["system"]           |
| "19817354-5cd1-4579-8c45-8b897808fdb4" | "server2.neo4j.svc.cluster.local:7687" | "Enabled" |
"Available" | ["system", "neo4j"] |
| "b3c91592-1806-41d0-9355-8fc6ba236043" | "server3.neo4j.svc.cluster.local:7687" | "Enabled" |
"Available" | ["system", "neo4j"] |
| "eefd7216-6096-46f5-9c41-a74f79684172" | "server1.neo4j.svc.cluster.local:7687" | "Enabled" |
"Available" | ["system", "neo4j"] |
+---------------------------------------------------------------------------------------------------
-------------------------------+
```

Notice in the output that although `server4` is now enabled, it is not hosting the `neo4j` database. You need to change the database topology to also use the new server.

6. Alter the `neo4j` database topology to be hosted on three primary and one secondary servers:

```
ALTER DATABASE neo4j SET TOPOLOGY 3 PRIMARIES 1 SECONDARY;
```

7. Now run the `SHOW DATABASES;` command to verify the new topology:

```
SHOW DATABASES;
```

```
+-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------+
| name     | type       | aliases | access       | address                          | role
| writer | requestedStatus | currentStatus | statusMessage | default | home   | constituents |
+-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------+
| "neo4j"  | "standard" | []      | "read-write" | "server2.neo4j.svc.cluster.local:7687" |
"primary"   | TRUE   | "online"        | "online"      | ""              | TRUE    | TRUE  | []
|
| "neo4j"  | "standard" | []      | "read-write" | "server4.neo4j.svc.cluster.local:7687" |
"secondary" | FALSE  | "online"        | "online"      | ""              | TRUE    | TRUE  | []
|
| "neo4j"  | "standard" | []      | "read-write" | "server3.neo4j.svc.cluster.local:7687" |
"primary"   | FALSE  | "online"        | "online"      | ""              | TRUE    | TRUE  | []
|
| "neo4j"  | "standard" | []      | "read-write" | "server1.neo4j.svc.cluster.local:7687" |
"primary"   | FALSE  | "online"        | "online"      | ""              | TRUE    | TRUE  | []
|
| "system" | "system"   | []      | "read-write" | "server2.neo4j.svc.cluster.local:7687" |
"primary"   | FALSE  | "online"        | "online"      | ""              | FALSE   | FALSE | []
|
| "system" | "system"   | []      | "read-write" | "server4.neo4j.svc.cluster.local:7687" |
"primary"   | FALSE  | "online"        | "online"      | ""              | FALSE   | FALSE | []
|
| "system" | "system"   | []      | "read-write" | "server3.neo4j.svc.cluster.local:7687" |
"primary"   | TRUE   | "online"        | "online"      | ""              | FALSE   | FALSE | []
|
| "system" | "system"   | []      | "read-write" | "server1.neo4j.svc.cluster.local:7687" |
"primary"   | FALSE  | "online"        | "online"      | ""              | FALSE   | FALSE | []
|
+-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------+
```

Note that `server4` now hosts the `neo4j` database with the `secondary` role.

Use custom images from private registries

From v4.4.4, Neo4j supports using custom images from private registries by adding new or existing `imagePullSecrets`.

Add an existing `imagePullSecret`

You can use an existing `imagePullSecret` for your Neo4j deployment by specifying its name in the *values.yaml* file. The Neo4j Helm chart checks if the provided `imagePullSecret` exists in the Kubernetes cluster and uses it. If a secret with the given name does not exist in the cluster, Helm chart throws an error.

Using an already existing secret **mysecret**

```
# values.yaml
# Override image settings in Neo4j pod
image:
  imagePullPolicy: IfNotPresent
  # set a customImage if you want to use your own docker image
  customImage: demo_neo4j_image:v1

  #imagePullSecrets list
  imagePullSecrets:
      - "mysecret"
```

Create and add a new `imagePullSecret`

You can create a new `imagePullSecret` for your Neo4j deployment by defining an equivalent imageCredential in the *values.yaml* file.

The Neo4j Helm chart creates a secret with the given name and uses it as an imagePullSecret to pull the custom image defined. The following example shows how to define a private docker registry imageCredential with the name mysecret.

Creating and adding mysecret as the imagePullSecret to the cluster.

```
# values.yaml
# Override image settings in Neo4j pod
image:
  imagePullPolicy: IfNotPresent
  # set a customImage if you want to use your own docker image
  customImage: demo_neo4j_image:v1

  #imagePullSecrets list
  imagePullSecrets:
      - "mysecret"

  #imageCredentials list for which secret of type docker-registry will be created automatically using the
details provided
  # registry, username, password, email are compulsory fields for an imageCredential, without any, helm
chart will throw an error
  # imageCredential name should be part of the imagePullSecrets list or else the respective
imageCredential will be ignored and no secret creation will be done
  imageCredentials:
    - registry: "https://index.docker.io/v1/"
      username: "demouser"
      password: "demopass123"
      email: "demo@company1.com"
      name: "mysecret"
```

Assign Neo4j pods to specific nodes

The Neo4j Helm chart provides support for assigning your Neo4j pods to specific nodes using nodeSelector labels.

You specify the nodeSelector labels in the *values.yaml* file.

If there is no node with the given labels, the Helm chart will throw an error.

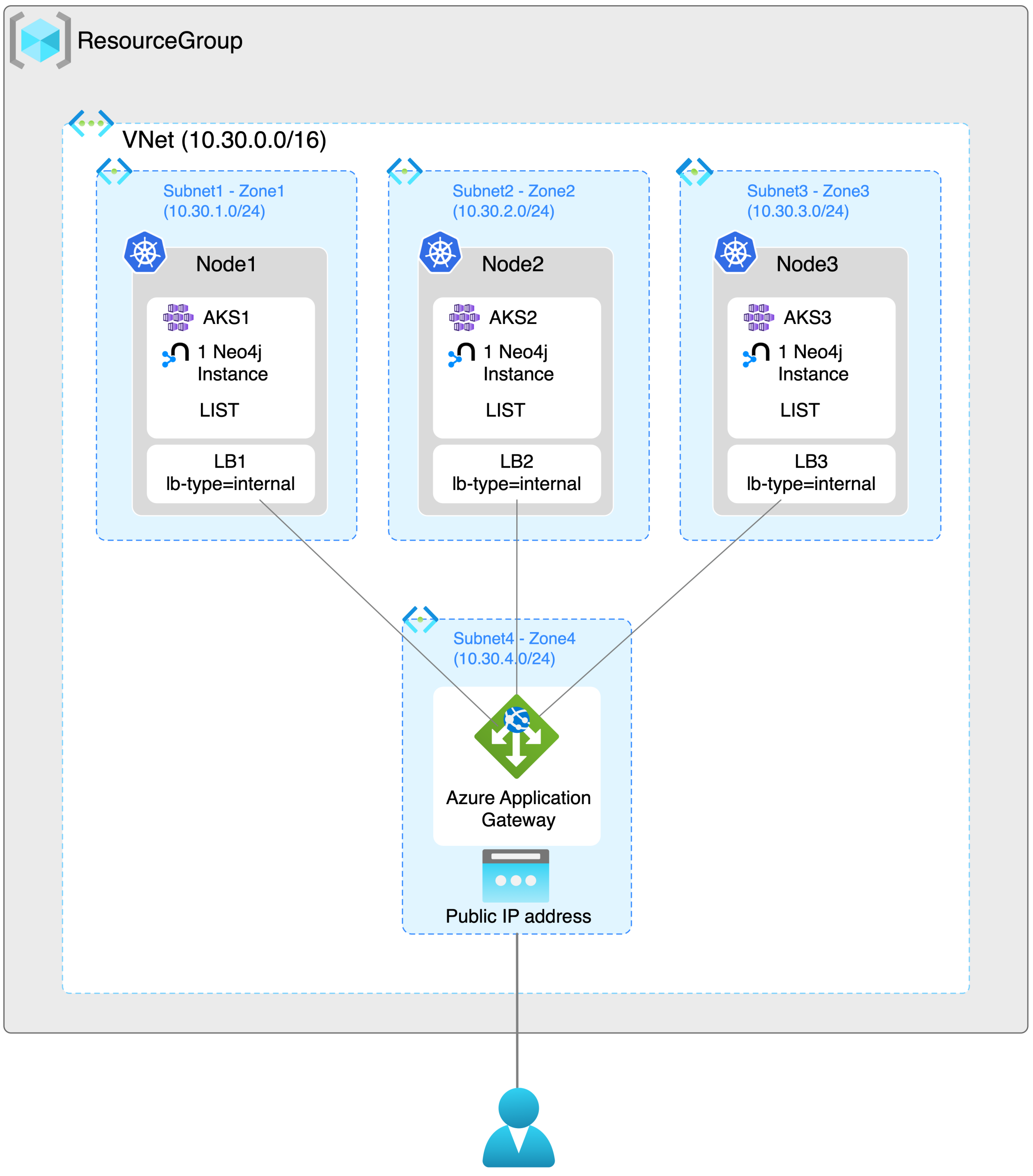
nodeSelector labels in values.yaml

```
#nodeSelector labels
#Ensure the respective labels are present on one of the cluster nodes or else Helm chart will throw an
error.
nodeSelector:
   nodeNumber: one
   name: node1
```

Deploy a single Neo4j cluster across multiple AKS clusters

With the Neo4j Helm chart, you can deploy a Neo4j cluster on multiple AKS clusters using load balancers and an Azure application gateway.

The following diagram is a schematic representation of a Neo4j cluster with three primary servers running on three different AKS clusters.



The diagram shows three Neo4j instances, each running on a different AKS cluster in a different availability zone as part of a single Neo4j cluster. Each AKS cluster also includes an internal load balancer for each Neo4j instance and a LIST discovery method. They allow the Neo4j instances to communicate with each other. The Neo4j cluster can be accessed from outside Kubernetes using an Azure application gateway.

The following steps are an example of how to deploy a Neo4j cluster on a multi-AKS cluster.

Create three AKS clusters in three availability zones

You must have the `Microsoft.Authorization/roleAssignments/write` permission to perform these tasks.

1. Install the `az` command-line interface (CLI) (https://docs.microsoft.com/en-us/cli/azure/install-azure-cli).
2. Create a resource group to host your virtual network. This example creates a resource group named

my-RG in the eastus location:

```
az group create \
    --name my-RG \
    --location eastus
```

Example output

```
{
  "id": "/subscriptions/5b9ae547-ce82-4834-b276-b72904ceaa84/resourceGroups/my-RG",
  "location": "eastus",
  "managedBy": null,
  "name": "my-RG",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null,
  "type": "Microsoft.Resources/resourceGroups"
}
```

3. In this resource group, create an Azure Virtual Network (VNet). This example creates a virtual network named my-VNet with the virtual network's address range 10.30.0.0/16:

```
az network vnet create \
  --name my-VNet \
  --resource-group my-RG \
  --address-prefixes 10.30.0.0/16
```

Example output

```
{
  "newVNet": {
    "addressSpace": {
      "addressPrefixes": [
        "10.30.0.0/16"
      ]
    },
    "bgpCommunities": null,
    "ddosProtectionPlan": null,
    "dhcpOptions": {
      "dnsServers": []
    },
    "enableDdosProtection": false,
    "enableVmProtection": null,
    "encryption": null,
    "etag": "W/\"97953f32-55fe-4821-aedd-ec7a800127e3\"",
    "extendedLocation": null,
    "flowTimeoutInMinutes": null,
    "id": "/subscriptions/5b9ae547-ce82-4834-b276-b72904ceaa84/resourceGroups/my-
RG/providers/Microsoft.Network/virtualNetworks/my-VNet",
    "ipAllocations": null,
    "location": "eastus",
    "name": "my-VNet",
    "provisioningState": "Succeeded",
    "resourceGroup": "my-RG",
    "resourceGuid": "4ed2a9f4-580e-4424-800b-1065ed9ad0a2",
    "subnets": [],
    "tags": {
      "Owner Department": "Engineering - Neo4j"
    },
    "type": "Microsoft.Network/virtualNetworks",
    "virtualNetworkPeerings": []
  }
}
```

4. Add four subnets to the virtual network you have created (my-VNet). They will be used by the Azure

resources you will deploy on each AKS cluster. The subnet address range must be unique within the address space for the virtual network.

```
az network vnet subnet create -g my-RG --vnet-name my-VNet -n subnet1 \
    --address-prefixes 10.30.1.0/24

az network vnet subnet create -g my-RG --vnet-name my-VNet -n subnet2 \
    --address-prefixes 10.30.2.0/24

az network vnet subnet create -g my-RG --vnet-name my-VNet -n subnet3 \
    --address-prefixes 10.30.3.0/24

az network vnet subnet create -g my-RG --vnet-name my-VNet -n subnet4 \
    --address-prefixes 10.30.4.0/24
```

Example output

```
{
  "addressPrefix": "10.30.1.0/24",
  "addressPrefixes": null,
  "applicationGatewayIpConfigurations": null,
  "delegations": [],
  "etag": "W/\"32bb3a61-c446-4c20-b596-d92b6b9e2e9f\"",
  "id": "/subscriptions/5b9ae547-ce82-4834-b276-b72904ceaa84/resourceGroups/my-
RG/providers/Microsoft.Network/virtualNetworks/my-VNet/subnets/subnet1",
  "ipAllocations": null,
  "ipConfigurationProfiles": null,
  "ipConfigurations": null,
  "name": "subnet1",
  "natGateway": null,
  "networkSecurityGroup": null,
  "privateEndpointNetworkPolicies": "Disabled",
  "privateEndpoints": null,
  "privateLinkServiceNetworkPolicies": "Enabled",
  "provisioningState": "Succeeded",
  "purpose": null,
  "resourceGroup": "my-RG",
  "resourceNavigationLinks": null,
  "routeTable": null,
  "serviceAssociationLinks": null,
  "serviceEndpointPolicies": null,
  "serviceEndpoints": null,
  "type": "Microsoft.Network/virtualNetworks/subnets"
}
{
  "addressPrefix": "10.30.2.0/24",
  "addressPrefixes": null,
  "applicationGatewayIpConfigurations": null,
  "delegations": [],
  "etag": "W/\"8ec29708-e749-4a89-813e-0290c3c9a6f7\"",
  "id": "/subscriptions/5b9ae547-ce82-4834-b276-b72904ceaa84/resourceGroups/my-
RG/providers/Microsoft.Network/virtualNetworks/my-VNet/subnets/subnet2",
  "ipAllocations": null,
  "ipConfigurationProfiles": null,
  "ipConfigurations": null,
  "name": "subnet2",
  "natGateway": null,
  "networkSecurityGroup": null,
  "privateEndpointNetworkPolicies": "Disabled",
  "privateEndpoints": null,
  "privateLinkServiceNetworkPolicies": "Enabled",
  "provisioningState": "Succeeded",
  "purpose": null,
  "resourceGroup": "my-RG",
  "resourceNavigationLinks": null,
  "routeTable": null,
  "serviceAssociationLinks": null,
  "serviceEndpointPolicies": null,
  "serviceEndpoints": null,
  "type": "Microsoft.Network/virtualNetworks/subnets"
}
{
  "addressPrefix": "10.30.3.0/24",
  "addressPrefixes": null,
  "applicationGatewayIpConfigurations": null,
  "delegations": [],
  "etag": "W/\"4b9ba2be-e385-48e7-be24-c52c79769c3a\"",
  "id": "/subscriptions/5b9ae547-ce82-4834-b276-b72904ceaa84/resourceGroups/my-
RG/providers/Microsoft.Network/virtualNetworks/my-VNet/subnets/subnet3",
  "ipAllocations": null,
  "ipConfigurationProfiles": null,
  "ipConfigurations": null,
  "name": "subnet3",
  "natGateway": null,
  "networkSecurityGroup": null,
  "privateEndpointNetworkPolicies": "Disabled",
  "privateEndpoints": null,
  "privateLinkServiceNetworkPolicies": "Enabled",
  "provisioningState": "Succeeded",
  "purpose": null,
  "resourceGroup": "my-RG",
  "resourceNavigationLinks": null,
  "routeTable": null,
  "serviceAssociationLinks": null,
  "serviceEndpointPolicies": null,
  "serviceEndpoints": null,
  "type": "Microsoft.Network/virtualNetworks/subnets"
}
{
  "addressPrefix": "10.30.4.0/24",
  "addressPrefixes": null,
  "applicationGatewayIpConfigurations": null,
  "delegations": [],
  "etag": "W/\"ff08c2d1-2166-4c64-9892-3cac9bc20fd1\"",
  "id": "/subscriptions/5b9ae547-ce82-4834-b276-b72904ceaa84/resourceGroups/my-
RG/providers/Microsoft.Network/virtualNetworks/my-VNet/subnets/subnet4",
  "ipAllocations": null,
  "ipConfigurationProfiles": null,
  "ipConfigurations": null,
  "name": "subnet4",
  "natGateway": null,
  "networkSecurityGroup": null,
  "privateEndpointNetworkPolicies": "Disabled",
  "privateEndpoints": null,
  "privateLinkServiceNetworkPolicies": "Enabled",
  "provisioningState": "Succeeded",
  "purpose": null,
  "resourceGroup": "my-RG",
  "resourceNavigationLinks": null,
  "routeTable": null,
  "serviceAssociationLinks": null,
  "serviceEndpointPolicies": null,
  "serviceEndpoints": null,
  "type": "Microsoft.Network/virtualNetworks/subnets"
}
```

5. Now you are ready to create the AKS clusters. Get the subscription ID of `subnet1` by either running the following command (it uses the `jq` command) or copying it from the subnet creation output.

   ```
   az network vnet subnet show -g my-RG --vnet-name my-VNet -n subnet1 --output json | jq .id
   ```

   Example output

   ```
   "/subscriptions/5b9ae547-ce82-4834-b276-b72904ceaa84/resourceGroups/my-
   RG/providers/Microsoft.Network/virtualNetworks/my-VNet/subnets/
   ```

6. Create the first AKS cluster named `my-aks-cluster-a` with 5 nodes in your resource group using the subscription ID.

```
az aks create --name my-aks-cluster-a --node-count=5 --zones 1 --vnet-subnet-id
"/subscriptions/5b9ae547-ce82-4834-b276-b72904ceaa84/resourceGroups/my-
RG/providers/Microsoft.Network/virtualNetworks/my-VNet/subnets/subnet1" -g my-RG

az aks create --name my-aks-cluster-b --node-count=5 --zones 1 --vnet-subnet-id
"/subscriptions/5b9ae547-ce82-4834-b276-b72904ceaa84/resourceGroups/my-
RG/providers/Microsoft.Network/virtualNetworks/my-VNet/subnets/subnet2" -g my-RG

az aks create --name my-aks-cluster-c --node-count=5 --zones 1 --vnet-subnet-id
"/subscriptions/5b9ae547-ce82-4834-b276-b72904ceaa84/resourceGroups/my-
RG/providers/Microsoft.Network/virtualNetworks/my-VNet/subnets/subnet3" -g my-RG
```

Example output

```
Waiting for AAD role to propagate[################################    ]  90.0000%
{
  "aadProfile": null,
  "addonProfiles": null,
  "agentPoolProfiles": [
    {
      "availabilityZones": [
        "1"
      ],
      "count": 5,
      "creationData": null,
      "currentOrchestratorVersion": "1.23.8",
      "enableAutoScaling": false,
      "enableEncryptionAtHost": false,
      "enableFips": false,
      "enableNodePublicIp": false,
      "enableUltraSsd": false,
      "gpuInstanceProfile": null,
      "hostGroupId": null,
      "kubeletConfig": null,
      "kubeletDiskType": "OS",
      "linuxOsConfig": null,
      "maxCount": null,
      "maxPods": 110,
      "minCount": null,
      "mode": "System",
      "name": "nodepool1",
      "nodeImageVersion": "AKSUbuntu-1804gen2containerd-2022.08.23",
      "nodeLabels": null,
      "nodePublicIpPrefixId": null,
      "nodeTaints": null,
      "orchestratorVersion": "1.23.8",
      "osDiskSizeGb": 128,
      "osDiskType": "Managed",
      "osSku": "Ubuntu",
      "osType": "Linux",
      "podSubnetId": null,
      "powerState": {
        "code": "Running"
      },
      "provisioningState": "Succeeded",
      "proximityPlacementGroupId": null,
      "scaleDownMode": null,
      "scaleSetEvictionPolicy": null,
      "scaleSetPriority": null,
      "spotMaxPrice": null,
      "tags": null,
      "type": "VirtualMachineScaleSets",
      "upgradeSettings": {
        "maxSurge": null
      },
      "vmSize": "Standard_DS2_v2",
      "vnetSubnetId": "/subscriptions/5b9ae547-ce82-4834-b276-b72904ceaa84/resourceGroups/my-
RG/providers/Microsoft.Network/virtualNetworks/my-VNet/subnets/subnet1",
      "workloadRuntime": null
    }
  ],
  "apiServerAccessProfile": null,
  "autoScalerProfile": null,
  "autoUpgradeProfile": null,
  "azurePortalFqdn": "my-aks-my-rg-5b9ae5-bd2a82e4.portal.hcp.eastus.azmk8s.io",
  "currentKubernetesVersion": "1.23.8",
  "disableLocalAccounts": false,
  "diskEncryptionSetId": null,
  "dnsPrefix": "my-aks-my-RG-5b9ae5",
  "enablePodSecurityPolicy": null,
  "enableRbac": true,
  "extendedLocation": null,
  "fqdn": "my-aks-my-rg-5b9ae5-bd2a82e4.hcp.eastus.azmk8s.io",
  "fqdnSubdomain": null,
  "httpProxyConfig": null,
  "id": "/subscriptions/5b9ae547-ce82-4834-b276-b72904ceaa84/resourcegroups/my-
RG/providers/Microsoft.ContainerService/managedClusters/my-aks-cluster-a",
  "identity": {
    "principalId": "16334702-6bbd-44a0-8090-a7739b881974",
    "tenantId": "54e85725-ed2a-49a4-a19e-11c8d29f9a0f",
    "type": "SystemAssigned",
    "userAssignedIdentities": null
  },
  "identityProfile": {
    "kubeletidentity": {
      "clientId": "a445b12d-52d9-4564-b5cf-daa98bf17ab8",
      "objectId": "91cc2d37-0407-4916-a4cd-51849fbc6541",
      "resourceId": "/subscriptions/5b9ae547-ce82-4834-b276-b72904ceaa84/resourcegroups/MC_my-RG_my-
aks-cluster-a_eastus/providers/Microsoft.ManagedIdentity/userAssignedIdentities/my-aks-cluster-a-
agentpool"
    }
  },
  "kubernetesVersion": "1.23.8",
  "linuxProfile": {
    "adminUsername": "azureuser",
    "ssh": {
      "publicKeys": [
        {
          "keyData": "ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAACAQDzCbi+J6eJq9RTsCGFFhTk/PQrl8jNzbFYsPZeu4BKvyrRz7JfWRgzGLu6OTJynuUejKy6Xl
NaqYsEoZMsFdOMMYoK/bVCCUwOaOrpGAqNF9dCKbKkEnA6iv6WgEIfVHGoCtMc3pBRU0R9rfWYpf3h7WT/oShnaLzVhPUG+4Jblx3K
2tRsZ5+2AEgQeniXUgtZRvPes8qXfr/OES7M7owI0VuOVgiuJo3//sCvDavGJwSAgUECzcPYOEwBfmTWNleYrluiEWc7Ye5Y+W8j86
V4L/vh4LRs14WZ92Jt6K3QhshGSpY0tcLnxg7fskdZDtdcSWIPWpbQLdTxdIETKe66qDiijLXkpw2m3XRe8nTc5ysoXGaKvzASAWyR
2FYpYvmaSSGe/65jeQMsDjSsEXnLRoDG2A3aHy5yV44QXSd4N9/+Znmea1WnB+tvOUuAlhIgjWvprRPXyhZHdybuQipXPErfYg4G83
HWMwh35D5qBAV3DeZUIYYATFszYdGfp3ghdu1LBVXsgH/sHaMZXp9uy5PAP4jOxfGpho3k+UoQZHK3wwskxhK8/IiWpRPRPWUbfhUO
ilUdkQup8hyfVfGpW7htW3crFwXFbU1LG5gDNrars0i3OHqT1snFB3R38vxDaXdOZCEVPSQAevOj3Q/WYfO2m5o+gp2sEQtEp4mG+w
== my.popova@gmail.com\n"
        }
      ]
    }
  },
  "location": "eastus",
  "maxAgentPools": 100,
  "name": "my-aks-cluster-a",
  "networkProfile": {
    "dnsServiceIp": "10.0.0.10",
    "dockerBridgeCidr": "172.17.0.1/16",
    "ipFamilies": [
      "IPv4"
    ],
    "loadBalancerProfile": {
      "allocatedOutboundPorts": null,
      "effectiveOutboundIPs": [
        {
          "id": "/subscriptions/5b9ae547-ce82-4834-b276-b72904ceaa84/resourceGroups/MC_my-RG_my-aks-
cluster-a_eastus/providers/Microsoft.Network/publicIPAddresses/e7480132-3f34-4f2d-bbc3-4e27e23e574c",
          "resourceGroup": "MC_my-RG_my-aks-cluster-a_eastus"
        }
      ],
      "enableMultipleStandardLoadBalancers": null,
      "idleTimeoutInMinutes": null,
      "managedOutboundIPs": {
        "count": 1,
        "countIpv6": null
      },
      "outboundIPs": null,
      "outboundIpPrefixes": null
    },
    "loadBalancerSku": "Standard",
    "natGatewayProfile": null,
    "networkMode": null,
    "networkPlugin": "kubenet",
    "networkPolicy": null,
    "outboundType": "loadBalancer",
    "podCidr": "10.244.0.0/16",
    "podCidrs": [
      "10.244.0.0/16"
    ],
    "serviceCidr": "10.0.0.0/16",
    "serviceCidrs": [
      "10.0.0.0/16"
    ]
  },
  "nodeResourceGroup": "MC_my-RG_my-aks-cluster-a_eastus",
  "podIdentityProfile": null,
  "powerState": {
    "code": "Running"
  },
  "privateFqdn": null,
  "privateLinkResources": null,
  "provisioningState": "Succeeded",
  "publicNetworkAccess": null,
  "resourceGroup": "my-RG",
  "securityProfile": {
    "azureKeyVaultKms": null,
    "defender": null
  },
  "servicePrincipalProfile": {
    "clientId": "msi",
    "secret": null
  },
  "sku": {
    "name": "Basic",
    "tier": "Free"
  },
  "storageProfile": {
    "diskCsiDriver": {
      "enabled": true
    },
    "fileCsiDriver": {
      "enabled": true
    },
    "snapshotController": {
      "enabled": true
    }
  },
  "systemData": null,
  "tags": {
    "Owner Department": "Engineering - Neo4j"
  },
  "type": "Microsoft.ContainerService/ManagedClusters",
  "windowsProfile": null
}
```

7. Repeat the previous two steps to create two more AKS clusters, named `my-aks-cluster-b` and `my-aks-cluster-c`.

8. Configure `kubectl` to use your AKS clusters using:

```
az aks get-credentials --name my-aks-cluster-a --admin -g my-RG
az aks get-credentials --name my-aks-cluster-b --admin -g my-RG
az aks get-credentials --name my-aks-cluster-c --admin -g my-RG
```

Example output

```
Merged "my-aks-cluster-a-admin" as current context in /Users/myuser/.kube/config
Merged "my-aks-cluster-b-admin" as current context in /Users/myuser/.kube/config
Merged "my-aks-cluster-c-admin" as current context in /Users/myuser/.kube/config
```

In rare cases, where the usual K8S Kubernetes discovery methods do not work in your deployment/environment, you can use the `multiCluster` flag along with the `LIST` discovery method and perform all your network settings manually, as if you were using VMs for example. You need one load balancer per Neo4j Instance.

Install Neo4j on each AKS cluster

Install the *neo4j/neo4j* helm chart on each AKS cluster. The `LoadBalancer` service will be configured to use a private IP address from the subnet that was associated with the cluster. For example `my-aks-cluster-a` uses `subnet1` with the range `10.30.1.0/24`, so any address can be used from `10.30.1.1-10.30.1.255`. In the example, the following are used for illustration:

- **server-1**: 10.30.1.101
- **server-2**: 10.30.2.101
- **server-3**: 10.30.3.101

You must have owner's permissions in the resource group or a custom permission with the `Microsoft.Authorization/roleAssignments/write` role not to face auth issues while deploying the load balancers.

Create a *values.yaml* file for each cluster member

Create a custom YAML file for each Neo4j cluster member, for example, *server-1.values.yaml*, *server-2.values.yaml*, *server-3.values.yaml*.

The property `services.neo4j.spec.loadBalancerIP` must be set to an IP address from the `subnet1` example.

1. Switch to the context of the first AKS cluster `my-aks-cluster-a-admin` using:

   ```
   kubectl config use-context my-aks-cluster-a-admin
   ```

2. In the *server-1.values.yaml*, add the following settings to the Neo4j configuration:

server-1.values.yaml

```
# Neo4j Configuration (yaml format)
neo4j:
  name: multicluster
  minimumClusterSize: 3
  acceptLicenseAgreement: "yes"
  edition: enterprise
volumes:
  data:
    mode: defaultStorageClass
services:
  neo4j:
    annotations:
      service.beta.kubernetes.io/azure-load-balancer-internal: "true"
    spec:
      loadBalancerIP: 10.30.1.101
    multiCluster: true
config:
  dbms.cluster.discovery.type: LIST
  dbms.cluster.discovery.endpoints: "10.30.1.101:5000, 10.30.2.101:5000, 10.30.3.101:5000"
  server.discovery.advertised_address: "10.30.1.101:5000"
  server.cluster.advertised_address: "10.30.1.101:6000"
  server.cluster.raft.advertised_address: "10.30.1.101:7000"
  server.bolt.advertised_address: "10.30.1.101:7687"
  server.routing.advertised_address: "10.30.1.101:7688"
```

3. In the *server-2.values.yaml*, add the following settings to the Neo4j configuration:

server-2.values.yaml

```
# Neo4j Configuration (yaml format)
neo4j:
  name: multicluster
  minimumClusterSize: 3
  acceptLicenseAgreement: "yes"
  edition: enterprise
volumes:
  data:
    mode: defaultStorageClass
services:
  neo4j:
    spec:
      loadBalancerIP: 10.30.2.101
    multiCluster: true
config:
  dbms.cluster.discovery.type: LIST
  dbms.cluster.discovery.endpoints: "10.30.1.101:5000, 10.30.2.101:5000, 10.30.3.101:5000"
  server.discovery.advertised_address: "10.30.2.101:5000"
  server.cluster.advertised_address: "10.30.2.101:6000"
  server.cluster.raft.advertised_address: "10.30.2.101:7000"
  server.bolt.advertised_address: "10.30.2.101:7687"
  server.routing.advertised_address: "10.30.2.101:7688"
```

4. In the *server-3.values.yaml*, add the following settings to the Neo4j configuration:

server-3.values.yaml

```
# Neo4j Configuration (yaml format)
neo4j:
  name: multicluster
  minimumClusterSize: 3
  acceptLicenseAgreement: "yes"
  edition: enterprise
volumes:
  data:
    mode: defaultStorageClass
services:
  neo4j:
    spec:
      loadBalancerIP: 10.30.3.101
    multiCluster: true
config:
  dbms.cluster.discovery.type: LIST
  dbms.cluster.discovery.endpoints: "10.30.1.101:5000, 10.30.2.101:5000, 10.30.3.101:5000"
  server.discovery.advertised_address: "10.30.3.101:5000"
  server.cluster.advertised_address: "10.30.3.101:6000"
  server.cluster.raft.advertised_address: "10.30.3.101:7000"
  server.bolt.advertised_address: "10.30.3.101:7687"
  server.routing.advertised_address: "10.30.3.101:7688"
```

Deploy the Neo4j cluster

1. Switch the context to `my-aks-cluster-a-admin` using:

```
kubectl config use-context my-aks-cluster-a-admin
```

2. Install `server-1` using the *server-1.values.yaml*:

```
helm install server-1 neo4j/neo4j -f /path/to/server-1.values.yaml
```

Example output

```
NAME: server-1
LAST DEPLOYED: Tue Nov  1 14:53:54 2022
NAMESPACE: neo4j
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Thank you for installing neo4j.

Your release "server-1" has been installed  in namespace "neo4j".

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

This release creates a single member of a Neo4j cluster. It will not become ready until it is able to
form a working Neo4j cluster by joining other Neo4j servers. To create a working cluster at least 3
servers are required.

Once you have a working Neo4j cluster, you can access the Neo4j browser using the IP address of the
multicluster-lb-neo4j service
eg. http://[SERVICE_IP]:7474

Graphs are everywhere!
```

3. Switch the context to `my-aks-cluster-b-admin`.

```
kubectl config use-context my-aks-cluster-b-admin
```

4. Install `server-2` using the *server-2.values.yaml*:

```
helm install server-2 neo4j/neo4j -f /path/to/server-2.values.yaml
```

5. Switch the context to `my-aks-cluster-c-admin`.

```
kubectl config use-context my-aks-cluster-c-admin
```

6. Install `server-3` using the *server-3.values.yaml*:

```
helm install server-3 neo4j/neo4j -f /path/to/server-3.values.yaml
```

7. Switch to each context and check that the pod there is `READY` and that they have formed a cluster.

```
kubectl get pods
```

```
NAME        READY   STATUS    RESTARTS   AGE
server-1-0   1/1     Running   0           4m51s
```

Create an Azure application gateway

You create an application gateway to access the Neo4j cluster from outside the AKS clusters.

1. Log in to the Azure portal.
2. In the **Search resources** area, look for Application Gateway.
   The Load balancing | Application Gateway window opens.
3. Click the **Create** button.
4. On the **Basics** tab, configure the following settings:
   a. From the **Resource group** dropdown, select `my-RG`.
   b. Add a name for your application gateway in the **Application gateway/name** field.
   c. From the **Region** dropdown, select your region. In this example, it is East US.
   d. Disable the autoscaling.
   e. From the **Virtual network** dropdown, select `my-VNet`.
   f. From the **Subnet** dropdown, select `subnet4 | 10.30.4.0/24`.
   g. Click **Next : Frontends >**.
5. On the **Frontends** tab, configure the Public IP address:
   a. From the **Frontend IP address type**, select **Public**.
   b. In **Public IP address**, click **Add new**.

            i. Add a name for it, for example, `apg-public`.

            ii. Click **OK**.

        c. Click **Next : Backends >**.

6. On the **Backends** tab, add the EXTERNAL-IPs of the load balancers:

    a. Click **Add a backend pool**.

        i. Add a name for the pool, for example, `neo4j-cluster`.

        ii. In the **Target type** list, add the EXTERNAL-IPs of the load balancers under **Target**.

    b. Click **Add**.

    c. Click **Next : Configuration >**.

7. On the **Configuration** tab, configure the routing rules:

    a. Click **Add a routing rule** and configure a routing rule for port `7474`.

        i. In the **Rule name** field, add a name for your rule, for example, `rule7474`.

        ii. For **Priority**, add `1`.

        iii. On the **Listener** tab, add a name for the **Listener** (e.g., `listener7474`), select **Frontend IP** to be **Public**, and add port `7474`.

        iv. On the **Backend targets** tab, configure the backend target and settings:

            A. From the **Target type** dropdown, select the **Backend pool** to be `neo4j-cluster`.

            B. For **Backend targets**, click **Add new** and config.

                I. In the **Backend settings name**, type `settings7474`.

                II. In the **Backend port**, type `7474`.

        v. Click **Add**.

    b. Click **Add a routing rule** and configure a routing rule for port `7687`.

        i. In the **Rule name** field, add a name for your rule, for example, `rule7687`.

        ii. For **Priority**, add `2`.

        iii. On the **Listener** tab, add a name for the **Listener** (e.g., `listener7687`), select **Frontend IP** to be **Public**, and add port `7687`.

        iv. On the **Backend targets** tab, configure the backend target and settings:

            A. From the **Target type** dropdown, select the **Backend pool** to be `neo4j-cluster`.

            B. For **Backend targets**, click **Add new** and config.

                I. In the **Backend settings name**, type `settings7687`.

                II. In the **Backend port**, type `7687`.

        v. Click **Add**.

    c. Click **Next : Tags >**.

    d. Click **Next : Review + create >**, review your configurations, and click **Create**.

Access the Neo4j cluster

After the application gateway is created and the deployment is complete, you can access the Neo4j cluster via the Neo4j Browser.

1. Copy the **Frontend public IP address** and paste it into a browser.
2. Add port `:7474`.
   Neo4j Browser opens.
3. Log in with your credentials (e.g., `neo4j`/`my-password`).
4. Verify that the cluster is up and running using the Cypher command `SHOW DATABASES`.

Troubleshooting

The following information can help you diagnose and correct a problem.

Locate and investigate problems with the Neo4j Helm chart

The rollout of the Neo4j Helm chart in Kubernetes can be thought of in these approximate steps:

1. *Neo4j Pod* is created.
2. Neo4j Pod is scheduled to run on a specific Kubernetes Node.
3. All Containers in the Neo4j Pod are created.
4. *InitContainers* in the Neo4j Pod is run.
5. *Containers* in the Neo4j Pod are run.
6. `Startup` and `Readiness` probes are checked.

After all these steps are completed successfully, the Neo4j StatefulSet, Pod, and Services must be in a `ready` state. You should be able to connect to and use your Neo4j database.

If the Neo4j Helm chart is installed successfully, but Neo4j is not starting and reaching a `ready` state in Kubernetes, then troubleshooting has two steps:

1. Check the state of resources in Kubernetes using `kubectl get` commands. This will identify which step has failed.
2. Collect the information relevant to that step.

Depending on the failed step, you can collect information from Kubernetes (e.g., using `kubectl describe`) and from the Neo4j process (e.g., checking the Neo4j debug log).

The following table provides simple steps to get started investigating problems with the Neo4j Helm chart rollout. For more information on how to debug applications in Kubernetes, see the Kubernetes documentation.

Table 15. Investigating problems with the Neo4j Helm chart rollout

Step	Diagnosis	Further investigation

Neo4j Pod created	If `kubectl get pod <release-name>-0` does not return a single Pod result, there is a problem with the pod creation.	Describe the Neo4j StatefulSet — check the output of `kubectl describe statefulset <release-name>`.
Neo4j Pod scheduled	If the state, shown in `kubectl get pod <release-name>-0`, is stuck in `Pending`, there is a problem with pod scheduling.	Describe the Neo4j Pod `kubectl describe pod <release-name>-0` and check the output.
Containers in the Neo4j Pod created	If the state, shown in `kubectl get pod <release-name>-0`, is stuck in `Waiting`, there is a problem with creating or starting containers.	Describe the Neo4j Pod — check the output of `kubectl describe pod <release-name>-0`, paying particular attention to `Events`.
InitContainers in the Neo4j Pod	If the state, shown in `kubectl get pod <release-name>-0`, is stuck in `Init:` (e.g.,`Init:CrashLoopBackOff`, `Init:Error` etc.), there is a problem with `InitContainers`. Note that if the pod `Status` is `PodInitializing` or `Running`, then `InitContainers` have already finished successfully.	Describe the Neo4j Pod — check the output of `kubectl describe pod <release-name>-0`, paying particular attention to `InitContainer` (note the `InitContainer` names) and `Events`. Fetch `InitContainer` logs using `kubectl logs <pod-name> -c <init-container-name>`.
Containers in the Neo4j Pod running	If the state, shown in `kubectl get pod <release-name>-0`, does NOT match any of the states listed above, but the Pod still does not reach `Running`, then there is a problem running containers in the Neo4j Pod.	Describe the Neo4j Pod — check the output of `kubectl describe pod <release-name>-0`, paying particular attention to the `Container` state (note the `Container` names) and `Events`. Fetch `Container` logs using `kubectl logs <pod-name> -c <init-container-name>`. If the Neo4j `Container` is starting but exits unexpectedly (e.g., the state is `CrashLoopBackOff`), follow the instructions for Neo4j crashes or restarts unexpectedly.
Startup and Readiness Probes	If the state, shown in `kubectl get pod <release-name>-0`, is `Running`, but the pod does not become `ready`, there is a problem with `Startup` or `Readiness` probes.	Describe the Neo4j Pod — check the output of `kubectl describe pod <release-name>-0`, paying particular attention to `Events` and probes. Check the pod log `kubectl logs <release-name>-0`, the Neo4j log `kubectl exec <release-name>-0 -- tail -n 100 /logs/neo4j.log`, and the Neo4j debug log `kubectl exec <release-name>-0 -- tail -n 500 /logs/debug.log`.

Neo4j crashes or restarts unexpectedly

If the Neo4j Pod starts but then crashes or restarts unexpectedly, there are a range of possible causes. Known causes include:

- An invalid or incorrect configuration of Neo4j, causing it to shut down shortly after the container is started.
- The Neo4j Java process runs out of memory and exits with `OutOfMemoryException`.

- There has been some disruption affecting the Kubernetes Node where the Neo4j Pod is scheduled, e.g., it is being shut drained or has shut down.
- Containers in the Neo4j Pod are shut down by the operating system for using more memory than the resource limit configured for the container (`OOMKilled`).
- Very long Garbage Collection pauses cause the Neo4j Pod `LivenessProbe` to fail, causing Kubernetes to restart Neo4j.

`OOMKILLED` and `OutOfMemoryException` appear very similar, but they appear in different places and have different fixes. It is important to be aware of this and be sure of what you are dealing with.

Here are some checks to help troubleshoot crashes and unexpected restarts:

Describe the Neo4j Pod

Use `kubectl` to describe the Neo4j Pod:

```
kubectl describe pod <release-name>-0
```

Check the Neo4j Container state

Check the `State` and `Last State` of the container. This shows how the `Last State` of a container that has restarted after being `OOMKilled` appears:

```
$ kubectl describe pod neo4j-0
```

```
State:          Running
  Started:      Mon, 1 Jan 2021 00:02:00 +0000
Last State:     Terminated
  Reason:       OOMKilled
  Exit Code:    137
  Started:      Mon, 1 Jan 2021 00:00:00 +0000
  Finished:     Mon, 1 Jan 2021 00:01:00 +0000
```

`Exit Code: 137` is indicative of `OOMKilled` if it appears here or in other logs, even if the `"OOMKilled"` string is not present.

Check recent `Events`

The `kubectl describe` output shows older events at the top and more recent events at the bottom. Generally, you can ignore older events.

A `Killing` *event that shows that the Neo4j container was killed by the Kubernetes* `kubelet`:

```
$ kubectl describe pod neo4j-0
```

```
Events:
Type    Reason       Age     From                  Message
----    ------       ----    ----                  -------
Normal  Scheduled    6m30s   default-scheduler     Successfully assigned default/neo4j-0 to k8s-node-a
...
Normal  Killing        56s   kubelet, k8s-node-a   Killing container with id docker://neo4j-0-neo4j:Need
to kill Pod
```

It is not clear from this event log alone why Kubernetes decided that the Neo4j container should be killed.

The next steps in this example could be to check:

- if the container was `OOMKilled`.
- if the container failed `Liveness` or `Startup` probes.
- investigate the node to see if there was some reason why it might kill the container, e.g.,`kubectl describe node <k8s node>`.

Check Neo4j logs and metrics

The Neo4j Helm chart configures Neo4j to persist logs and metrics on provided volumes. If no volume is explicitly configured for logs or metrics, they are stored persistently on the Neo4j data volume. This ensures that the logs and metrics outputs from a Neo4j instance that crashes or shuts down unexpectedly are preserved.

Collect data from a running Neo4j Pod

- Download all Neo4j logs from a pod using `kubectl cp` commands:

  ```
  kubectl cp <neo4j-pod-name>:/logs neo4j-logs/
  ```

- If CSV metrics collection is enabled for Neo4j (the default), download all Neo4j metrics from a pod using:

  ```
  kubectl cp <neo4j-pod-name>:/metrics neo4j-metrics/
  ```

Collect data from a not running Neo4j Pod

If the Neo4j Pod is not running or is crashing so frequently that `kubectl cp` is not feasible, the Neo4j deployment should be put into offline maintenance mode to collect logs and metrics.

Check container logs

The logs for the main Neo4j DBMS process are persisted to disk and can be accessed as described in Check Neo4j logs and metrics. However, the logs for Neo4j startup and logs for other Containers in the Neo4j Pod are sent to the container's `stdout` and `stderr` streams. These container logs can be viewed using `kubectl logs <pod name> -c <container name>`.

Unfortunately, if the container has restarted following a crash or unexpected shutdown, typically, `kubectl logs` shows the logs for the new container instance (following the restart), and the logs for the previous container instance (the instance that shut down unexpectedly) are not available via `kubectl logs`.

To capture the logs for a crashing container, you can try:

- View the container logs in a log collector/aggregator that is connected to your Kubernetes cluster, e.g., Stackdriver, Cloudwatch Logs, Logstash, etc. If you are using a managed Kubernetes platform, this is usually enabled by default.
- Use `kubectl logs --follow` to stream the logs of a running container until it crashes again.

[3] Not recommended because of inconsistencies in Docker Desktop handling of `hostPath` volumes.

Configuration

The topics described are:

- The *neo4j.conf* file — An introduction to the primary configuration file in Neo4j.
- Default file locations — An overview of where files are stored in the different Neo4j distributions and the necessary file permissions for running Neo4j.
- Ports — An overview of the ports relevant to a Neo4j installation.
- Configure Neo4j connectors — How to configure Neo4j connectors.
- Set initial password — How to set an initial password.
- Password and user recovery — How to recover after a lost admin password.
- Update 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 All configuration settings.

The neo4j.conf file

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

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

`neo4j.conf` conventions

The syntax in the `neo4j.conf` file follows the following conventions:

- The equals sign (`=`) maps configuration setting keys to configuration values.
- Lines that start with the number sign (`#`) are handled as comments.
- Trailing comments are not supported.
- Empty lines are ignored.
- Configuring a setting in *neo4j.conf* overwrites any default values. If you want to amend the default values with custom ones, you must explicitly list the default values along with the new ones.
- The configuration settings are not ordered.
- The configuration settings have strict validation enabled by default. It prevents Neo4j from starting if the *neo4j.conf* file contains *typos*, *incorrect information*, or *duplicates* (except for `server.jvm.additional`). If you set more than one value for `server.jvm.additional`, each setting value adds another custom JVM argument to the `java` launcher.

To disable the strict validation, set `server.config.strict_validation.enabled=false`.

Configuration settings

General synopsis

Neo4j configuration settings have the following general synopsis:

`<prefix>.<scope>.<component>….<component>.<name>`

Prefix

Prefixes are reserved for denoting two special cases (most settings do not have a prefix):

- `initial` — Settings that are only used during the initialization but are ignored thereafter. For example, `initial.server.mode_constraint`, `initial.dbms.default_database`, etc.
- `internal` — The prefix replaces the terms `unsupported` and `experimental` used in previous versions. This namespace is dedicated to features that are used internally and may change without notice.

Scope

All configuration settings fall into one of the following scopes that behave differently:

- `db` settings can be varied between each database but must be consistent across all configuration files in a cluster/DBMS.
- `dbms` settings must be consistent across all configuration files in a cluster/DBMS.
- `server` settings apply only to the specific server and can be varied between configuration files across a cluster/DBMS.
- `browser` settings apply only to Neo4j Browser.
- `client` settings apply only to the client.

In Neo4j 5, the `fabric` scope is no longer available. All configuration settings identified by the `fabric` namespace in the `neo4j.conf` file are moved into the `system` database. The Cypher surface is extended to support the Fabric configuration.

For more information, see Composite databases.

Component

Component namespaces are used to group settings that affect similar systems.

Name

The name of the setting. It may contain a common verb and unit patterns, such as `size`, `enabled`, etc. Words are separated by an underscore.

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

JVM-specific configuration settings

A Java virtual machine (JVM) is a virtual machine that enables a computer to run Java programs and programs written in other languages that are also compiled 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 16. JVM-specific settings

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

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.

List currently active settings

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

Example 17. 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       |
+----------------------------------------------------+
| "server.default_advertised_address" | "localhost" |
| "initial.dbms.default_database"     | "neo4j"     |
| "server.default_listen_address"     | "localhost" |
+----------------------------------------------------+
```

For information about dynamic settings, see Update dynamic settings and Dynamic configuration settings reference.

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

Command expansion

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

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

How it works

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

A generic example would be:

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

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

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.

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 only be writeable (but not executable) by its owner.
- The neo4j.conf file must only be readable by its group and owner.
- The Neo4j process must run as a user who is either the owner of the *neo4j.conf* file or in the group of 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 only be writeable (but not executable) by the user that the Neo4j process runs as.
- The *neo4j.conf* file must only be readable by the user that the Neo4j process runs as.

Logging

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

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

Error Handling

The scripts' execution may generate two types of errors:

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

Default file locations

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

Table 17. 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 active Neo4j DBMS, select *Terminal*, and run `cd bin`.
Certificates	The Neo4j TLS certificates.	*<neo4j-home>/certificates*	*<neo4j-home>\certificates*	*/var/lib/neo4j/certificates*	From the Open dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd certificates`.
Configuration [5]	The Neo4j configuration settings, Log4j configuration settings, and the JMX access credentials.	*<neo4j-home>/conf/neo4j.conf* *<neo4j-home>/conf/server-logs.xml* *<neo4j-home>/conf/user-log.xml*	*<neo4j-home>\conf\neo4j.conf* *<neo4j-home>\conf\server-logs.xml* *<neo4j-home>\conf\user-log.xml*	*/etc/neo4j/neo4j.conf* */etc/neo4j/server-logs.xml* */etc/neo4j/user-log.xml*	From the Open dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd conf`.
Data [6]	All data-related content, such as databases, transactions, cluster-state (if applicable), dumps, and the *cypher.script* files (from the `neo4j-admin database restore` command).	*<neo4j-home>/data*	*<neo4j-home>\data*	*/var/lib/neo4j/data*	From the Open dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd data`.
Import	All CSV files that the command `LOAD CSV` uses as sources to import data in Neo4j.	*<neo4j-home>/import*	*<neo4j-home>\import*	*/var/lib/neo4j/import*	From the Open dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd import`.
Labs [7]	Contains APOC Core.	*<neo4j-home>/labs*	*<neo4j-home>\labs*	*/var/lib/neo4j/labs*	From the Open dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd labs`.

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 active Neo4j DBMS, select *Terminal*, and run `cd lib`.
Licenses	For storing license files from Neo4j.	*<neo4j-home>/licenses*	*<neo4j-home>\licenses*	*/var/lib/neo4j/licenses*	From the Open dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd licences`.
Logs	The Neo4j log files.	*<neo4j-home>/logs* [8]	*<neo4j-home>\logs*	*/var/log/neo4j/* [9]	From the Open dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd logs`.
Metrics	The Neo4j built-in metrics for monitoring the Neo4j DBMS and each individual database.	*<neo4j-home>/metrics*	*<neo4j-home>\metrics*	*/var/lib/neo4j/metrics*	From the Open dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd metrics`.
Plugins	Custom code that extends Neo4j, for example, user-defined procedures, functions, and security plugins.	*<neo4j-home>/plugins*	*<neo4j-home>\plugins*	*/var/lib/neo4j/plugins*	From the Open dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd plugins`.
Products	The jar files of the Neo4j products: Graph Data Science Library and Bloom. The folder also contains a *README.txt* file with information on enabling them.	*<neo4j-home>/products*	*<neo4j-home>\products*	*/var/lib/neo4j/products*	From the Open dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd products`.
Run	The processes IDs.	*<neo4j-home>/run*	*<neo4j-home>\run*	*/var/lib/neo4j/run*	From the Open dropdown menu of your active Neo4j DBMS, select *Terminal*, and run `cd run`.

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 18. Configuration of <neo4j-home> and conf

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

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

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

File permissions

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

Read only

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

Read and write

- *data*
- *logs*

- *metrics*
- *run*

Execute

- all files in *bin* and *tmp*

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

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

Ports

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

Specific recommendations on port openings cannot be made, as the firewall configuration must be performed according to your particular conditions.

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

Listen address configuration settings

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

Table 19. Listen address configuration settings overview

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

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

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

Advertised address configuration settings

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

Table 20. Advertised address configuration settings overview

Name	Default port	Related configuration setting
HTTP	`7474`	`server.http.advertised_address`
HTTPS	`7473`	`server.https.advertised_address`
Bolt	`7687`	`server.bolt.advertised_address`
Cluster discovery management	`5000`	`server.discovery.advertised_address`
Cluster transaction	`6000`	`server.cluster.advertised_address`
Cluster RAFT	`7000`	`server.cluster.raft.advertised_address`
Cluster routing connector	`7688`	`server.routing.advertised_address`

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

Ports used by Neo4j

Backup Enterprise edition

Default port: `6362`

Table 21. Backup

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

Related configuration setting	Default value	Description
`server.backup.enabled`	`true`	Enable support for running online backups.

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

For more information, see Server configuration.

HTTP

Default port: `7474`

Table 22. HTTP connector

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

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

For more information, see Configure connectors.

HTTPS

Default port: `7473`

Table 23. HTTPS connector

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

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

For more information, see Configure connectors.

Bolt

Default port: `7687`

Table 24. Bolt connector

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

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

For more information, see Configure connectors.

Cluster Enterprise edition

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

Table 25. Cluster listen address

Name	Default port	Default value	Related configuration setting
Discovery management	`5000`	`:5000`	`server.discovery.listen_address`
Transaction	`6000`	`:6000`	`server.cluster.listen_address`
RAFT	`7000`	`:7000`	`server.cluster.raft.listen_address`
Routing connector	`7688`	`:7688`	`server.routing.listen_address`

Table 26. Cluster advertised address

Name	Default port	Default value	Related configuration setting
Discovery management	`5000`	`:5000`	`server.discovery.advertised_address`
Transaction	`6000`	`:6000`	`server.cluster.advertised_address`
RAFT	`7000`	`:7000`	`server.cluster.raft.advertised_address`
Routing connector	`7688`	`:7688`	`server.routing.advertised_address`

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

For more information, see:

- Deploy a basic cluster
- Settings reference

Graphite monitoring

Default port: `2003`

Table 27. Graphite

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

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

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

Prometheus monitoring

Default port: `2004`

Table 28. Prometheus

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

For more information, see Prometheus.

JMX monitoring

Default port: `3637`

Table 29. Java Management Extensions

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

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

Remote debugging

Default port: `5005`

Table 30. Remote debugging

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

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

Configure connectors

Neo4j provides support for Bolt, HTTP, and HTTPS protocols using connectors. Connectors are configured in the *neo4j.conf* file.

Available connectors

The table below lists the available Neo4j connectors:

Table 31. Neo4j connectors and port number

Connector name	Protocol	Default port number
`server.bolt`	Bolt	`7687`
`server.http`	HTTP	`7474`
`server.https`	HTTPS	`7473`

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

Configuration options

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

Table 32. Configuration option suffixes for connectors

Option name	Default	Setting(s)	Description
`enabled`	`true` [10]	`server.bolt.enabled`, `server.http.enabled`, `server.https.enabled` [11]	This setting allows the client connector to be enabled or disabled. When disabled, Neo4j does not listen for incoming connections on the relevant port.

Option name	Default	Setting(s)	Description
listen_address	127.0.0.1:<connector-default-port>	server.bolt.listen_address, server.https.listen_address, server.http.listen_address	This setting specifies how Neo4j listens for incoming connections. It consists of two parts; an IP address (e.g. 127.0.0.1 or 0.0.0.0) and a port number (e.g. 7687), and is expressed in the format <ip-address>:<port-number>. See below for an example of usage.
advertised_address	localhost:<connector-default-port>	server.bolt.advertised_address, server.https.advertised_address, server.http.advertised_address	This setting specifies the address that clients should use for this connector. This is useful in a cluster as it allows each server to correctly advertise addresses of the other servers in the cluster. The advertised address consists of two parts; an address (fully qualified domain name, hostname, or IP address) and a port number (e.g. 7687), and is expressed in the format <address>:<port-number>. See below for an example of usage.
tls_level	DISABLED	server.bolt.tls_level	This setting is only applicable to the Bolt connector. It allows the connector to accept encrypted and/or unencrypted connections. The default value is DISABLED, where only unencrypted client connections are to be accepted by this connector, and all encrypted connections will be rejected. Other values are REQUIRED and OPTIONAL. Use REQUIRED when only encrypted client connections are to be accepted by this connector, and all unencrypted connections will be rejected. Use OPTIONAL where either encrypted or unencrypted client connections are accepted by this connector.

Example 18. Specify listen_address for the Bolt connector

To listen for Bolt connections on all network interfaces (0.0.0.0) and on port 7000, set the listen_address for the Bolt connector:

```
server.bolt.listen_address=0.0.0.0:7000
```

Example 19. Specify `advertised_address` for the Bolt connector

If routing traffic via a proxy, or if port mappings are in use, it is possible to specify `advertised_address` for each connector individually. For example, if port 7687 on the Neo4j Server is mapped from port 9000 on the external network, specify the `advertised_address` for the Bolt connector:

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

Options for Bolt thread pooling

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

Defaults for addresses

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

`server.default_listen_address`

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

Example 20. Specify `listen_address` for the Bolt connector

To listen for Bolt connections on all network interfaces (0.0.0.0) and on port 7000, set the `listen_address` for the Bolt connector:

```
server.bolt.listen_address=0.0.0.0:7000
```

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

```
server.default_listen_address=0.0.0.0

server.bolt.listen_address=:7000
```

`server.default_advertised_address`

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

Example 21. Specify `advertised_address` for the Bolt connector

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

```
server.bolt.advertised_address=server1:9000
```

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

```
server.default_advertised_address=server1

server.bolt.advertised_address=:9000
```

The default address settings can only accept the hostname or IP address portion of the full socket address. Port numbers are protocol-specific, and can only be added by the protocol-specific connector configuration.

For example, if you configure the default address value to be `example.com:9999`, Neo4j will fail to start and you will get an error in neo4j.log.

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.

Neo4j 5.3

The default minimum password length is 8 characters. Use the `dbms.security.auth_minimum_password_length` configuration to change it. The requirement has been introduced in **Neo4j 5.3**.

Syntax:

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

Example 22. 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 dbms set-initial-password h6u4%kr
```

Example 23. Use the `set-initial-password` command of neo4j-admin with the optional `--require-password-change` flag

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

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

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

Password and user recovery

This page describes how to reset a password to recover a user’s access when their password is lost. It specifically focuses on how to recover an admin user if all the admin users have been unassigned the admin role, and how to recreate the built-in admin role if it has been dropped.

Disable authentication

1. Stop Neo4j:

```
$ bin/neo4j stop
```

2. Open the *neo4j.conf* file and set `dbms.security.auth_enabled` parameter to `false` to disable the authentication:

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

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 `server.default_listen_address` parameter:

```
#server.default_listen_address=<your_configuration>
```

or

   - Providing the specific localhost value:

```
server.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.

   ```
   #server.http.enabled=true
   #server.https.enabled=true
   server.bolt.listen_address:127.0.0.1
   ```

3. Start all Core servers:

```
$ bin/neo4j start
```

Recover a lost password

You can use a client such as Cypher Shell or the Neo4j Browser to connect to the `system` database and set a new password for the admin user.

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.

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.

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.

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
   ```

Update dynamic settings

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.

Discover dynamic settings

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

Example 24. Discover dynamic settings

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

```
+------------------------------------------------------------------+
| name                                                             |
+------------------------------------------------------------------+
| "db.checkpoint.iops.limit"                                       |
| "db.format"                                                      |
| "db.lock.acquisition.timeout"                                    |
| "db.logs.query.early_raw_logging_enabled"                        |
| "db.logs.query.enabled"                                          |
| "db.logs.query.max_parameter_length"                             |
| "db.logs.query.obfuscate_literals"                               |
| "db.logs.query.parameter_logging_enabled"                        |
| "db.logs.query.plan_description_enabled"                         |
| "db.logs.query.threshold"                                        |
| "db.logs.query.transaction.enabled"                              |
| "db.logs.query.transaction.threshold"                            |
| "db.memory.transaction.max"                                      |
| "db.memory.transaction.total.max"                                |
| "db.track_query_cpu_time"                                        |
| "db.transaction.bookmark_ready_timeout"                          |
| "db.transaction.concurrent.maximum"                              |
| "db.transaction.sampling.percentage"                             |
| "db.transaction.timeout"                                         |
| "db.transaction.tracing.level"                                   |
| "db.tx_log.preallocate"                                          |
| "db.tx_log.rotation.retention_policy"                            |
| "db.tx_log.rotation.size"                                        |
| "dbms.cypher.render_plan_description"                            |
| "dbms.memory.transaction.total.max"                              |
| "dbms.routing.client_side.enforce_for_domains"                   |
| "dbms.routing.reads_on_writers_enabled"                          |
| "dbms.security.key.name"                                         |
| "dbms.security.keystore.password"                                |
| "dbms.security.keystore.path"                                    |
| "dbms.security.ldap.authentication.attribute"                    |
| "dbms.security.ldap.authentication.user_dn_template"             |
| "dbms.security.ldap.authorization.access_permitted_group"        |
| "dbms.security.ldap.authorization.group_membership_attributes"   |
| "dbms.security.ldap.authorization.group_to_role_mapping"         |
| "dbms.security.ldap.authorization.nested_groups_enabled"         |
| "dbms.security.ldap.authorization.nested_groups_search_filter"   |
| "dbms.security.ldap.authorization.user_search_base"              |
| "dbms.security.ldap.authorization.user_search_filter"            |
| "server.cluster.catchup.connect_randomly_to_server_group"         |
| "server.databases.default_to_read_only"                          |
| "server.databases.read_only"                                     |
| "server.databases.writable"                                      |
| "server.groups"                                                  |
| "server.memory.pagecache.flush.buffer.enabled"                   |
| "server.memory.pagecache.flush.buffer.size_in_pages"             |
+------------------------------------------------------------------+

46 rows
```

Update dynamic settings

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

Syntax:

`CALL dbms.setConfigValue(setting, value)`

Returns:

Nothing on success.

Exceptions:

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

The following example shows how to dynamically enable query logging.

Example 25. Set a config value

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

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

Example 26. Try to set invalid config value

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

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

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

Example 27. Reset a config value to default

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

Transaction log

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

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

ℹ The transaction log has nothing to do with log monitoring.

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 differential 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
`server.directories.transaction.logs.root`	`transactions`	Root location where Neo4j will store transaction logs for configured databases.
`db.tx_log.preallocate`	`true`	Specify if Neo4j should try to preallocate logical log file in advance.
`db.tx_log.rotation.retention_policy`	`2 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.
`db.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.

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 `server.directories.transaction.logs.root`. For maximum performance, it is recommended to configure transaction logs to be stored on a dedicated device.

Log rotation

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

Log retention

Manually deleting transaction log files is not supported.

You can control the number of transaction logs that Neo4j keeps using the parameter `db.tx_log.rotation.retention_policy`. It is set to `2 days` by default, which means Neo4j keeps logical logs that contain any transaction committed within 2 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 differential 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 33. Types that can be used to control log retention

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

Example 28. Configure log retention policy

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

- Keep transaction logs indefinitely:

  ```
  db.tx_log.rotation.retention_policy=true
  ```

  or

  ```
  db.tx_log.rotation.retention_policy=keep_all
  ```

- Keep only the most recent non-empty log:

  ```
  db.tx_log.rotation.retention_policy=false
  ```

  or

  ```
  db.tx_log.rotation.retention_policy=keep_none
  ```

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

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

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

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

Log pruning

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

Two things are necessary for a file to be removed:

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

Observing that you have more transaction log files than you expected is likely due to checkpoints either not happening frequently enough, or taking too long. This is a temporary condition and the gap between the expected and the observed number of log files will be closed on the next successful checkpoint. The interval between checkpoints can be configured using:

Checkpoint configuration	Default value	Description
`db.checkpoint.interval.time`	`15m`	Configures the time interval between checkpoints.
`db.checkpoint.interval.tx`	`100000`	Configures the transaction interval between checkpoints.

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 `db.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`.
[10] When Neo4j is used in embedded mode, the default value is `false`.
[11] The default value for `server.https.enabled` is `false`.

Manage databases

This chapter describes how to create and manage multiple active databases and contains the following:

Introduction

Concepts

With Neo4j 5 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 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.

Graph

A data model within a database.

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 Neo4j 5 each standard database contains a single graph. Many administrative commands refer to a specific graph by using the database name.

A database defines a *transaction domain* 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 5 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.

Composite database

A composite database is a logical grouping of multiple graphs contained in other, standard databases.

A composite database defines an *execution context* and a (limited) *transaction domain*.

For more information, see composite databases.

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

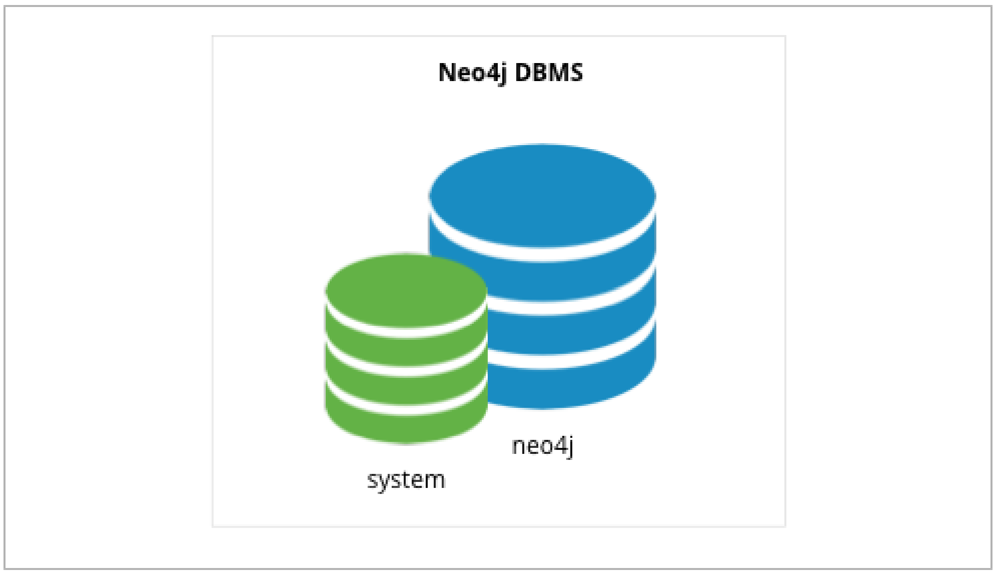


Figure 1. A default Neo4j installation.

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.

The `system` database

All installations include a built-in database named `system`, which contains meta-data and security configuration.

The `system` database behaves differently than all other databases. In particular, when connected to this database you can only perform a specific set of administrative functions, as described in detail in Cypher Manual → Database management.

Most of the available administrative commands are restricted to users with specific administrative privileges. An example of configuring security privileges is described in Fine-grained access control. Security administration is described in detail in Cypher Manual → Access Control .

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

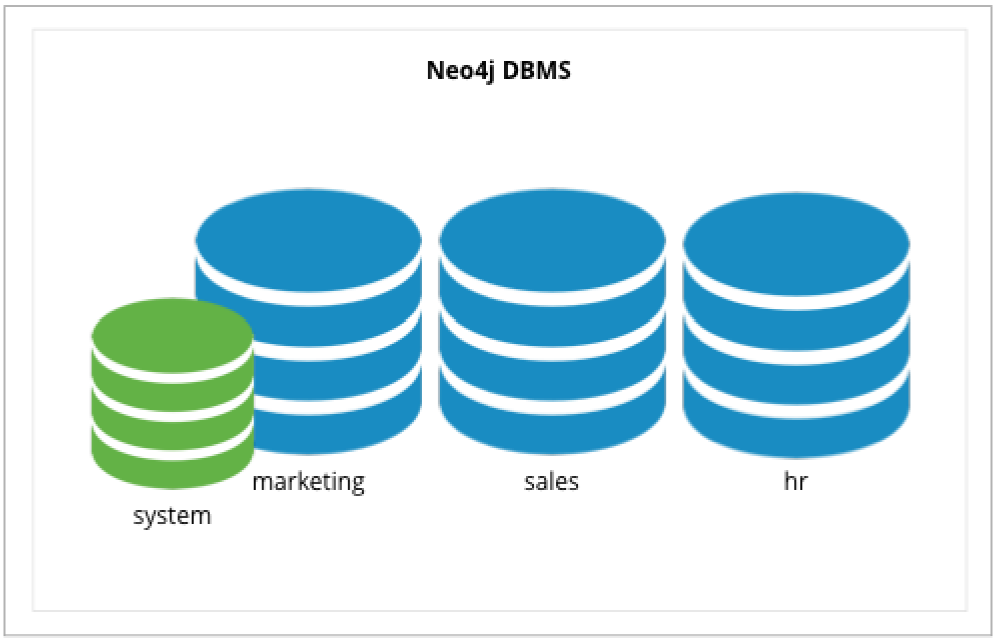


Figure 2. A multiple database Neo4j installation.

The default and home database

If a user connects to Neo4j without specifying a database, they will be connected to a home database. When choosing a home database the server will first use the home database configured for that user. If the connecting user does not have a home database configured, the server will use the default database, which every Neo4j instance has.

The default database is configurable. For details, see configuration parameters.

The following image illustrates an installation of Neo4j containing the three databases for user data, named `marketing`, `sales` and `hr`, and the `system` database. The default database is `sales`:

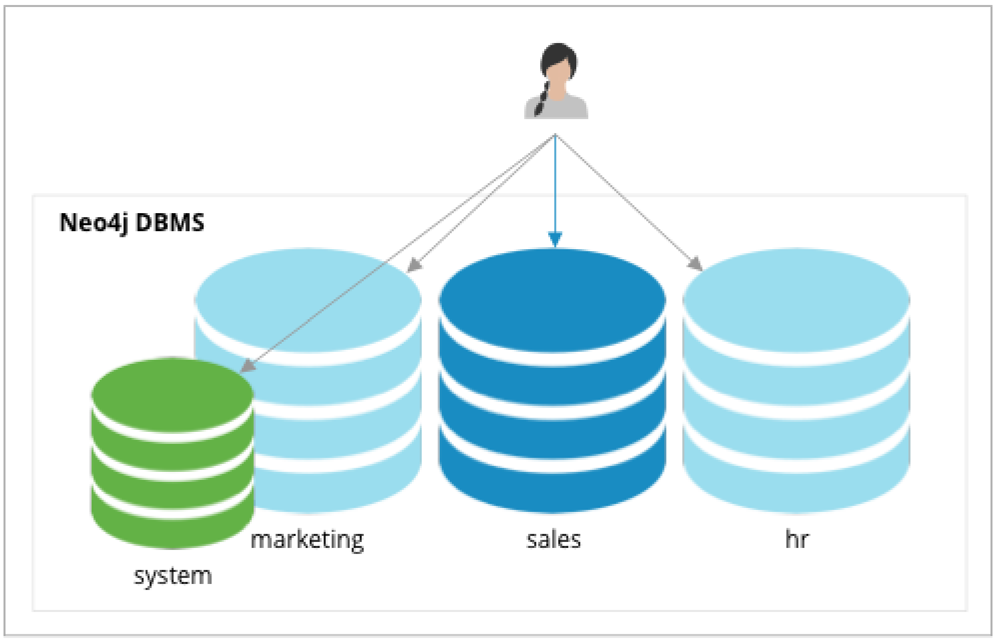


Figure 3. A multiple database Neo4j installation, with a default database.

Per-user home databases Enterprise edition

Per-user home databases are controlled via the Cypher administration commands.

To set a home database for a user, this user must exist as a record in Neo4j. Therefore, for deployments using auth providers other than native, you create a native user with a matching username and then set a home database for that user. For more information on creating native users and configuring a home database for a user, see Cypher Manual → User Management.

Administration and configuration

Administrative commands

> Administrative commands should not be used during a rolling upgrade. For more information, see Upgrade and Migration Guide → Upgrade a cluster.
>
> For detailed information on Cypher administrative commands, see Cypher Manual → Database management.

Before using administrative commands, it is important to understand the difference between stopped databases, and dropped databases:

- Databases that are stopped with the `STOP` command are completely shutdown, and may be started again through the `START` command. In a cluster, as long as a database is in a shutdown state, it can not be considered available to other members of the cluster. It is not possible to do online backups against shutdown databases and they need to be taken into special consideration during disaster recovery, as they do not have a running Raft machine while shutdown.
- Dropped databases are completely removed and are not intended to be used again at all.

The following Cypher commands are used on the `system` database to manage multiple databases:

Command	Description
`CREATE DATABASE name`	Create and start a new database. Enterprise edition
`DROP DATABASE name`	Drop (remove) an existing database. Enterprise edition
`ALTER DATABASE name`	Alter (modify) an existing database. Enterprise edition
`START DATABASE name`	Start a database that has been stopped.
`STOP DATABASE name`	Shut down a database.
`SHOW DATABASE name`	Show the status of a specific database.
`SHOW DATABASES`	Show the name and status of all the databases.
`SHOW DEFAULT DATABASE`	Show the name and status of the default database.
`SHOW HOME DATABASE`	Show the name and status of the home database for the current user.

Naming rules for databases are as follows:

- Length must be between 3 and 63 characters.
- The first character of a name must be an ASCII alphabetic character.
- Subsequent characters must be ASCII alphabetic or numeric characters, dots or dashes; `[a..z][0..9].-`
- Names are case-insensitive and normalized to lowercase.
- Names that begin with an underscore and with the prefix `system` are reserved for internal use.

All of the above commands are executed as Cypher commands, and the database name is subject to the standard Cypher restrictions on valid identifiers. In particular, the `-` (dash) and `.` (dot) characters are not legal in Cypher variables, and therefore names with dashes must be enclosed within back-ticks. For example, ``CREATE DATABASE `main-db` ``. Database names are the only identifier for which dots don't need to be escaped. For example, `main.db` is a valid database name.

It is possible to create an alias to refer to an existing database to avoid these restrictions. For more information, see link:https://neo4j.com/docs/cypher-manual/5/aliases#alias-management-create-database-alias,Cypher Manual → Creating database aliases>>

For detailed information on Cypher administrative commands, see Cypher Manual → Access Control.

For examples of using the Cypher administrative commands to manage multiple active databases, see Queries.

Configuration parameters

Configuration parameters are defined in the neo4j.conf file.

The following configuration parameters are applicable for managing databases:

Parameter name	Description
`initial.dbms.default_database`	Name of the default database for the Neo4j instance. The database is created if it does not exist when the instance starts. **Default value:** `neo4j` In a clustered setup, the value of `initial.dbms.default_database` is only used to set the initial default database. To change the default database at a later point, see [manage-databases-cc-default].
`server.max_databases`	Maximum number of databases that can be used in a Neo4j single instance or cluster. The number includes all the online and offline databases. The value is an integer with a minimum value of 2. Enterprise edition **Default value:** `100` Once the limit has been reached, it is not possible to create any additional databases. Similarly, if the limit is changed to a number lower than the total number of existing databases, no additional databases can be created.
`server.databases.default_to_read_only`	Default mode of all databases. If this setting is set to `true` all existing and new databases will be in read only mode, and so will prevent write queries. **Default value:** `false`

Parameter name	Description
`server.databases.read_only`	List of database names for which to prevent write queries. This set can contain also not yet existing databases, but not the `system` database. Regardless of settings of `server.databases.default_to_read_only`, `server.databases.read_only` and `dbms.databases.writable` the `system` database will never be read-only and will always accept write queries. Another way of preventing writes is to set the database access to read-only using the ALTER DATABASE command. Example configuration: `server.databases.read_only=["foo", "bar"]`
`dbms.databases.writable`	List of database names for which to accept write queries. This set can contain also not yet existing databases. The value of this setting is ignored if `server.databases.default_to_read_only` is set to `false`. If a database name is present in both sets, the database will be read-only and prevent write queries. If most of your databases would read-only with a few exceptions, it can be easier to set `config_server.databases.default_to_read_only` to `true`, and then put the names of the non read-only databases into `dbms.databases.writeable`. Example configuration: `dbms.databases.writable=["foo", "bar"]`

Although it is possible to achieve the same goal, i.e. set a database to read-only, both by using the Cypher command `ALTER DATABASE` and by using configuration parameters in `neo4j.conf`, it is important to understand the difference between the two. `ALTER DATABASE foo SET ACCESS READ ONLY` effectively sets the database `foo` to read-only *across the entire DBMS*.

Using configuration parameters is more subtle and allows you to configure access on each instance separately, in case of a cluster for example. If you use `server.databases.default_to_read_only` *all databases on that instance* are set to read-only.

If both the Cypher command and the configuration parameters are used and they contain conflicting information, the database in question is set to read-only.

Queries

For detailed information on Cypher administrative commands, see Cypher Manual → Database management.

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.

Show the status of a specific database

Example 29. `SHOW DATABASE`

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

In standalone mode:

```
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
| name    | type       | aliases | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage | default | home   | constituents |
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
| "neo4j" | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | TRUE    | TRUE   | []           |
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+

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

Or in a cluster:

```
+-----------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+
| name    | type       | aliases | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage | default   | home   | constituents |
+-----------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+
| "neo4j" | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | TRUE      | TRUE   | []           |
| "neo4j" | "standard" | []      | "read-write" | "localhost:7688" | "primary" | FALSE  | "online"
| "online"      | ""            | TRUE      | TRUE   | []           |
| "neo4j" | "standard" | []      | "read-write" | "localhost:7689" | "primary" | FALSE  | "online"
| "online"      | ""            | TRUE      | TRUE   | []           |
+-----------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+

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

Show the status of all databases

Example 30. `SHOW DATABASES`

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

In standalone mode:

```
+-----------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+
| name     | type       | aliases | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage | default | home   | constituents |
+-----------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+
| "neo4j"  | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | TRUE    | TRUE  | []           |
| "system" | "system"   | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | FALSE   | FALSE | []           |
+-----------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+

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

Or in a cluster:

```
+-----------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------+
| name     | type       | aliases | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage | default   | home   | constituents |
+-----------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------+
| "neo4j"  | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | TRUE      | TRUE  | []           |
| "neo4j"  | "standard" | []      | "read-write" | "localhost:7688" | "primary" | FALSE  | "online"
| "online"      | ""            | TRUE      | TRUE  | []           |
| "neo4j"  | "standard" | []      | "read-write" | "localhost:7689" | "primary" | FALSE  | "online"
| "online"      | ""            | TRUE      | TRUE  | []           |
| "system" | "system"   | []      | "read-write" | "localhost:7687" | "primary" | FALSE  | "online"
| "online"      | ""            | FALSE     | FALSE | []           |
| "system" | "system"   | []      | "read-write" | "localhost:7688" | "primary" | TRUE   | "online"
| "online"      | ""            | FALSE     | FALSE | []           |
| "system" | "system"   | []      | "read-write" | "localhost:7689" | "primary" | FALSE  | "online"
| "online"      | ""            | FALSE     | FALSE | []           |
+-----------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------+

6 rows available after 5 ms, consumed after another 1 ms
```

Switching between `online` and `offline` states is achieved using the `START DATABASE` and `STOP DATABASE` commands.

Note that databases hosted on offline servers are also returned by the `SHOW DATABASES` command. For such databases the `address` column displays `NULL`, the `currentStatus` column displays `unknown`, and the `statusMessage` displays `Server is unavailable`.

Show the status of the default database

The config setting `initial.dbms.default_database` defines which database is created and started by default when Neo4j starts. The default value of this setting is `neo4j`.

Example 31. `SHOW DEFAULT DATABASE`

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

In standalone mode:

```
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------+
| name    | type       | aliases | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage | constituents |
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------+
| "neo4j" | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""             | []           |
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------+

1 row available after 57 ms, consumed after another 2 ms
```

Or in a cluster:

```
+-----------------------------------------------------------------------------------------------------
-------------------------------------------------------+
| name    | type       | aliases | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage   | constituents |
+-----------------------------------------------------------------------------------------------------
-------------------------------------------------------+
| "neo4j" | "standard" | []      | "read-write" | "localhost:7687" | "primary" | FALSE  | "online"
| "online"      | ""               | []           |
| "neo4j" | "standard" | []      | "read-write" | "localhost:7688" | "primary" | TRUE   | "online"
| "online"      | ""               | []           |
| "neo4j" | "standard" | []      | "read-write" | "localhost:7689" | "primary" | FALSE  | "online"
| "online"      | ""               | []           |
+-----------------------------------------------------------------------------------------------------
-------------------------------------------------------+

3 row available after 57 ms, consumed after another 2 ms
```

You can change the default database by using `initial.dbms.default_database`, and restarting the server.

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

Create a database Enterprise edition

Example 32. 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     | type       | aliases | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage | default | home   | constituents |
+----------------------------------------------------------------------------------------------------
------------------------------------------------------------------------+
| "neo4j"  | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | TRUE    | TRUE   | []           |
| "system" | "system"   | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | FALSE   | FALSE  | []           |
| "sales"  | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | FALSE   | FALSE  | []           |
+----------------------------------------------------------------------------------------------------
------------------------------------------------------------------------+

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

Or in a cluster:

```
+----------------------------------------------------------------------------------------------------
------------------------------------------------------------------------+
| name     | type       | aliases | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage | default | home   | constituents |
+----------------------------------------------------------------------------------------------------
------------------------------------------------------------------------+
| "neo4j"  | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | TRUE    | TRUE   | []           |
| "neo4j"  | "standard" | []      | "read-write" | "localhost:7688" | "primary" | FALSE  | "online"
| "online"      | ""            | TRUE    | TRUE   | []           |
| "neo4j"  | "standard" | []      | "read-write" | "localhost:7689" | "primary" | FALSE  | "online"
| "online"      | ""            | TRUE    | TRUE   | []           |
| "system" | "system"   | []      | "read-write" | "localhost:7687" | "primary" | FALSE  | "online"
| "online"      | ""            | FALSE   | FALSE  | []           |
| "system" | "system"   | []      | "read-write" | "localhost:7688" | "primary" | TRUE   | "online"
| "online"      | ""            | FALSE   | FALSE  | []           |
| "system" | "system"   | []      | "read-write" | "localhost:7689" | "primary" | FALSE  | "online"
| "online"      | ""            | FALSE   | FALSE  | []           |
| "sales"  | "standard" | []      | "read-write" | "localhost:7687" | "primary" | FALSE  | "online"
| "online"      | ""            | FALSE   | FALSE  | []           |
| "sales"  | "standard" | []      | "read-write" | "localhost:7688" | "primary" | FALSE  | "online"
| "online"      | ""            | FALSE   | FALSE  | []           |
| "sales"  | "standard" | []      | "read-write" | "localhost:7689" | "primary" | TRUE   | "online"
| "online"      | ""            | FALSE   | FALSE  | []           |
+----------------------------------------------------------------------------------------------------
------------------------------------------------------------------------+

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

Switch a database Enterprise edition

Example 33. `:use <database-name>`

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

Create or replace a database

Example 34. `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     | type       | aliases | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage | default | home   | constituents |
+-----------------------------------------------------------------------------------------------------
------------------------------------------------------------------------+
| "neo4j"  | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | TRUE    | TRUE  | []           |
| "system" | "system"   | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | FALSE   | FALSE | []           |
| "sales"  | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | FALSE   | FALSE | []           |
+-----------------------------------------------------------------------------------------------------
------------------------------------------------------------------------+

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

Or in a cluster:

```
+-----------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+
| name     | type       | aliases | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage | default   | home   | constituents |
+-----------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+
| "neo4j"  | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | TRUE      | TRUE  | []           |
| "neo4j"  | "standard" | []      | "read-write" | "localhost:7688" | "primary" | FALSE  | "online"
| "online"      | ""            | TRUE      | TRUE  | []           |
| "neo4j"  | "standard" | []      | "read-write" | "localhost:7689" | "primary" | FALSE  | "online"
| "online"      | ""            | TRUE      | TRUE  | []           |
| "system" | "system"   | []      | "read-write" | "localhost:7687" | "primary" | FALSE  | "online"
| "online"      | ""            | FALSE     | FALSE | []           |
| "system" | "system"   | []      | "read-write" | "localhost:7688" | "primary" | TRUE   | "online"
| "online"      | ""            | FALSE     | FALSE | []           |
| "system" | "system"   | []      | "read-write" | "localhost:7689" | "primary" | FALSE  | "online"
| "online"      | ""            | FALSE     | FALSE | []           |
| "sales"  | "standard" | []      | "read-write" | "localhost:7687" | "primary" | FALSE  | "online"
| "online"      | ""            | FALSE     | FALSE | []           |
| "sales"  | "standard" | []      | "read-write" | "localhost:7688" | "primary" | FALSE  | "online"
| "online"      | ""            | FALSE     | FALSE | []           |
| "sales"  | "standard" | []      | "read-write" | "localhost:7689" | "primary" | TRUE   | "online"
| "online"      | ""            | FALSE     | FALSE | []           |
+-----------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+

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

```
neo4j@system> :use sales
neo4j@sales> match (n) return count(n) as countNode;
```

```
+-----------+
| countNode |
+-----------+
| 0         |
+-----------+

1 row available after 15 ms, consumed after another 1 ms
```

Stop a database

Example 35. `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     | type       | aliases | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage | default | home   | constituents |
+----------------------------------------------------------------------------------------------------
------------------------------------------------------------------------+
| "neo4j"  | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""              | TRUE    | TRUE   | []           |
| "system" | "system"   | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""              | FALSE   | FALSE  | []           |
| "sales"  | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "offline"
| "offline"     | ""              | FALSE   | FALSE  | []           |
+----------------------------------------------------------------------------------------------------
------------------------------------------------------------------------+

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

Or in a cluster:

```
+----------------------------------------------------------------------------------------------------
------------------------------------------------------------------------+
| name     | type       | aliases | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage | default | home   | constituents |
+----------------------------------------------------------------------------------------------------
------------------------------------------------------------------------+
| "neo4j"  | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""              | TRUE    | TRUE   | []           |
| "neo4j"  | "standard" | []      | "read-write" | "localhost:7688" | "primary" | FALSE  | "online"
| "online"      | ""              | TRUE    | TRUE   | []           |
| "neo4j"  | "standard" | []      | "read-write" | "localhost:7689" | "primary" | FALSE  | "online"
| "online"      | ""              | TRUE    | TRUE   | []           |
| "system" | "system"   | []      | "read-write" | "localhost:7687" | "primary" | FALSE  | "online"
| "online"      | ""              | FALSE   | FALSE  | []           |
| "system" | "system"   | []      | "read-write" | "localhost:7688" | "primary" | TRUE   | "online"
| "online"      | ""              | FALSE   | FALSE  | []           |
| "system" | "system"   | []      | "read-write" | "localhost:7689" | "primary" | FALSE  | "online"
| "online"      | ""              | FALSE   | FALSE  | []           |
| "sales"  | "standard" | []      | "read-write" | "localhost:7687" | "unknown" | FALSE  | "offline"
| "offline"     | ""              | FALSE   | FALSE  | []           |
| "sales"  | "standard" | []      | "read-write" | "localhost:7688" | "unknown" | FALSE  | "offline"
| "offline"     | ""              | FALSE   | FALSE  | []           |
| "sales"  | "standard" | []      | "read-write" | "localhost:7689" | "unknown" | FALSE  | "offline"
| "offline"     | ""              | FALSE   | FALSE  | []           |
+----------------------------------------------------------------------------------------------------
------------------------------------------------------------------------+

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

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

```
Unable to get a routing table for database 'sales' because this database is unavailable
neo4j@sales[UNAVAILABLE]>
```

Start a database

Example 36. `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     | type       | aliases | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage | default | home   | constituents |
+-----------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+
| "neo4j"  | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | TRUE    | TRUE   | []           |
| "system" | "system"   | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | FALSE   | FALSE  | []           |
| "sales"  | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | FALSE   | FALSE  | []           |
+-----------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+

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

Or in a cluster:

```
+-----------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+
| name     | type       | aliases | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage | default | home   | constituents |
+-----------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+
| "neo4j"  | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | TRUE    | TRUE   | []           |
| "neo4j"  | "standard" | []      | "read-write" | "localhost:7688" | "primary" | FALSE  | "online"
| "online"      | ""            | TRUE    | TRUE   | []           |
| "neo4j"  | "standard" | []      | "read-write" | "localhost:7689" | "primary" | FALSE  | "online"
| "online"      | ""            | TRUE    | TRUE   | []           |
| "system" | "system"   | []      | "read-write" | "localhost:7687" | "primary" | FALSE  | "online"
| "online"      | ""            | FALSE   | FALSE  | []           |
| "system" | "system"   | []      | "read-write" | "localhost:7688" | "primary" | TRUE   | "online"
| "online"      | ""            | FALSE   | FALSE  | []           |
| "system" | "system"   | []      | "read-write" | "localhost:7689" | "primary" | FALSE  | "online"
| "online"      | ""            | FALSE   | FALSE  | []           |
| "sales"  | "standard" | []      | "read-write" | "localhost:7687" | "primary" | FALSE  | "online"
| "online"      | ""            | FALSE   | FALSE  | []           |
| "sales"  | "standard" | []      | "read-write" | "localhost:7688" | "primary" | FALSE  | "online"
| "online"      | ""            | FALSE   | FALSE  | []           |
| "sales"  | "standard" | []      | "read-write" | "localhost:7689" | "primary" | TRUE   | "online"
| "online"      | ""            | FALSE   | FALSE  | []           |
+-----------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+

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

Drop or remove a database Enterprise edition

Example 37. DROP DATABASE

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

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

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

```
+-----------------------------------------------------------------------------------------------------
------------------------------------------------------------------------+
| name     | type       | aliases | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage | default | home   | constituents |
+-----------------------------------------------------------------------------------------------------
------------------------------------------------------------------------+
| "neo4j"  | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | TRUE    | TRUE  | []           |
| "system" | "system"   | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | FALSE   | FALSE | []           |
+-----------------------------------------------------------------------------------------------------
------------------------------------------------------------------------+

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

Error handling

When running the database management queries, such as `CREATE DATABASE`, it is possible to encounter errors.

Observing errors

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

Example 38. Fail to create a database

```
neo4j@system> CREATE DATABASE foo;
```

```
0 rows available after 108 ms, consumed after another 0 ms
```

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

In standalone mode:

```
+-----------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
| name    | type       | aliases | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage              | default | home   | constituents |
+-----------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
| "foo"   | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "dirty"       | "File system permissions" | FALSE   | FALSE | []           |
+-----------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+

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

In a cluster:

```
+-----------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
| name    | type       | aliases | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage              | default | home   | constituents |
+-----------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
| "foo"   | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""                        | FALSE   | FALSE | []           |
| "foo"   | "standard" | []      | "read-write" | "localhost:7688" | "primary" | FALSE  | "online"
| "online"      | ""                        | FALSE   | FALSE | []           |
| "foo"   | "standard" | []      | "read-write" | "localhost:7689" | "primary" | FALSE  | "online"
| "dirty"       | "File system permissions" | FALSE   | FALSE | []           |
+-----------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+

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

Database states

A database management operation may fail for a number of reasons. For example, if the file system instance has incorrect permissions, or Neo4j itself is misconfigured. As a result, the contents of the `statusMessage` column in the `SHOW DATABASE` query results may vary significantly.

However, databases may only be in one of a select number of states:

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`.

While `dropped` is a valid database state, it is only transiently observable, as database records are removed from `SHOW DATABASE` results once the `DROP` operation is complete.

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.

If a database is in the `quarantined` state, retrying the last operation will not work.

Example 39. Retry to start a database

```
neo4j@system> START DATABASE foo;
```

```
0 rows available after 108 ms, consumed after another 0 ms
```

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

```
+-----------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------+
| name   | type       | aliases | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage             | default | home   | constituents |
+-----------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------+
| "foo"  | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "offline"     | "File system permissions" | FALSE   | FALSE | []           |
+-----------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------+

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

After investigating and addressing the underlying issue, you can start the database again and verify that it is running properly:

```
neo4j@system> START DATABASE foo;
```

```
0 rows available after 108 ms, consumed after another 0 ms
```

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

```
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
| name     | type       | aliases | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage | default | home   | constituents |
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
| "foo"    | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | FALSE   | FALSE | []           |
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+

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

If repeated retries of a command have no effect, or if a database is in a `dirty` state, you may drop and recreate the database, as detailed in Cypher manual → Database management.

When running `DROP DATABASE` as part of an error handling operation, you can also append `DUMP DATA` to the command. It produces a database dump that can be further examined and potentially repaired.

Quarantined databases

There are two ways to get a database into a `quarantined` state:

- By using the `dbms.quarantineDatabase` procedure locally to isolate a specific database. The procedure must be executed on the instance whose copy of the database you want to quarantine. A reason for that can be, for example, when a database is unable to start on a given instance due to a file system permissions issue with the volume where the database is located or when a recently started database begins to log errors. The quarantine state renders the database inaccessible on that instance and prevents its state from being changed, for example, with the `START DATABASE` command.

  > If running in a cluster, database management commands such as `START DATABASE foo` will still take effect on the instances which have **not** quarantined `foo`.

- When a database encounters a severe error during its normal run, which prevents it from a further operation, Neo4j stops that database and brings it into a `quarantined` state. Meaning, it is not possible to restart it with a simple `START DATABASE` command. You have to execute `CALL dbms.quarantineDatabase(databaseName, false)` on the instance with the failing database in order to lift the quarantine.

After lifting the quarantine, the instance will automatically try to bring the database to the desired state.

> It is recommended to run the quarantine procedure over the `bolt://` protocol rather than `neo4j://`, which may route requests to unexpected instances.

Syntax:

```
CALL dbms.quarantineDatabase(databaseName,setStatus,reason)
```

Arguments:

Name	Type	Description
`databaseName`	String	The name of the database that will be put into or removed from quarantine.
`setStatus`	Boolean	`true` for placing the database into quarantine; `false` for lifting the quarantine.
`reason`	String	(Optional) The reason for placing the database in quarantine.

Returns:

Name	Type	Description
`databaseName`	String	The name of the database.
`quarantined`	String	Actual state.

Name	Type	Description
`result`	String	Result of the last operation. The result contains the user, the time, and the reason for the quarantine.

The `dbms.quarantineDatabase` procedure replaces `dbms.cluster.quarantineDatabase`, which has been deprecated in Neo4j 4.3 and will be removed with the next major version.

Quarantine a database

```
neo4j@system> CALL dbms.quarantineDatabase("foo",true);
```

```
+--------------------------------------------------------------------------------------+
| databaseName | quarantined | result                                                  |
+--------------------------------------------------------------------------------------+
| "foo"        | TRUE        | "By neo4j at 2020-10-15T15:10:41.348Z: No reason given" |
+--------------------------------------------------------------------------------------+

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

Check if a database is quarantined

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

```
+-----------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------+
| name  | type       | aliases | access       | address           | role      | writer | requestedStatus |
currentStatus | statusMessage                                           | default | home   | constituents |
+-----------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------+
| "foo" | "standard" | []      | "read-write" | "localhost:7688" | "unknown" | FALSE  | "online"        |
"quarantined" | "By neo4j at 2020-10-15T15:10:41.348Z: No reason given" | FALSE    | FALSE | []           |
| "foo" | "standard" | []      | "read-write" | "localhost:7689" | "primary" | FALSE  | "online"        |
"online"      | ""                                                      | FALSE    | FALSE | []           |
| "foo" | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"        |
"online"      | ""                                                      | FALSE    | FALSE | []           |
+-----------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------+

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

A `quarantined` state is persisted for user databases. This means that if a database is quarantined, it will remain so even if that Neo4j instance is restarted. You can remove it only by running the `dbms.quarantineDatabase` procedure on the instance where the quarantined database is located, passing `false` for the `setStatus` parameter.

The one exception to this rule is for the built-in `system` database. Any quarantine for that database is removed automatically after instance restart.

Connecting remote databases

A remote database alias can be used to provide connection to one or more graphs, on one or more remote standalone servers or clusters.

Although remote database aliases do not store any data, they enable users or applications to perform queries on remote databases as if they were on the local DBMS server. All configuration can be done with Administration commands on a running system. Any changes are automatically synchronized across all instances of a cluster.

The following steps describe the setup required to define a remote database alias both for local and remote DBMSs. They are also useful should there be any changes in the name of the databases, or the location of standalone servers and cluster instances. Additionally, you will also find information here on best practices and additional guidance on any changes in the configuration.

Setup example

In this example, *Alice* is an administrator and *Carol* is a user who needs access to a database managed by *Bob*:

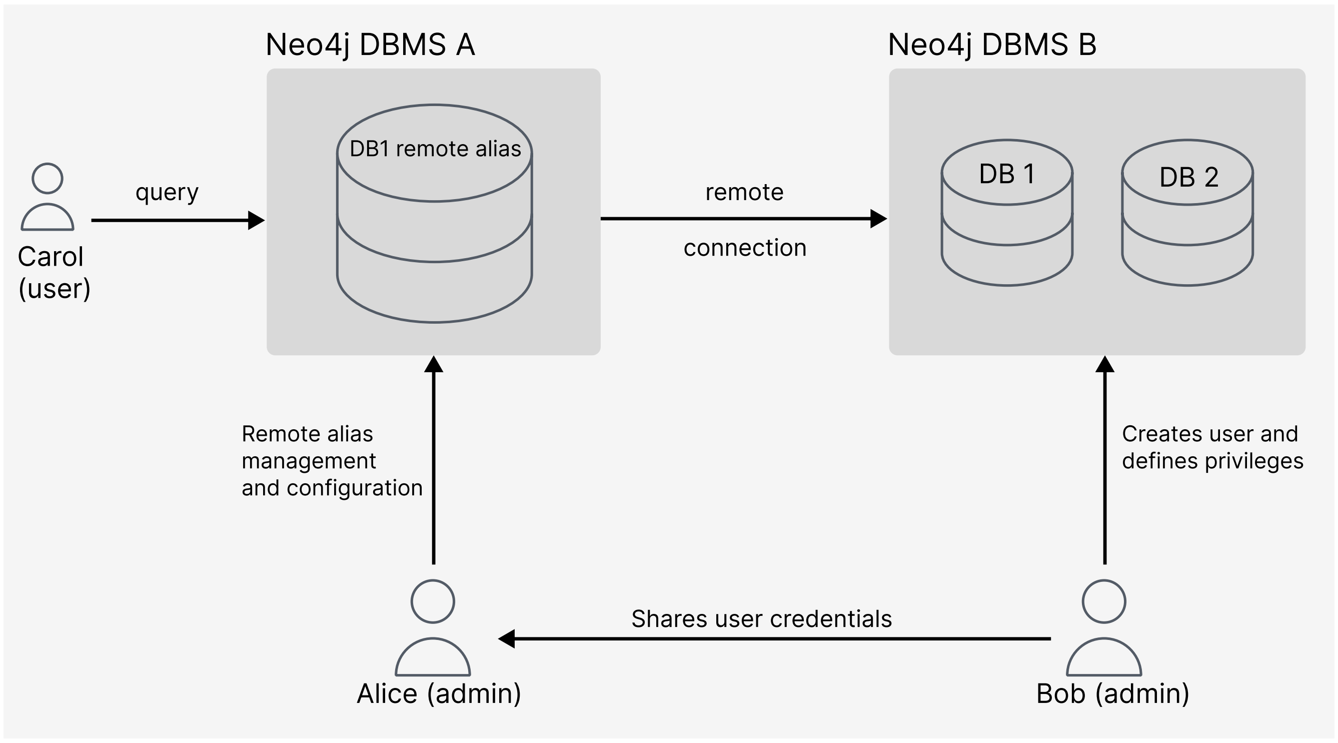


Figure 4. Overview of the required remote database alias setup

A remote alias defines:

- Which user of the remote **DBMS B** is used.
- Where the remote database is located.
- How to connect to the remote database using driver settings.

A remote database alias is only accessible to users with appropriate privileges. In the example above, *Bob* is the administrator responsible for deciding which databases (`Db1` or `Db2`) the remote aliases can write and/or read. Meanwhile, *Alice* is the administrator that assigns who has access to the privileges set by *Bob*. In the example, *Alice* will assign that access to *Carol*.

See Cypher manual → DBMS administration for more information.

Carol can use her own regular credentials to access the remote database `Db1` in DBMS after *Alice* assigns this privilege to her user profile. This configuration will also allow *Carol* to access `Db2` in **DBMS B**. If the administrators decide this should not be the case, then *Bob* must define the appropriate privileges (see Cypher manual → Access control for further information).

Configuration of a remote DBMS (*Bob*)

In the suggested example, there are two administrators: *Alice* and *Bob*. *Bob* is the administrator responsible for the setup of the remote DBMS, which includes the following steps:

1. Create the user profile to share with *Alice*. Currently, only user and password-based authentication (e.g. native authentication) are supported.
2. Define the permissions for the user. If you don't want this user to access `Db2`, here is where you set it.
3. Securely transmit the credentials to *Alice*, setting up the link to database `Db1`. It is recommended to create a custom role to track all users shared on a remote connection, so that they remain trackable.

```
CREATE USER alice SET PASSWORD 'secret'
CREATE ROLE remote
GRANT ACCESS ON DATABASE neo4j TO remote
GRANT MATCH {*} ON GRAPH neo4j TO remote
GRANT ROLE remote TO alice
```

In this case, *Bob* must do the required setup for the SSL framework and check whether the database accepts a non-local connection if required.

```
# accept non-local connections
server.default_listen_address=0.0.0.0

# configure ssl for bolt
dbms.ssl.policy.bolt.enabled=true
dbms.ssl.policy.bolt.base_directory=certificates/bolt
dbms.ssl.policy.bolt.private_key=private.key
dbms.ssl.policy.bolt.public_certificate=public.crt
dbms.ssl.policy.bolt.client_auth=NONE

# enforcing ssl connection
server.bolt.tls_level=REQUIRED
```

Configuration of a DBMS with a remote database alias (*Alice*)

As *Alice*, you need to generate an encryption key. In this case, the credentials of a user of **DBMS B** are reversibly encrypted and stored in the system database of **DBMS A**.

Since the algorithm used is AES/GCM, an AES encryption key needs to be provided. It should have length 256, and be stored in a password-protected keystore, of format pkcs12.

The key can be generated by using the following keytool command in your terminal, which is included in Java Platform, Standard Edition:

```
keytool -genseckey -keyalg aes -keysize 256 -storetype pkcs12 -keystore [keystore-name] -alias [key-name]
-storepass [keystore-password]
```

It is recommended to generate the keystore on the same Java version on which Neo4j is run, as the supported encryption algorithms may vary. For details on the version of Java required by Neo4j, see System requirements → Java.

Set configuration

The following configuration is necessary for creating a remote database alias:

Configuration	Description
`dbms.security.keystore.path`	The absolute path to the keystore file, including the file name.
`dbms.security.keystore.password`	The password to the keystore file. Use Command expansion to set the password.
`dbms.security.key.name`	The name of the secret key

To prevent unauthorized access, the keystore file must be stored in a trusted location. This is the main way to protect the encrypted passwords which will be stored on the system database. It shouldn't be accessible to any user except for the administrator and `neo4j`, for whom the keystore file must be readable.

In a cluster, *Alice* needs to share the same keystore file over all instances.

For example, these would be valid additions to the config when using the suggested keytool command:

```
dbms.security.keystore.path=/home/secure-folder/keystore-name.pkcs12
dbms.security.keystore.password=$(conf/password.sh)
dbms.security.key.name=key-name
```

Where `password.sh` might look like this:

```
#!/bin/bash
echo "$KEYSTORE_PASSWORD_ENVIRONMENT_VARIABLE"
```

Additionally, don't forget to change the permissions of the configuration file, and start Neo4j with the command expansion flag:

```
chmod 640 conf/neo4j.conf
bin/neo4j start --expand-commands
```

Managing remote database aliases

You can use the alias commands to manage remote database aliases. In this case, it is strongly recommended to connect to a remote database alias with a secured connection.

Please note that only client-side SSL is supported. By default, remote aliases require a secured URI scheme such as `neo4j+s`. This can be disabled by setting the driver setting `ssl_enforced` to `false`.

For example, the following command can be used to create a remote database alias:

```
CREATE ALIAS `remote-neo4j` FOR DATABASE `neo4j` AT "neo4j+s://location:7687" USER alice PASSWORD 'secret'
```

In order to do so, either database management or alias management privileges are required. The permission to create an alias can be granted like this:

```
GRANT CREATE ALIAS ON DBMS TO administrator
```

Here is how to grant the `ACCESS` privileges to use the remote database alias:

```
GRANT ACCESS ON DATABASE `remote-neo4j` TO role
```

If a transaction modifies an alias (e.g. changing the database targeted on DBMS B), other transactions concurrently executing against that alias may be aborted and rolled back for safety. This prevents issues such as a transaction executing against multiple target databases for the same alias.

Changing the encryption key

If the encryption key in the keystore is changed, the encrypted credentials for existing remote database aliases will need to be updated as they will no longer be readable.

If there is a failure when reading the keystore file, investigate the `debug.log` to find out which parameter is the source of the problem. In case it is not possible to connect to the remote alias after its creation, verify its settings by connecting to the remote database at https://browser.neo4j.io/ or at your local browser.

User connection to remote database aliases

A user can connect to a remote database alias the same way they would do to a database. This includes:

- Connecting directly to the remote database alias.
- The `USE` clause enables a user to query a remote database alias that they are not directly connected to:

```
USE `remote-neo4j` MATCH (n) RETURN *
```

- Connecting to a remote database alias as a home database. This needs to be set by Administrator A. See more about User Management.

```
ALTER USER alice SET HOME DATABASE `remote-neo4j`
```

Remote alias transactions will not be visible in `SHOW TRANSACTIONS` on DBMS A. However, they can be accessed and terminated on the remote database when connecting with the same user.

Composite databases

Introduction

Overview

As of Neo4j 5, the Fabric technology has been extended by *composite databases* and, as such, these replace what was known as Fabric in Neo4j 4.x. Composite databases is a functionality which allows queries that access multiple graphs at once.

Composite databases enable:

- **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.**

Concepts

Composite databases don't store any data on their own, but rather give access to the graphs found on other databases. These graphs are added to the composite database by means of database aliases.

A Neo4j DBMS can have multiple composite databases, and they can be created both in single-instance deployments, and in cluster deployments.

Composite databases are managed using administrative commands. They are created with the `CREATE COMPOSITE DATABASE` command.

Example 40. Creating a composite database

```
CREATE COMPOSITE DATABASE cineasts
```

Constituent graphs are added with the `CREATE ALIAS` administrative command, for example:

Example 41. Creating an alias on a composite database

```
CREATE ALIAS cineasts.latest
  FOR DATABASE movies2022
```

Aliases can also be created for databases on other DBMSs:

Example 42. Creating an alias for a remote database on a composite database

```
CREATE ALIAS cineasts.upcoming
  FOR DATABASE upcoming
  AT 'neo4j+s://other.dbms.com'
  USER $user
  PASSWORD $password
```

The `SHOW DATABASE` administrative command includes composite databases.

Their `type` is reported as `"composite"`, and the `constituents` column lists the names of the aliases contained.

Example 43. Showing a composite database

```
SHOW DATABASE cineasts YIELD name, type, constituents
```

```
+----------------------------------------------------------------------+
| name       | type        | constituents                              |
+----------------------------------------------------------------------+
| "cineasts" | "composite" | ["cineasts.latest", "cineasts.upcoming"] |
+----------------------------------------------------------------------+
```

The `SHOW ALIASES FOR DATABASE` administrative command can be used to inspect aliases on composite databases in further detail.

Example 44. Showing composite database aliases

```
SHOW ALIASES FOR DATABASE
```

```
+-----------------------------------------------------------------------------------------+
| name                | database     | location | url                         | user      |
+-----------------------------------------------------------------------------------------+
| "cineasts.latest"   | "movies2022" | "local"  | NULL                        | NULL      |
| "cineasts.upcoming" | "upcoming"   | "remote" | "neo4j+s://other.dbms.com" | "cineast" |
+-----------------------------------------------------------------------------------------+
```

For a full description of the administrative commands for managing composite databases, see Cypher Manual → Database management.

Querying

The examples featured in this section make use of the two Cypher clauses: `CALL {}`.

Connecting drivers and applications

Drivers and client applications connect to composite databases just like standard databases. For more information, see *Databases and execution context* in the Neo4j Driver manuals.

Graph selection

Queries submitted to a composite database may contain several `USE` clauses that direct different parts of the query to different constituent graphs.

Each constituent graph is named after the alias that introduces it into the composite database.

The examples will assume the same setup as the one created in the Introduction

Query a single graph

Example 45. Reading and returning data from a single graph

```
USE cineasts.latest
MATCH (movie:Movie)
RETURN movie.title AS title
```

The `USE` clause at the beginning of the query selects the `cineasts.latest` graph for all the subsequent clauses. `MATCH` is performed on that graph.

Query multiple graphs

Example 46. Reading and returning data from two graphs

```
USE cineasts.latest
MATCH (movie:Movie)
RETURN movie.title AS title
  UNION
USE cineasts.upcoming
MATCH (movie:Movie)
RETURN movie.title AS title
```

The first part of the `UNION` query selects the `cineasts.latest` graph and the second part selects the `cineasts.upcoming` graph.

Dynamic graph access

Queries can also select constituent graphs dynamically, using the form `USE graph.byName(graphName)`.

Example 47. Reading and returning data from dynamically selected graphs

```
UNWIND ['cineasts.latest', 'cineasts.upcoming'] AS graphName
CALL {
  USE graph.byName(graphName)
  MATCH (movie:Movie)
  RETURN movie
}
RETURN movie.title AS title
```

In the example above, the part of the query accessing graph data, `MATCH (movie:Movie)`, is wrapped in a sub-query with a dynamic `USE` clause. `UNWIND` is used to get the names of our graphs, each on one row. The `CALL {}` sub-query executes once per input row. In this case, once selecting `cineasts.latest`, and once selecting `cineasts.upcoming`.

Listing graphs

The built-in function `graph.names()` returns a list containing the names of all constituent graphs on the current composite database.

Example 48. The `graph.names()` function

```
UNWIND graph.names() AS graphName
RETURN graphName
```

```
+---------------------+
| graphName           |
+---------------------+
| "cineasts.latest"   |
| "cineasts.upcoming" |
+---------------------+
```

The names returned by this function can be used for dynamic graph access.

Example 49. Reading and returning data from all graphs

```
UNWIND graph.names() AS graphName
CALL {
  USE graph.byName(graphName)
  MATCH (movie:Movie)
  RETURN movie
}
RETURN movie.title
```

Query result aggregation

Example 50. Getting the earliest release year of all movies from all graphs

```
UNWIND graph.names() AS graphName
CALL {
  USE graph.byName(graphName)
  MATCH (movie:Movie)
  RETURN movie.released AS released
}
RETURN min(released) AS earliest
```

The sub-query returns the `released` property of each movie, from each constituent graph. The `RETURN` at the end of the main query aggregates across the full result to calculate the global minimum.

Correlated subqueries

This query finds all movies in `cineasts.upcoming` that are to be released in the same month as the longest movie in `cineasts.latest`.

Example 51. Correlated subquery

```
CALL {
  USE cineasts.latest
  MATCH (movie:Movie)
  RETURN movie.releasedMonth AS monthOfLongest
    ORDER BY movie.runningTime DESC
    LIMIT 1
}
CALL {
  USE cineasts.upcoming
  WITH monthOfLongest
  MATCH (movie:Movie)
  WHERE movie.releasedMonth = monthOfLongest
  RETURN movie
}
RETURN movie
```

The first part of the query finds the movie with the longest running time from `cineasts.latest`, and returns its release month. The second part queries for all movies in `cineasts.upcoming` that fulfill our condition and returns them. The sub-query imports the `monthOfLongest` variable using `WITH monthOfLongest`, to make it accessible.

Updates

Composite database queries can perform updates to constituent graphs.

Example 52. Constituent graph update

```
USE cineasts.upcoming
CREATE (:Movie {title: 'Dune: Part Two'})
```

Updates can only be performed on a single constituent graph per transaction.

Example 53. Multi-graph update will fail

```
UNWIND graph.names() AS graphName
CALL {
  USE graph.byName(graphName)
  CREATE (:Movie {title: 'The Flash'})
}
```

```
Writing to more than one database per transaction is not allowed.
```

Limitations

Queries on composite databases have a few limitations.

Graph accessing operations

Consider a composite database query:

```
UNWIND graph.names() AS graphName
CALL {
  USE graph.byName(graphName)
  MATCH (movie:Movie)
  RETURN movie
}
RETURN movie
```

Here the outer clauses, i.e. the `UNWIND`, the `CALL` itself, and the final `RETURN`, appear in the root scope of the query, without a specific chosen graph. Clauses or expressions in scopes where no graph has been specified must not be graph-accessing.

See examples of graph-accessing operations:

- `MATCH (n)`
- `CREATE (:X)`
- `WITH [p=()--() | p] AS paths`

Examples of non-graph-accessing operations:

- `RETURN 1 + 2 AS number`
- `WITH node.property AS val`

Nested `USE` clauses

An inner scope must use the same graph as its outer scope:

```
USE cineasts.latest
MATCH (n)
CALL {
  USE cineasts.upcoming
  MATCH (m)
  RETURN m
}
RETURN n, m
```

```
Nested subqueries must use the same graph as their parent query.
Attempted to access graph cineasts.upcoming
"    USE cineasts.upcoming"
     ^
```

Sub-queries without a USE clause can be nested. They inherit the specified graph from the outer scope.

```
CALL {
  USE cineasts.upcoming
  CALL {
    MATCH (m:Movie)
    RETURN m
  }
  RETURN m
}
RETURN m
```

Built-in graph functions

Graph functions are located in the namespace graph. The following table provides a description these functions:

Table 34. Built-in graph functions

Function	Explanation
graph.names()	Provides a list of names of all constituent graphs on the current composite database.
graph.byName(graphName)	Used with the USE clause to select a constituent graph by name dynamically. This function is supported only with USE clauses.

Sharding data with the copy command

The copy command can be used to filter out data when creating database copies. In the following example, a sample database is separated into 3 shards.

Example 54. Use the `copy` command to filter out data

The sample database contains the following data:

```
(p1 :Person :S2 {id:123, name: "Ava"})
(p2 :Person :S2 {id:124, name: "Bob"})
(p3 :Person :S3 {id:125, name: "Cat", age: 54})
(p4 :Person :S3 {id:126, name: "Dan"})
(t1 :Team :S1 :SAll {id:1, name: "Foo", mascot: "Pink Panther"})
(t2 :Team :S1 :SAll {id:2, name: "Bar", mascot: "Cookie Monster"})
(d1 :Division :SAll {name: "Marketing"})
(p1)-[:MEMBER]->(t1)
(p2)-[:MEMBER]->(t2)
(p3)-[:MEMBER]->(t1)
(p4)-[:MEMBER]->(t2)
```

The data has been prepared using queries to add the labels `:S1`,`:S2`, `:S3`, and `:SAll`, which denotes the target shard. *Shard 1* contains the team data. *Shard 2* and *Shard 3* contain person data.

1. Create *Shard 1* with:

   ```
   $neo4j-home> bin/neo4j-admin database copy neo4j shard1 \
      --copy-only-nodes-with-labels=S1,SAll \ ①
      --skip-labels=S1,S2,S3,SAll ②
   ```

   ① The `--copy-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 database copy neo4j shard2 \
      --copy-only-nodes-with-labels=S2,SAll \
      --skip-labels=S1,S2,S3,SAll \
      --copy-only-node-properties=Team.id
   ```

   In *Shard 2*, you want to keep the `:Team` nodes as proxy nodes, to be able to link together information from the separate shards. The nodes will be included since they have the label `:SAll`, but you specify `--copy-only-node-properties` so as to not duplicate the team information from *Shard 1*.

   ```
   (p1 :Person {id:123, name: "Ava"})
   (p2 :Person {id:124, name: "Bob"})
   (t1 :Team {id:1})
   (t2 :Team {id:2})
   (d1 :Division {name: "Marketing"})
   (p1)-[:MEMBER]->(t1)
   (p2)-[:MEMBER]->(t2)
   ```

Observe that --copy-only-node-properties did not filter out Person.name since the :Person label was not mentioned in the filter.

3. Create *Shard 3*, but with the filter --skip-node-properties, instead of --copy-only-node-properties.

```
$neo4j-home> bin/neo4j-admin database copy neo4j shard3 \
   --copy-only-nodes-with-labels=S3,SAll \
   --skip-labels=S1,S2,S3,SAll \
   --skip-node-properties=Team.name,Team.mascot
```

The result is:

```
(p3 :Person {id:125, name: "Cat", age: 54})
(p4 :Person {id:126, name: "Dan"})
(t1 :Team {id:1})
(t2 :Team {id:2})
(d1 :Division {name: "Marketing"})
(p3)-[:MEMBER]->(t1)
(p4)-[:MEMBER]->(t2)
```

As demonstrated, you can achieve the same result with both --skip-node-properties and --copy-only-node-properties. In this example, it is easier to use --copy-only-node-properties because only one property should be kept. The relationship property filters works in the same way.

Further considerations

Sharding an existing database

An existing database can be sharded with the help of the neo4j-admin database copy command. See Sharding data with the copy command for an example.

Composite database transactions

Composite databases do not support WRITE transactions that span across multiple graphs. To avoid common mistakes that may lead to data corruption, WRITE operations on more than one graph within the same transaction, are not allowed. Transactions with queries that read from multiple graphs, or read from multiple graphs and write to a single graph, are allowed.

Neo4j embedded

Composite databases are not available when Neo4j is used as an embedded database in Java applications. Composite databases 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.

Clustering

This chapter describes the following:

- Introduction — An overview of the clustering architecture.
- Set up a cluster — The basics of configuring and deploying a new cluster.
  - Deploy a basic cluster — How to set up a basic cluster.
  - Cluster server discovery — How servers in a cluster discover each other.
  - Leadership, routing and load balancing — Election of leaders, routing and load balancing.
  - Intra-cluster encryption — How to secure the cluster communication.
- Managing servers in a cluster — How to manage manage the servers in a cluster.
- Managing databases in a cluster — How to manage the databases in a cluster.
- Monitoring — Monitoring of a cluster.
  - Monitoring servers — The tools available for monitoring the servers in a cluster.
  - Monitoring databases — The tools available for monitoring the databases in a cluster.
- Settings reference — A summary of the most important cluster settings.
- Clustering glossary — A glossary of terms used in the clustering documentation. Further information:

Introduction

Overview

Neo4j's clustering provides these main features:

1. **Safety:** Servers hosting databases in primary mode provide a fault tolerant platform for transaction processing which remains available while a simple majority of those Primary Servers are functioning.
2. **Scale:** Servers hosting databases in secondary mode 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.
4. **Operability:** Database management is separated from server management.

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. Additionally, the administration of a cluster is uncomplicated, including scaling the size of the cluster and distributing and balancing the available resources.

The remainder of this section contains an overview of how clustering works in production, including both operational and application aspects.

Operational view

From an operational point of view, it is useful to view the cluster as a homogenous pool of servers which run a number of databases. The servers have two different database-hosting capabilities, referred to as *Primary* and *Secondary* modes. A server can simultaneously act as a primary host for one or more databases and as a secondary host for other databases. Similarly, it is possible for a database to be hosted on only one server, even when that server is part of a cluster. In such cases, the server is always hosting that database in primary mode.

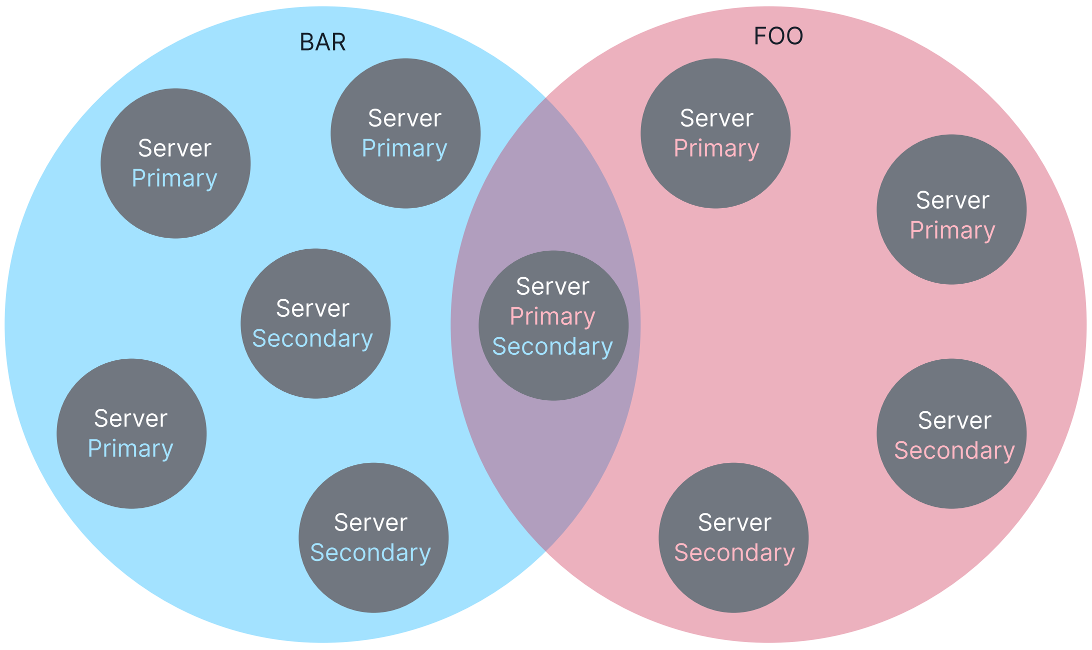


Figure 5. Cluster Architecture

The two modes 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.

Primary mode

A server hosting a database in primary mode allows read and write operations. A database can be hosted by one or more primary hosts.

To achieve high availability, a database should be created with multiple primaries. If high availability is not required, then a database may be created with a single primary for minimum write latency. The remainder of this section assumes a database has multiple primaries.

Database primaries achieve high availability by replicating all transactions using the Raft protocol. Raft ensures that the data is safely durable by waiting for a majority of primaries in a database (N/2+1) to acknowledge a transaction, before acknowledging its commit to the end user application. In practice, only one of the multiple primaries execute write transactions from clients. This writer is elected automatically from amongst a database's primaries and may change over time. The writer primary synchronously replicates writes to the other primaries. The database secondaries replicates the writes asynchronously from more up-to-date members of the cluster.

This synchronous replication has an impact on write transaction latency. Implicitly, write transactions are acknowledged by the fastest majority, but as the number of primaries of the database grows, so does the size of the majority needed to acknowledge a write.

The fault tolerance for a database is calculated with the formula M = 2F + 1, where M is the number of primaries required to tolerate F faults. For example:

- In order to tolerate two failed primaries, you need a topology of five servers hosting your database in primary mode.
- The smallest fault-tolerant cluster, a cluster that can tolerate one fault, must have three database primaries.
- It is also possible to create a cluster consisting of only two primaries. However, that cluster is not fault-tolerant. If one of the two servers fails, the remaining server becomes read-only.
- A database with a single primary server cannot tolerate any faults either. Therefore it is recommended to have three or more primaries to achieve high availability.

ℹ	With database primaries, should the database suffer enough primary failures, it can no longer process writes and becomes read-only to preserve safety.

Secondary mode

Database secondaries are asynchronously replicated from primaries via transaction log shipping. They periodically poll an upstream server for new transactions and have these shipped over. Many secondaries can be fed data from a relatively small number of primaries, allowing for a large fan out of the query workload for scale.

Databases can typically have relatively large numbers of secondaries. Losing a secondary does not impact the database’s availability, aside from the loss of its fraction of graph query throughput. It does not affect the fault tolerance of the database.

The main responsibility of database secondaries is to scale out read workloads. Secondaries act like caches for the graph data and are fully capable of executing arbitrary (read-only) queries and procedures.

Due to its asynchronous nature, secondaries may not provide all transactions committed on the primary server(s).

Causal consistency

While the operational mechanics of the cluster are interesting from an application point of view, it is also helpful to think about how applications use the database to get their work done. In many applications, it is typically desirable to both read from the graph and write to the graph. Depending on the nature of the workload, it is common to want reads from the graph to take into account previous writes to ensure causal consistency.

Causal consistency is one of numerous consistency models used in distributed computing. It ensures that causally related operations are seen by every instance in the system in the same order. Consequently, client applications are guaranteed to read their own writes, regardless of which instance they communicate with. This simplifies interaction with large clusters, allowing clients to treat them as a single (logical) server.

Causal consistency makes it possible to write to databases hosted on servers in primary mode (where data is safe) and read those writes from databases hosted on servers in secondary mode (where graph operations are scaled out). For example, causal consistency guarantees that the write which created a user account is present when that same user subsequently attempts to log in.

On executing a transaction, the client can ask for a bookmark which it then presents as a parameter to subsequent transactions. Using that bookmark, the cluster can ensure that only servers which have processed the client's bookmarked transaction will run its next transaction. This provides a *causal chain* which ensures correct read-after-write semantics from the client's point of view.

Aside from the bookmark everything else is handled by the cluster. The database drivers work with the cluster topology manager to choose the most appropriate servers to route queries to. For instance, routing reads to database secondaries and writes to database primaries.

Deploy a basic cluster

The first step in setting up a cluster infrastructure is configuring a number of servers to form a cluster that you can run your databases on. The following configuration settings are important to consider when deploying a new cluster. See also Settings reference for more detailed descriptions and examples.

Table 35. Important settings for clusters

Option name	Description
`server.default_advertised_address`	The address that other machines are told to connect to. In the typical case, this should be set to the fully qualified domain name or the IP address of this server.
`server.default_listen_address`	The address or network interface this machine uses to listen for incoming messages. Setting this value to `0.0.0.0` makes Neo4j bind to all available network interfaces.
`dbms.cluster.discovery.endpoints`	This setting contains the network for at least one server in the cluster and must be set to the same value on all cluster members. The behavior of this setting can be modified by configuring the setting `dbms.cluster.discovery.type`. This is described in detail in [clustering-discovery]
`initial.dbms.default_primaries_count`	The number of initial servers in primary mode. If not specified, it defaults to one server in primary mode.
`initial.dbms.default_secondaries_count`	The number of initial servers in secondary mode. If not specified, it defaults to zero servers in secondary mode.

The following example shows how to set up a basic cluster with three servers with primary hosting capabilities.

Configuring any listen address to be something other than `localhost`, `127.0.0.1`, or another loopback address, will expose the Neo4j process to connections from outside of the server that it is running on.

Make sure you understand the security implications and strongly consider setting up encryption.

Configure a cluster with three servers

The following example shows how to set up a basic cluster with three members hosting the default database, `neo4j` (in addition to the `system` database), in primary mode.

Example 55. Configure a cluster with three servers in primary mode

In this example, three servers named `server01.example.com`, `server02.example.com` and `server03.example.com` are configured. Neo4j Enterprise Edition is installed on all three servers. They are configured by preparing neo4j.conf on each server. Note that they are all identical, except for the configuration of `server.default_advertised_address`:

neo4j.conf on server01.example.com:

```
server.default_listen_address=0.0.0.0
server.default_advertised_address=server01.example.com
dbms.cluster.discovery.endpoints=server01.example.com:5000,server02.example.com:5000,server03.example
.com:5000
initial.dbms.default_primaries_count=3
```

neo4j.conf on server02.example.com:

```
server.default_listen_address=0.0.0.0
server.default_advertised_address=server02.example.com
dbms.cluster.discovery.endpoints=server01.example.com:5000,server02.example.com:5000,server03.example
.com:5000
initial.dbms.default_primaries_count=3
```

neo4j.conf on server03.example.com:

```
server.default_listen_address=0.0.0.0
server.default_advertised_address=server03.example.com
dbms.cluster.discovery.endpoints=server01.example.com:5000,server02.example.com:5000,server03.example
.com:5000
initial.dbms.default_primaries_count=3
```

The Neo4j servers are ready to be started. The startup order does not matter.

After the cluster has started, it is possible to connect to any of the instances and run `SHOW SERVERS` to check the status of the cluster. This shows information about each member of the cluster:

```
SHOW SERVERS;
```

```
+----------------------------------------------------------------------------------------------------
-------+
| name                                   | address           | state     | health      | hosting
|
+----------------------------------------------------------------------------------------------------
-------+
| "d6fbe54b-0c6a-4959-9bcb-dcbbe80262a4" | "server001:7687" | "Enabled" | "Available" | ["system",
"neo4j"] |
| "e56b49ea-243f-11ed-861d-0242ac120002" | "server002:7687" | "Enabled" | "Available" | ["system",
"neo4j"] |
| "73e9a990-0a97-4a09-91e9-622bf0b239a4" | "server003:7687" | "Enabled" | "Available" | ["system",
"neo4j"] |
+----------------------------------------------------------------------------------------------------
-------+
```

For more extensive information about each server, use the `SHOW SERVERS YIELD *` command:

```
SHOW SERVERS YIELD *;
```

```
+------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------
---------------------------------------------+
| serverId                               | name                                   | address
| state      | health       | hosting             | requestedHosting     | tags | allowedDatabases |
deniedDatabases | modeConstraint | version     |
+------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------
---------------------------------------------+
| "d6fbe54b-0c6a-4959-9bcb-dcbbe80262a4" | "d6fbe54b-0c6a-4959-9bcb-dcbbe80262a4" | "server001:7687"
| "Enabled" | "Available" | ["system", "neo4j"] | ["system", "neo4j"] | []   | []                | []
| "NONE"         | "5.0.0"     |
| "e56b49ea-243f-11ed-861d-0242ac120002" | "e56b49ea-243f-11ed-861d-0242ac120002" | "server002:7687"
| "Enabled" | "Available" | ["system", "neo4j"] | ["system", "neo4j"] | []   | []                | []
| "NONE"         | "5.0.0"     |
| "73e9a990-0a97-4a09-91e9-622bf0b239a4" | "73e9a990-0a97-4a09-91e9-622bf0b239a4" | "server003:7687"
| "Enabled" | "Available" | ["system", "neo4j"] | ["system", "neo4j"] | []   | []                | []
| "NONE"         | "5.0.0"     |
+------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------
---------------------------------------------+
```

Startup time

The instance may appear unavailable while it is joining the cluster. If you want to follow along with the startup, you can see the messages in neo4j.log.

Create new databases in a cluster

As mentioned in the Introduction, a server in a cluster can either host a database in primary or secondary mode. For transactional workloads, a cluster with predominantly primaries is preferred for fault tolerance and automatic failover.

The cluster can preferably have more secondaries if the workload is more analytical. Such configuration is optimized for scalability but it is not fault-tolerant and does not provide automatic failover. Both scenarios are covered in the following examples.

Example 56. Create a new database with three primaries

In the `system` database on one of the servers from the previous example, execute the following Cypher command to create a new database:

```
CREATE DATABASE foo
TOPOLOGY 3 PRIMARIES
```

If `TOPOLOGY` is not specified, the database is created according to `initial.dbms.default_primaries_count` specified in `neo4j.conf`. Also, if `initial.dbms.default_secondaries_count` is specified to any other number than 0, the second line of the command would read `TOPOLOGY 3 PRIMARIES 0 SECONDARIES`. Thus the number specified with `TOPOLOGY` overrides both `initial.dbms.default_primaries_count` and `initial.dbms.default_secondaries_count` (if applicable) provided that the specified numbers do not exceed the number of available servers.

Example 57. Create a new database with one primary and two secondaries

In the system database on one of the servers from the previous example, execute the following Cypher command to create a new database:

```
CREATE DATABASE bar
TOPOLOGY 1 PRIMARY 2 SECONDARIES
```

Note that this operation is possible even without specifying initial.dbms.default_secondaries_count in the initial configuration. Anything specified in the TOPOLOGY part of the Cypher command overrides the initial.dbms.default_secondaries_count setting.

Cluster server discovery

In order to form or connect to a running cluster, any new member needs to know the addresses of at least some of the other servers in the cluster. This information is used to bind to the servers in order to run the discovery protocol and get the full information about the cluster. The best way to do this 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 servers are replaced and the new members have different addresses, the list becomes outdated. An outdated list can be avoided by ensuring that the new members can be reached via the same address as the old members, but this is not always practical.
- Under some circumstances the addresses are unknown when configuring the cluster. This can be the case, for example, when using container orchestration to deploy a cluster.

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 continuously exchanges information about changes to the topology. The behavior of the initial discovery is determined by the parameters dbms.cluster.discovery.type and dbms.cluster.discovery.endpoints, 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. Use the default dbms.cluster.discovery.type=LIST and hard code the addresses in the configuration of each machine. This alternative is illustrated by Configure a cluster with three servers example.

Discovery using DNS with multiple records

When using initial discovery with DNS, a DNS record lookup is performed when a server starts up. Once a server has joined a cluster, further membership changes are communicated amongst the servers in the cluster as part of the discovery service.

The following DNS-based mechanisms can be used to get the addresses of other servers in the cluster for discovery:

`dbms.cluster.discovery.type=DNS`

With this configuration, the initial discovery members are resolved from *DNS A* records to find the IP addresses to contact. The value of `dbms.cluster.discovery.endpoints` should be set to a single domain name and the port of the discovery service. For example: `dbms.cluster.discovery.endpoints=cluster01.example.com:5000`. The domain name should return an A record for every server in the cluster when a DNS lookup is performed. Each A record returned by DNS should contain the IP address of the server in the cluster. The configured server uses all the IP addresses from the A records to join or form a cluster.

> The discovery port must be the same on all servers when using this configuration. If this is not possible, consider using the discovery type `SRV` instead.

`dbms.cluster.discovery.type=SRV`

With this configuration, the initial discovery members are resolved from *DNS SRV* records to find the IP addresses/hostnames and discovery service ports to contact. The value of `dbms.cluster.discovery.endpoints` should be set to a single domain name and the port set to `0`. For example: `dbms.cluster.discovery.endpoints=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 servers to be discovered. The configured server uses all the addresses from the SRV record to join or form a cluster.

Discovery in Kubernetes

A special case is when a cluster is running in Kubernetes and each server is running as a Kubernetes service. Then, the addresses of the other servers can be obtained using the List Service API, as described in the Kubernetes API documentation.

The following settings are used to configure for this scenario:

- Set `dbms.cluster.discovery.type=K8S`.
- Set `dbms.kubernetes.label_selector` to the label selector for the cluster services. For more information, see the Kubernetes official documentation.
- Set `dbms.kubernetes.service_port_name` to the name of the service port used in the Kubernetes service definition for the Core's discovery port. For more information, see the Kubernetes official documentation.

With this configuration, `dbms.cluster.discovery.endpoints` is not used and any value assigned to it is ignored.

- The pod running Neo4j must use a service account that has permission to list services. For further information, see the Kubernetes documentation on RBAC authorization or ABAC authorization.
- The configured `server.discovery.advertised_address` must exactly match the Kubernetes-internal DNS name, which is of the form `<service-name>.<namespace>.svc.cluster.local`.

As with DNS-based methods, the Kubernetes record lookup is only performed at startup.

Leadership, routing and load balancing

Elections and leadership

The servers in a cluster use the Raft protocol to ensure consistency and safety. An implementation detail of Raft is that it uses a *Leader* role to impose an ordering on an underlying log with other instances acting as *Followers* which replicate the leader's state. Specifically in Neo4j, this means that writes to the database are ordered by the server currently playing the *Leader* role for the respective database.

Only servers hosting a database in primary mode can be elected leaders for that database, provided that the cluster contains more than one primary server. If a Neo4j DBMS cluster contains multiple databases, each one of those databases operates within a logically separate Raft group, and therefore each has an individual leader. This means that a server may act both as *Leader* for some databases, and as *Follower* for other databases.

If a follower has not heard from the leader for a while, then it can initiate an election and attempt to become the new leader. The follower makes itself a *Candidate* and asks other servers to vote for it. If it can get a majority of the votes, then it assumes the leader role. Servers do not vote for a candidate which is less up-to-date than itself. There can only be one leader at any time per database, and that leader is guaranteed to have the most up-to-date log.

Elections are expected to occur during the normal running of a cluster and they do not pose an issue in and of itself. If you are experiencing frequent re-elections and they are disturbing the operation of the cluster then you should try to figure out what is causing them. Some common causes are environmental issues (e.g. a flaky networking) and work overload conditions (e.g. more concurrent queries and transactions than the hardware can handle).

Leadership balancing

Write transactions are always routed to the leader for the respective database. As a result, unevenly distributed leaderships may cause write queries to be disproportionately directed to a subset of servers. By default, Neo4j avoids this by automatically transferring database leaderships so that they are evenly distributed throughout the cluster. Additionally, Neo4j automatically transfers database leaderships away from instances where those databases are configured to be read-only using server.databases.read_only or similar.

Server-side routing

Server-side routing is a complement to the client-side routing, performed by a Neo4j Driver.

In a cluster deployment of Neo4j, Cypher queries may be directed to a cluster member that is unable to run the given queries. With server-side routing enabled, such queries are rerouted internally to a cluster member that is expected to be able to run them. This situation can occur for write-transaction queries when they address a database for which the receiving cluster member is not the leader.

The cluster role for cluster members is per database. Thus, if a write-transaction query is sent to a cluster member that is not the leader for the specified database (specified either via the Bolt Protocol or by the Cypher syntax: `USE` clause), server-side routing is performed if properly configured.

Server-side routing is enabled by the DBMS, by setting `dbms.routing.enabled=true` for each cluster member. The listen address (`server.routing.listen_address`) and advertised address (`server.routing.advertised_address`) also need to be configured for server-side routing communication.

Client connections need to state that server-side routing should be used and this is available for Neo4j Drivers and HTTP API.

Neo4j Drivers can only use server-side routing when the `neo4j://` URI scheme is used. The Drivers do not perform any routing when the `bolt://` URI scheme is used, instead connecting directly to the specified host.

On the cluster-side you must fulfill the following prerequisites to make server-side routing available:

- Set `dbms.routing.enabled=true` on each member of the cluster.
- Configure `server.routing.listen_address`, and provide the advertised address using `server.routing.advertised_address` on each member.
- Optionally, you can set `dbms.routing.default_router=SERVER` on each member of the cluster.

The last prerequisite enforces server-side routing on the clients by sending out a routing table with exactly one entry to the client. Therefore, `dbms.routing.default_router=SERVER` configures a cluster member to make its routing table behave like a standalone instance. The implication is that if a Neo4j Driver connects to this cluster member, then the Neo4j Driver sends all requests to that cluster member. Please note that the default configuration for `dbms.routing.default_router` is `dbms.routing.default_router=CLIENT`. See `dbms.routing.default_router` for more information.

The HTTP-API of each member benefits from these settings automatically.

Server-side routing connector configuration

Rerouted queries are communicated over the Bolt Protocol using a designated communication channel. The receiving end of the communication is configured using the following settings:

- `dbms.routing.enabled`

- `server.routing.listen_address`
- `server.routing.advertised_address`

Server-side routing driver configuration

Server-side routing uses the Neo4j Java driver to connect to other cluster members. This driver is configured with settings of the format:

- `dbms.routing.driver.*`

> 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.

Intra-cluster encryption

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

Introduction

The security solution for cluster communication is based on standard SSL/TLS technology (referred to jointly as SSL). Encryption is just one aspect of security, the other cornerstones are authentication and integrity. A secure solution is based on a key infrastructure which is deployed together with a requirement of authentication.

The SSL support in the platform is documented in detail in SSL framework. This section covers the specifics as they relate to securing a cluster.

Under SSL, an endpoint can authenticate itself using certificates managed by a Public Key Infrastructure (PKI).

> The deployment of a secure key management infrastructure is beyond the scope of this manual, and should be entrusted to experienced security professionals. The example deployment illustrated below is for reference purposes only.

Example deployment

Generate and install cryptographic objects

The generation of cryptographic objects is for the most part outside the scope of this manual. It generally requires having a PKI with a Certificate Authority (CA) within the organization and they should be able to advise here. Note that the information in this manual relating to the PKI is mainly for illustrative purposes.

When the certificates and private keys are obtained they can be installed on each of the servers. Each server has a certificate of its own, signed by a CA, and the corresponding private key. The certificate of the CA is installed into the `trusted` directory, and any certificate signed by the CA is thus trusted. This means that the server now has the capability of establishing trust with other servers.

🔥	Be sure to exercise caution when using CA certificates in the `trusted` directory, as any certificates signed by that CA are then trusted to join the cluster. 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 a mutual authentication setup is deployed, which means that both ends of a channel have to authenticate. To enable mutual authentication the SSL policy must have `client_auth` set to `REQUIRE` (which is the default). Servers are by default required to authenticate themselves, so there is no corresponding server setting.

If the certificate for a particular server is compromised, it is possible to revoke it by installing a Certificate Revocation List (CRL) in the `revoked` directory. It is also possible to redeploy using a new CA. For contingency purposes, it is advised to have a separate intermediate CA specifically for the cluster which can be substituted in its entirety should it ever become necessary. This approach would be much easier than having to handle revocations and ensuring their propagation.

Example 58. Generate and install cryptographic objects

In this example, assume that the private key and certificate file are named *private.key* and *public.crt*, respectively. The policy configuration for the key and certificate names/locations can be overridden if different names are desired. For this server, use the default configuration, create the appropriate directory structure, and install the certificate:

```
$neo4j-home> mkdir certificates/cluster
$neo4j-home> mkdir certificates/cluster/trusted
$neo4j-home> mkdir certificates/cluster/revoked

$neo4j-home> cp $some-dir/private.key certificates/cluster
$neo4j-home> cp $some-dir/public.crt certificates/cluster
```

Configure the cluster SSL policy

By default, cluster communication is unencrypted. To configure a cluster to encrypt its intra-cluster communication, set `dbms.ssl.policy.cluster.enabled` to `true`.

An SSL policy utilizes the installed cryptographic objects and additionally allows parameters to be configured. Use the following parameters in the configuration:

Table 36. 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`	A particular single strong cipher can be enforced and thus remove any doubt about which cipher gets negotiated and chosen. The selected cipher offers Perfect Forward Secrecy (PFS) 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`	With control of the entire cluster, the latest TLS standard can be enforced without any concern for backwards compatibility. It has no known security vulnerabilities and uses the most modern algorithms for key exchanges, etc.

In the following example, an SSL policy is created and configured that is used in the example cluster.

Example 59. Configure the cluster SSL policy

This example assumes that the directory structure is created, and certificate files are installed, as per the previous example.

Add the following content to the *neo4j.conf* file:

```
dbms.ssl.policy.cluster.enabled=true
dbms.ssl.policy.cluster.tls_versions=TLSv1.2
dbms.ssl.policy.cluster.ciphers=TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
dbms.ssl.policy.cluster.client_auth=REQUIRE
```

Any user data communicated between instances is now secured. Note that an instance that is not correctly setup is not able to communicate with the others.

The policy must be configured on every server with the same settings. The actual cryptographic objects installed are mostly different since they do not share the same private keys and corresponding certificates. However, the trusted CA certificate is shared.

Validate the secure operation of the cluster

To make sure that everything is secured as intended, it makes sense to validate using external tooling such as, for example, the open source assessment tools `nmap` or `OpenSSL`.

Example 60. Validate the secure operation of the cluster

This example uses the `nmap` tool to validate the secure operation of the cluster. A simple test to perform is a cipher enumeration using the following command:

```
nmap --script ssl-enum-ciphers -p <port> <hostname>
```

The hostname and port have to be adjusted according to the example configuration. This can prove that TLS is in fact enabled and that only the intended cipher suites are enabled. All servers and all applicable ports should be tested.

For testing purposes, it is also possible to utilize a separate testing instance of Neo4j which, for example, has an untrusted certificate in place. The expected result of this is that the test server is not able to participate in replication of user data. The debug logs generally indicate an issue by printing an SSL or certificate-related exception.

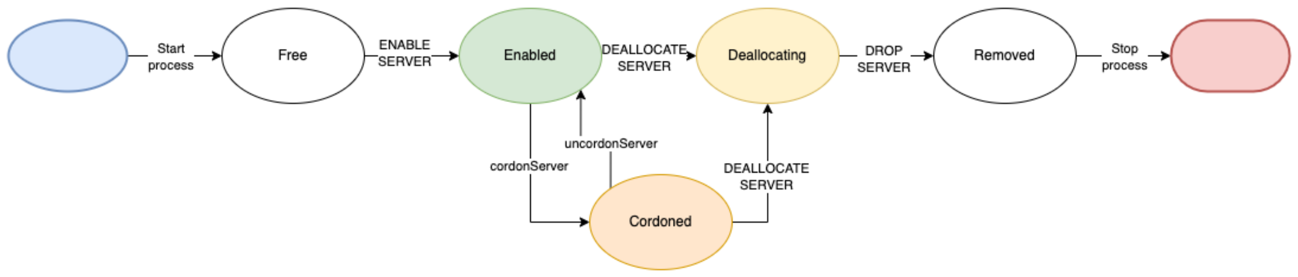
Managing servers in a cluster

As described previously, server-management is completely separate from database-management in a clustered environment. This section describes how to work with servers in a cluster: adding and removing them, as well as altering their metadata.

Server states

A server can exist in five different states within the DBMS:

- Free
- Enabled
- Deallocating
- Cordoned
- Dropped



Free state

When a server is discovered by the discovery service (see Cluster server discovery for more information), it is created in the Free state. Servers in this state have a unique automatically generated ID, but are otherwise unconfigured. These free servers are not yet part of the cluster and cannot be allocated to host any databases.

When first discovered, a server's name defaults to the value of its generated server ID.

Enabled state

A server in the free state needs to be explicitly enabled in order to be considered an active member of the cluster. The command `ENABLE SERVER server name` is used to transition a server into this *Enabled* state. Subsequently, the server may be allocated databases to host.

Deallocating state

When a server is no longer needed, it cannot be removed from the cluster while it is still allocated to host any databases. The command `DEALLOCATE DATABASE[S] FROM SERVER[S] server[,…]` is used to transition servers to the *Deallocating* state, reallocating all their hosted databases to other servers in the cluster. Additionally, servers which are deallocating will not have any further databases allocated to them.

Cordoned state

The *Cordoned* state is similar to *Deallocating* in that servers in this state will not be allocated to host additional databases. Unlike *Deallocating* however, cordoned servers do not lose the databases they already host.

A server is transitioned from the *Enabled* state to the *Cordoned* state by executing the procedure `dbms.cluster.cordonServer`. A server in the *Cordoned* state may be transitioned to *Deallocating*, or back to *Enabled*.

This state is primarily used for error handling.

Dropped state

Once a server is in state *Deallocating* and is only hosting the system database, it is safe to drop it. The command `DROP SERVER server name` logically removes the server from the cluster. However, as long as the server's Neo4j process is running, it is still visible to the other cluster members in the *Dropped* state. Once the Neo4j process is stopped, the server finally disappears. Once dropped, a server cannot rejoin a cluster.

The same physical hardware can rejoin the cluster, provided the Neo4j installation has been "reset" (either re-installing, or running `neo4j-admin unbind`), causing it to receive a new generated server ID on next startup.

`SHOW SERVERS`

The Cypher command `SHOW SERVERS` displays all current servers running in the cluster, including both servers yet to be enabled (i.e. servers in the *Free* state) in the DBMS as well as dropped servers.

```
neo4j@neo4j> SHOW SERVERS;
+------------------------------------------------------------------------------------------------------------
---------+
| name                                   | address          | state     | health      | hosting
|
+------------------------------------------------------------------------------------------------------------
---------+
| "135ad202-5405-4d3c-9822-df39f59b823c" | "localhost:7690" | "Dropped" | "Available" | ["system"]
|
| "25a7efc7-d063-44b8-bdee-f23357f89f01" | "localhost:7689" | "Enabled" | "Available" | ["system", "foo",
"neo4j"] |
| "42a97acc-acf6-40c0-aff2-3993e90db1ff" | "localhost:7691" | "Free"    | "Available" | ["system"]
|
| "782f0ee2-5474-4250-b905-4cd8b8f586ba" | "localhost:7688" | "Enabled" | "Available" | ["system", "foo",
"neo4j"] |
| "8512c9b9-d9e8-48e6-b037-b15b0004ca18" | "localhost:7687" | "Enabled" | "Available" | ["system", "foo",
"neo4j"] |
+------------------------------------------------------------------------------------------------------------
---------+
```

To display all available information about the servers in the cluster, use `SHOW SERVERS YIELD *`:

```
neo4j@neo4j> SHOW SERVERS YIELD *;
+------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------+
| serverId                               | name                                   | address          |
httpAddress      | httpsAddress | state          | health      | hosting                           |
requestedHosting           | tags | allowedDatabases | deniedDatabases | modeConstraint | version
|
+------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------+
| "135ad202-5405-4d3c-9822-df39f59b823c" | "135ad202-5405-4d3c-9822-df39f59b823c" | "localhost:7690" |
"localhost:7477" | NULL         | "Deallocating" | "Available" | ["system"]                    | ["system"]
| []   | []               | []               | "NONE"          | "5.0.0-drop09.0" |
| "25a7efc7-d063-44b8-bdee-f23357f89f01" | "25a7efc7-d063-44b8-bdee-f23357f89f01" | "localhost:7689" |
"localhost:7476" | NULL         | "Enabled"      | "Available" | ["system", "foo", "neo4j"] | ["system",
"foo", "neo4j"] | []   | []               | []               | "NONE"          | "5.0.0-drop09.0" |
| "42a97acc-acf6-40c0-aff2-3993e90db1ff" | "42a97acc-acf6-40c0-aff2-3993e90db1ff" | "localhost:7691" |
"localhost:7478" | NULL         | "Free"         | "Available" | ["system"]                    | []
| []   | []               | []               | "NONE"          | "5.0.0-drop09.0" |
| "782f0ee2-5474-4250-b905-4cd8b8f586ba" | "782f0ee2-5474-4250-b905-4cd8b8f586ba" | "localhost:7688" |
"localhost:7475" | NULL         | "Enabled"      | "Available" | ["system", "foo", "neo4j"] | ["system",
"foo", "neo4j"] | []   | []               | []               | "NONE"          | "5.0.0-drop09.0" |
| "8512c9b9-d9e8-48e6-b037-b15b0004ca18" | "8512c9b9-d9e8-48e6-b037-b15b0004ca18" | "localhost:7687" |
"localhost:7474" | NULL         | "Enabled"      | "Available" | ["system", "foo", "neo4j"] | ["system",
"foo", "neo4j"] | []   | []               | []               | "NONE"          | "5.0.0-drop09.0" |
+------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------+
```

Add a server to the cluster

To add a server to a running cluster (see Deploy a basic cluster for more information on how to set up a basic cluster), configure it to discover other existing cluster members. There are several different ways to do this, see Cluster server discovery. Once the new server is configured to discover the cluster's members, it can be started.

Once started, the new server appears in the output of `SHOW SERVERS` with the *Free* state. Copy the server's name from `SHOW SERVERS` and enable it:

```
neo4j@neo4j> ENABLE SERVER '42a97acc-acf6-40c0-aff2-3993e90db1ff';
```

The ENABLE command can take several options:

```
neo4j@neo4j> ENABLE SERVER '25a7efc7-d063-44b8-bdee-f23357f89f01' OPTIONS
    {modeConstraint:'PRIMARY', allowedDatabases:['foo']};
```

modeConstraint is used to control whether a server can be used to host a database in only primary or secondary mode. allowedDatabases and deniedDatabases are collections of database names that filter which databases may be hosted on a server. The allowedDatabases and deniedDatabases are mutually exclusive and if both are specified, an error is returned.

allowedDatabases and deniedDatabases do not affect composite databases, they are always available everywhere.

If no options are set, a server can host any database in any mode. Servers can also provide default values for these options via their *neo4j.conf* files on first startup.

```
initial.server.mode_constraint='PRIMARY'
initial.server.allowed_databases='foo'
initial.server.denied_databases='bar','baz'
```

If conflicting options are provided between *neo4j.conf* and the ENABLE SERVER command, those provided to ENABLE SERVER are used.

Hosting databases on added servers

Once enabled, a server does not automatically host databases unless:

- New databases are created.
- Existing database topologies are altered to request more hosts.
- Another server is transitioned to the *Deallocating* state.
- You explicitly rebalance the databases across the cluster.

The command REALLOCATE DATABASE[S] can be used to rebalance database allocations across the cluster, adding some to the newly added server(s). This command can be used with DRYRUN to get a view of how the databases would be rebalanced.

DRYRUN is introduced in Neo4j 5.2 and thus does not work in previous versions.

```
neo4j@neo4j> DRYRUN REALLOCATE DATABASES;
+-----------------------------------------------------------------------------------------------------------
------------------------------+
| database | fromServerName | fromServerId                               | toServerName | toServerId
| mode      |
+-----------------------------------------------------------------------------------------------------------
------------------------------+
| "bar"    | "server-1"     | "00000000-27e1-402b-be79-d28047a9418a" | "server-5"    | "00000003-b76c-483f-
b2ca-935a1a28f3db" | "primary" |
| "bar"    | "server-3"     | "00000001-7a21-4780-bb83-cee4726cb318" | "server-4"    | "00000002-14b5-4d4c-
ae62-56845797661a" | "primary" |
+-----------------------------------------------------------------------------------------------------------
------------------------------+
```

Removing a server from the cluster

Removing a server from the cluster requires two steps: deallocating, then dropping.

Deallocating databases from a server

Remember that before removing a server from an existing cluster, any databases allocated to it must be moved (see Deallocating state), using the `DEALLOCATE DATABASES` command:

```
neo4j@neo4j> DRYRUN DEALLOCATE DATABASES FROM SERVER '135ad202-5405-4d3c-9822-df39f59b823c';
```

When deallocating databases from servers, it is important to be mindful of the topology for each database to ensure that there are sufficient servers left in the cluster to satisfy the topologies of each database. Attempting to deallocate database(s) from a server that would result in less available servers than required fails with an error and no changes are made.

For example, if the cluster contains 5 servers and a database `foo` has a topology requiring 3 primaries and 2 secondaries, then it is *not* possible to deallocate any of the original 5 servers, without first enabling a 6th, or altering the desired topology of `foo` to require fewer servers overall.

The command can be used with `DRYRUN` to get a view of how the databases would be moved from the deallocated server(s).

```
neo4j@neo4j> DRYRUN DEALLOCATE DATABASES FROM SERVER '135ad202-5405-4d3c-9822-df39f59b823c';
+-----------------------------------------------------------------------------------------------------------
---------------------------------+
| database | fromServerName | fromServerId                           | toServerName | toServerId
| mode        |
+-----------------------------------------------------------------------------------------------------------
---------------------------------+
| "db1"    | "server-3"     | "135ad202-5405-4d3c-9822-df39f59b823c" | "server-5"   | "00000003-b30a-434e-
b9bf-1a5c8009773a" | "secondary" |
+-----------------------------------------------------------------------------------------------------------
---------------------------------+
```

Dropping a server

Once `DEALLOCATE DATABASES` is executed for a server, its databases begin being moved. It is important not to attempt the next step before `SHOW SERVERS` reports that the deallocating server no longer hosts any databases besides `system`.

For example, do not drop the server `135ad202-5405-4d3c-9822-df39f59b823c` given the following output:

```
neo4j@neo4j> SHOW SERVERS;
+-----------------------------------------------------------------------------------------------------------
---------+
| name                                   | address          | state           | health        | hosting
|
+-----------------------------------------------------------------------------------------------------------
---------+
| "135ad202-5405-4d3c-9822-df39f59b823c" | "localhost:7690" | "Deallocating" | "Available" | ["system",
"foo"]     |
+-----------------------------------------------------------------------------------------------------------
---------+
```

The deallocation process may take some time, as `foo` must be successfully copied and started on a new server before it is stopped on `135ad202-5405-4d3c-9822-df39f59b823c` in order to preserve the availability and fault tolerance of `foo`.

Once `SHOW SERVERS` reflects that the server no longer hosts `foo`, the server may be dropped:

```
neo4j@neo4j> DROP SERVER '135ad202-5405-4d3c-9822-df39f59b823c';
```

Once this command has been executed successfully, the neo4j process on the server in question may be stopped.

Controlling a server's metadata

Altering server options

A running server can have its options modified using the `ALTER SERVER` command. For example, to prevent a server from hosting databases in `PRIMARY`, execute the following:

```
neo4j@neo4j> ALTER SERVER '25a7efc7-d063-44b8-bdee-f23357f89f01' SET OPTIONS {modeConstraint:'SECONDARY'};
```

Altering servers may cause databases to be moved, and should be performed with care. For example, if the server `25a7efc7-d063-44b8-bdee-f23357f89f01` hosts database `foo` in primary mode when the above command is executed, then another server must begin hosting `foo` in primary mode. Likewise, if `ALTER SERVER '25a7efc7-d063-44b8-bdee-f23357f89f01' SET OPTIONS {allowedDatabases:['bar','baz']};` is executed, then `foo` is forced to move.

`allowedDatabases` and `deniedDatabases` do not affect composite databases, they are always available everywhere.

As with the `DEALLOCATE DATABASES FROM SERVER …` command, if the alteration of a server's options renders it impossible for the cluster to satisfy one or more of the databases' topologies, then the command fails and no changes are made.

Input provided to `SET OPTIONS {…}` replaces **all** existing options, rather than being combined with them. For instance if `SET OPTIONS {modeConstraint:'SECONDARY'}` is executed followed by `SET OPTIONS {allowedDatabases:['foo']}`, the execution of the second `ALTER` removes the mode constraint.

Renaming a server

When first discovered and enabled, a server's name defaults to the value of its generated server ID. However, this can be changed later using the following command:

```
neo4j@neo4j> RENAME SERVER '25a7efc7-d063-44b8-bdee-f23357f89f01' TO 'eu-server-4';
```

This only affects the name of the server; the ID of the server remains fixed as `25a7efc7-d063-44b8-bdee-`

f23357f89f01.

Error handling

Occasionally, servers in a cluster may suffer issues such as network partitions or process crashes. These easiest way to observe these server failures is by executing `SHOW SERVERS` and checking for `'Unavailable'` in the `health` column.

An `Available` health status does not indicate that a server is functioning perfectly, only that other servers in the cluster are able to make contact with it. For more in depth monitoring of cluster and server health, see Monitoring clusters.

If the issue with the `Unavailable` server proves permanent, then the server should be removed. However, if the issue is temporary then it likely is not desirable to remove these servers entirely as this causes all their hosted databases to be moved. Instead it is preferable to prevent those servers from being allocated any new databases to host, either as a result of databases being created or moved.

This is known as *cordoning* the server in question, and can be achieved by executing the following procedure against the `system` database:

```
neo4j@neo4j> CALL dbms.cluster.cordonServer('25a7efc7-d063-44b8-bdee-f23357f89f01');
```

`SHOW SERVERS` should then reflect that the server in question is now in *Cordoned* state.

Once the issue with the server has been resolved, the server can be returned to its previous *Enabled* state as follows:

```
neo4j@neo4j> CALL dbms.cluster.uncordonServer('25a7efc7-d063-44b8-bdee-f23357f89f01');
```

An unavailable server which has not been cordoned may still be allocated to host new databases. When the server recovers it observes that it is due to host these databases and begin catching up from some other available server (if one exists). However, in the meantime those databases have reduced fault tolerance or, worse, reduced availability. See Disaster Recovery for more details.

Managing databases in a cluster

The number of both primary and secondary servers to host a database can be set when the database is created and altered after creation. The command `CREATE DATABASE` can be used to specify the initial topology and `ALTER DATABASE` can be used to change the topology once the database is created. If a database is no longer needed, the command `DROP DATABASE` deletes the database from the cluster.

`CREATE DATABASE`

The command to create a database in a cluster is not significantly different from the command to create a database in a non-clustered environment (see Cypher Manual → Database management for more information on database management on single servers). The difference in a clustered environment is that the topology can be specified, i.e. how many primaries and secondaries are desired for the database. To create a database `foo` with 3 servers hosting the database in primary mode and 2 servers in secondary mode, the command looks like this:

```
CREATE DATABASE foo TOPOLOGY 3 PRIMARIES 2 SECONDARIES
```

The command can only be executed successfully if the cluster's servers are able to satisfy the specified topology. If they are not, the command results in an error. For example, if the cluster's servers are set up with mode constraints to contain two primaries and three secondaries, or if only four servers exist, the command fails with an error.

`ALTER DATABASE`

To alter the topology of or read/write access to a database after it has been created, use the command `ALTER DATABASE`.

Alter topology

To change the topology of the database `foo` from the previous example, the command can look like this:

```
ALTER DATABASE foo SET TOPOLOGY 2 PRIMARIES 1 SECONDARY
```

Like the `CREATE DATABASE` command, this command results in an error if the cluster does not contain sufficient servers to satisfy the requested topology.

When there is more than one possible permutation of the specified topology, Neo4j uses an allocator to decide how to spread the database across the cluster. This normally happens when the cluster is configured with more servers than the sum of the number of primaries and secondaries for any one database.

It is not possible to automatically transition to or from a topology with a single primary host. This limitation is due to Raft replication not being present when only a single server is running, i.e. the server is running in *standalone* or *single mode*. Attempting to do so results in an error.

However, it is possible to *manually* do this transition. The first step is to back up the database, see Backup and restore for more information. Once the database is backed up, the next step is to drop the database, see Administrative commands for more information. The last step is to either seed a cluster from the backup with the new topology, or to restore the backup on a single server. See Seed a cluster further on for information on seeding.

Alter access

To alter the access to the database `foo`, the syntax looks like this:

```
ALTER DATABASE foo SET ACCESS {READ ONLY | READ WRITE}
```

By default, a newly created database has both read and write access.

Seed a cluster

There are two different ways to seed a cluster with data. The first option is to use a *designated seeder*, where a designated server is used to create a backed-up database on other servers in the cluster. The other options is to seed the cluster from URI, where all servers to host a database are seeded with an identical seed from an external source specified by the URI. Keep in mind that using a designated seeder can be problematic in some situations as it is not known in advance how a database is going to be allocated to the servers in a cluster. Also, this method relies on the seed already existing on one of the servers.

Designated seeder

In order to designate a server in the cluster as a seeder, a database backup is transferred to that server using the `neo4j-admin database restore` command. Subsequently, that server is used as the source for other cluster members to copy the backed-up database from.

This example creates a user database called `foo`, hosted on three servers in primary mode. The `foo` database **should not** previously exist on any of the servers in the cluster.

If a database with the same name as your backup already exists, use the command `DROP DATABASE` to delete it and all users and roles associated with it.

1. Restore the `foo` database on one server. In this example, the `server01` member is used.

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

2. Find the server ID of `server01` by logging in to Cypher Shell and running `SHOW SERVERS`. Use any database to connect.

```
SHOW SERVERS;
```

```
+-----------------------------------------------------------------------------------------------------------
-------+
| name                                   | address          | state     | health      | hosting
|
+-----------------------------------------------------------------------------------------------------------
-------+
| "25a7efc7-d063-44b8-bdee-f23357f89f01" | "localhost:7689" | "Enabled" | "Available" | ["system",
"neo4j"] |
| "782f0ee2-5474-4250-b905-4cd8b8f586ba" | "localhost:7688" | "Enabled" | "Available" | ["system",
"neo4j"] |
| "8512c9b9-d9e8-48e6-b037-b15b0004ca18" | "localhost:7687" | "Enabled" | "Available" | ["system",
"neo4j"] |
+-----------------------------------------------------------------------------------------------------------
-------+
```

3. On one of the servers, use the `system` database and create the database `foo` using the server ID of `server01`.

```
CREATE DATABASE foo OPTIONS {existingData: 'use', existingDataSeedInstance: '8512c9b9-d9e8-48e6-b037-
b15b0004ca18'};
```

4. Verify that the `foo` database is online on the desired number of servers, in the desired roles. If the `foo` database is of considerable size, the execution of the command can take some time.

```
SHOW DATABASE foo;
```

```
+------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------+
| name  | type       | aliases | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage | default | home  | constituents |
+------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------+
| "foo" | "standard" | []      | "read-write" | "localhost:7687" | "primary" | FALSE  | "online"
| "online"      | ""            | FALSE   | FALSE | []           |
| "foo" | "standard" | []      | "read-write" | "localhost:7688" | "primary" | FALSE  | "online"
| "online"      | ""            | FALSE   | FALSE | []           |
| "foo" | "standard" | []      | "read-write" | "localhost:7689" | "primary" | TRUE   | "online"
| "online"      | ""            | FALSE   | FALSE | []           |
+------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------+

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

Seed from URI

This method seeds all servers with an identical seed from an external source, specified by the URI. The seed can be either a backup or a dump from an existing database.

The mechanism is pluggable, allowing new sources of seeds to be supported. These are called *seed providers*. The product has built-in support for seed from a mounted file system (file), FTP server, HTTP/HTTPS server and Amazon S3. The `URLConnectionSeedProvider` supports the following:

- file:
- ftp:
- http:
- https:
- URIs

Accordingly, the `S3SeedProviders` supports:

- S3:
- URIs

The URI of the seed is specified when the `CREATE DATABASE` command is issued:

```
CREATE DATABASE foo OPTIONS {existingData: 'use', seedURI:'s3://myBucket/myBackup.backup'}
```

Download and validation of the seed is only performed as the new database is started. If it fails, the database is not available and it has the `statusMessage`: `Unable to start database` of the `SHOW DATABASES` command.

```
neo4j@neo4j> SHOW DATABASES;
+-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
--+
| name    | type       | aliases | access       | address          | role      | writer | requestedStatus
| currentStatus | statusMessage                                              | default | home   |
constituents |
+-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
--+
| "seed3" | "standard" | []      | "read-write" | "localhost:7682" | "unknown" | FALSE  | "online"
| "offline"     | "Unable to start database `DatabaseId{3fe1a59b[seed3]}`" | FALSE   | FALSE | []
|
+-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
--+
```

To determine the cause of the problem, it is recommended to look at the `debug.log`.

Certain seed providers, such as S3, may require additional configuration. This is specified with the `seedConfig` option. This option expects a comma-separated list of configurations. Each configuration value is specified as a name followed by `=` and the value, as such:

```
CREATE DATABASE foo OPTIONS { existingData: 'use', seedURI: 's3:/myBucket/myBackup,backup', seedConfig:
'region=eu-west-1' }
```

Some seed providers may also want to pass credentials into the provider. These are specified with the `seedCredentials` option. Seed credentials are securely passed from the Cypher command to each server hosting the database. For this to work, Neo4j on each server in the cluster must be configured with identical keystores. This is identical to the configuration required by remote aliases, see Configuration of DBMS with remote database alias. If this configuration is not performed, the `seedCredential` option fails.

Seed provider reference

URL scheme	Seed provider	URI example
`file:`	`URLConnectionSeedProvider`	`file:/tmp/backup1.backup`
`ftp:`	`URLConnectionSeedProvider`	`ftp://myftp.com/backups/backup1.backup`
`http:`	`URLConnectionSeedProvider`	`http://myhttp.com/backups/backup1.backup`
`https:`	`URLConnectionSeedProvider`	`https://myhttp.com/backups/backup1.backup`
`S3:`	`S3SeedProvider`	`s3://mybucket/backups/backup1.backup`

Controlling locations with allowed/denied databases

A database can by default be allocated to run on any server in a cluster. However, it is possible to constrain the servers that specific databases are hosted on. This is done with `ENABLE SERVER` and `ALTER SERVER`, described in Managing servers in a cluster. The following options are available:

- `allowedDatabases` - a set of databases that are allowed to be hosted on a server.
- `deniedDatabases` - a set of databases that are denied to be hosted on a server. Allowed and denied are mutually exclusive.

- `modeConstraint` - controls in what mode (primary, secondary, or none) databases can be hosted on a server. If not set, there are no mode constraints on the server.

Change the default database

You can use the procedure `dbms.setDefaultDatabase("newDefaultDatabaseName")` to change the default database for a DBMS.

1. Ensure that the database to be set as default exists, otherwise create it using the command `CREATE DATABASE <database-name>`.
2. Show the name and status of the current default database by using the command `SHOW DEFAULT DATABASE`.
3. Stop the current default database using the command `STOP DATABASE <database-name>`.
4. Run `CALL dbms.setDefaultDatabase("newDefaultDatabaseName")` against the `system` database to set the new default database.
5. Optionally, you can start the previous default database as non-default by using `START DATABASE <database-name>`.

Monitor servers

To monitor the state of individual servers in a cluster, use the `SHOW SERVERS` command.

Listing Servers

Syntax:

```
SHOW SERVERS
```

Returns:

Name	Type	Description
`name`	String	The friendly name of the server, or its UUID if no name is set.
`address`	String	The address of the Bolt port for the server. May be `null`.
`state`	String	The state of the server in the topology.
`health`	String	The current availability of the server.

Name	Type	Description
`hosting`	List<String>	Databases that this server is currently running.

Possible values of `state`

- `Free` - server has been started, but not added to the cluster. It needs to be added with `ENABLE SERVER`.
- `Enabled` - server is part of cluster and can have database allocated to it.
- `Cordoned` - server may be hosting databases, but cannot have any more allocated to it.
- `Deallocating` - server is in the process of removing its databases. This may take some time, because it does not stop its copy of a database, if in primary mode, until another server has fully started its copy, to preserve the requested number of primaries.
- `Dropped` - server has been removed from the cluster, but the process has not exited.

Possible values of `health`

- `Available` - server has recently been in contact with the cluster member executing `SHOW SERVERS`.
- `Unavailable` - server has not had successful network communication with the cluster for a while.

`Unavailable` does not necessarily mean the server is not running, just that there are network problems connecting to it.

Example 61. Listing servers in a cluster

When running `SHOW SERVERS` against a cluster, expect similar output to the following:

```
+-----------------------------------------------------------------------------------------------------
--+
| name                                   | address          | state     | health      | hosting
|
+-----------------------------------------------------------------------------------------------------
--+
|"f4ae1895-26f1-4b93-bd31-
6f482be80d3d"|"localhost:7681"|"Enabled"|"Available"|["system","foo","neo4j"]|
|"ffa55a5b-2aca-45fc-be09-
2a894067025c"|"localhost:7682"|"Enabled"|"Available"|["system","foo","neo4j"]|
|"server3"                              |"localhost:7683"|"Enabled"|"Available"|["system","neo4j"]
|
+-----------------------------------------------------------------------------------------------------
--+
```

Listing more details of servers

If more details about the servers are needed, `SHOW SERVERS` can be appended with `YIELD *`.

Syntax:

```
SHOW SERVERS YIELD *
```

Returns:

Name	Type	Description
`serverId`	String	The UUID of the server
`name`	String	The friendly name of the server, or its UUID if no name is set.
`address`	String	The address of the Bolt port for the server. May be `null`.
`httpAddress`	String	The address of the HTTP port for the server. May be `null`.
`httpsAddress`	String	The address of the HTTPS port for the server. May be `null`.
`state`	String	The state of the server in the topology.
`health`	String	The current availability of the server.
`hosting`	List<String>	Databases that this server is currently running.
`requestedHosting`	List<String>	Databases that this server is supposed to be running. May be fewer databases if the server is in the process of safely stopping one, or more databases if the server is in the process of starting one up.
`tags`	List<String>	Tags applied to this server. Used for routing policies.

Name	Type	Description
`allowedDatabases`	List<String>	A list of the only databases that are allowed on this server. Empty means all are allowed.
`deniedDatabases`	List<String>	A list of databases that may not be hosted on this server. Empty means all are allowed.
`modeConstraint`	String	A limit on what modes (i.e. `primary` or `secondary`) a database can be in on this server.
`version`	String	The version of Neo4j this server is running.

Only one of `allowedDatabases` and `deniedDatabases` can be set as they are mutually exclusive.

Possible values of `modeConstraint`

- `NONE` - any modes can be allocated to this server.
- `PRIMARY` - only primary modes can be allocated to this server. These may be the target of writes for the database.
- `SECONDARY` - only secondary modes can be allocated to this server. These will never write to the database, only read.

Example 62. Listing more details about servers in a cluster

When running `SHOW SERVERS YIELD *` in a cluster, expect similar output to the following:

```
+-----------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+
| serverId                              | name                             | address        |
httpAddress    | httpsAddress | state    | health    | hosting               | requestedHosting
| tags | allowedDatabases| deniedDatabases| modeConstraint| version        |
+-----------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+
|"f4ae1895-26f1-4b93-bd31-6f482be80d3d"|"f4ae1895-26f1-4b93-bd31-
6f482be80d3d"|"localhost:7681"|"localhost:7471"|null
|"Enabled"|"Available"|["system","foo","neo4j"]|["system","foo","neo4j"]|[]    |[]               |[]
|"NONE"         |"5.0.0-SNAPSHOT"|
|"ffa55a5b-2aca-45fc-be09-2a894067025c"|"ffa55a5b-2aca-45fc-be09-
2a894067025c"|"localhost:7682"|"localhost:7472"|null
|"Enabled"|"Available"|["system","foo","neo4j"]|["system","foo","neo4j"]|[]    |[]               |[]
|"NONE"         |"5.0.0-SNAPSHOT"|
|"72bd3d0f-c1d1-4d39-9da7-015f5656e40b"|"server3"
|"localhost:7683"|"localhost:7473"|null          |"Enabled"|"Available"|["system","neo4j"]
|["system","neo4j"]      |[]    |[]               |[]               |"NONE"         |"5.0.0-SNAPSHOT"|
+-----------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+
```

Monitor databases

In addition to the system-wide metrics and logs described in previous sections, to monitor the state of individual databases hosted in a cluster, use the `SHOW DATABASES` command.

Listing Databases

Syntax:

```
SHOW DATABASES
```

Returns:

Name	Type	Description
`name`	String	The human-readable name of the database.
`type`	String	`standard`, `system` or `composite`.
`aliases`	List<String>	The names of any aliases the database may have.
`access`	String	The database access mode, either `read-write` or `read-only`.

Name	Type	Description
`address`	String	The bolt address of the server hosting the database.
`role`	String	The cluster role which the server fulfills for this database.
`writer`	Boolean	`true` for the database node that accepts writes. This node is either the leader for this database in a cluster or this is a standalone server.
`requestedStatus`	String	The state that an operator has requested the database to be in.
`currentStatus`	String	The state the database is actually in on this server.
`statusMessage`	String	A message explaining the current state of the database, which could be an error encountered by the Neo4j server when transitioning the database to `requestedStatus`, if any.
`default`	Boolean	Whether this database is the default for this DBMS.
`home`	Boolean	Whether this database is the home database for this user.
`constituents`	List<String>	A list of alias names making up this composite database, null for non-composite databases.

Note that for failed databases, `currentStatus` and `requestedStatus` are different. This can imply an error. For example:

- A database may take a while to transition from "offline" to "online", due to performing recovery.
- During normal operation, the `currentStatus` of a database may be transiently different from its `requestedStatus`, due to a necessary automatic process, such as one server copying store files from another.

The possible statuses are `initial`, `offline`, `store copying`, `deallocating`, `unknown`, `dirty`, and `quarantined`.

Additionally, note that databases hosted on servers that are offline are also returned by the `SHOW DATABASES` command. For such databases the `address` column displays `NULL`, the `currentStatus` column displays `unknown`, and the `statusMessage` displays `Server is unavailable`.

Example 63. Listing databases in standalone Neo4j

When executing `SHOW DATABASES` against a standalone server, the following output is expected:

```
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------+
| name     | type       | aliases | access       | address          | role       | writer | requestedStatus
| currentStatus | statusMessage | default | home | constituents |
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------+
|"neo4j"   |"standard" |[]       |"read-write"|"localhost:7687"| "primary" | true   | "online"
| "online"      | ""            |true      |true  |[]             |
|"system"  |"system"   |[]       |"read-write"|"localhost:7687"| "primary" | true   | "online"
| "online"      | ""            |false     |false |[]             |
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------+
```

Note that the `role`, `writer`, and `address` columns are primarily intended to distinguish between the states of a given database, across multiple servers deployed in a cluster. In a standalone deployment with a single server, the `address` field should be the same for every database, the `role` field should always be "primary", and the `writer` field should be true.

Example 64. Listing databases in a cluster

When running `SHOW DATABASES` against a cluster, expect similar output to the following:

```
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------+
| name     | type       | aliases | access     | address          | role      | writer | requestedStatus
| currentStatus | statusMessage | default | home | constituents |
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------+
|"neo4j"  |"standard" |[]       |"read-write"|"localhost:7681"|"primary"  |false   |"online"
|"online"       |""             |true      |true  |[]           |
|"neo4j"  |"standard" |[]       |"read-write"|"localhost:7682"|"primary"  |false   |"online"
|"online"       |""             |true      |true  |[]           |
|"neo4j"  |"standard" |[]       |"read-write"|"localhost:7683"|"primary"  |true    |"online"
|"online"       |""             |true      |true  |[]           |
|"neo4j"  |"standard" |[]       |"read-write"|"localhost:7684"|"secondary"|false   |"online"
|"online"       |""             |true      |true  |[]           |
|"system" |"system"   |[]       |"read-write"|"localhost:7681"|"primary"  |true    |"online"
|"online"       |""             |false     |false |[]           |
|"system" |"system"   |[]       |"read-write"|"localhost:7682"|"primary"  |false   |"online"
|"online"       |""             |false     |false |[]           |
|"system" |"system"   |[]       |"read-write"|"localhost:7683"|"primary"  |false   |"online"
|"online"       |""             |false     |false |[]           |
|"system" |"system"   |[]       |"read-write"|"localhost:7684"|"secondary"|false   |"online"
|"online"       |""             |false     |false |[]           |
|"foo"    |"standard" |[]       |"read-write"|"localhost:7681"|"primary"  |true    |"online"
|"online"       |""             |false     |false |[]           |
|"foo"    |"standard" |[]       |"read-write"|"localhost:7684"|"secondary"|false   |"online"
|"online"       |""             |false     |false |[]           |
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------+
```

Note that `SHOW DATABASES` does **not** return 1 row per database. Instead, it returns 1 row per database, per server that hosts it in the cluster. Therefore, if the cluster has four servers hosting 3 databases each with 3 primaries and one secondary, 12 rows are displayed. In addition this means that if all the servers that host a database are offline, the database will not appear in the results of `SHOW DATABASES`.

The possible roles are "primary", "secondary", and "unknown".

Note that different servers may have different roles for each database, and a server may have different roles for different databases.

If a database is offline on a particular server, either because it was stopped by an operator, or an error has occurred, its cluster `role` is "unknown".

Listing a single database

The number of rows returned by `SHOW DATABASES` can be quite large, especially when run in a cluster. You can filter the rows returned by database name (e.g. "foo") by using the command `SHOW DATABASE foo`.

Syntax:

```
SHOW DATABASE databaseName
```

Arguments:

Name	Type	Description
`databaseName`	String	The name of the database whose status to report.

Returns:

Name	Type	Description
`name`	String	The human-readable name of the database.
`type`	String	`standard`, `system`, or `composite`.
`aliases`	List<String>	The names of any aliases the database may have.
`access`	String	The database access mode, either `read-write` or `read-only`.
`address`	String	The bolt address of the server hosting the database.
`role`	String	The cluster role which the server fulfills for this database.
`writer`	Boolean	`true` for the database node that accepts writes. This node is either the leader for this database in a cluster or this is a standalone server.
`requestedStatus`	String	The state that an operator has requested the database to be in.
`currentStatus`	String	The state the database is actually in on this server.

Name	Type	Description
`statusMessage`	String	A message explaining the current state of the database, which could be an error encountered by the Neo4j server when transitioning the database to `requestedStatus`, if any.
`default`	Boolean	Whether this database is the default for this DBMS.
`home`	Boolean	Whether this database is the home database for this user.
`constituents`	List<String>	A list of alias names making up this composite database, null for non-composite databases.

Example 65. Listing statuses for database foo

When running `SHOW DATABASE foo` in a cluster, expect similar output to the following:

```
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------
------------------+
| name    | type       | aliases    | access      | address          | role      | writer  |
requestedStatus  | currentStatus  | statusMessage                                        | default  |
home  | constituents |
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------
------------------+
| "foo"    | "standard" | []         | "read-write" | "localhost:7681" | "primary" | false    |
"online"          | "online"       | ""                                                 | true      |
true   | []             |
| "foo"    | "standard" | []         | "read-write" | "localhost:7682" | "unknown" | false    |
"online"          | "dirty"        | "An error occurred! Unable to start database ..." | true      |
true   | []             |
| "foo"    | "standard" | []         | "read-write" | "localhost:7683" | "primary" | true     |
"online"          | "online"       | ""                                                 | true      |
true   | []             |
| "foo"    | "standard" | []         | "read-write" | "localhost:7684" | "unknown" | false    |
"online"          | "dirty"        | "An error occurred! Unable to start database ..." | true      |
true   | []             |
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------
------------------+
```

Listing more details about databases

If more details about the databases are needed, `SHOW DATABASES` can be appended with `YIELD *`.

Syntax:

```
SHOW DATABASES YIELD *
```

Returns:

Name	Type	Description	Example value
`name`	String	The human-readable name of the database.	"foo"
`type`	String	`standard`, `system` or `composite`	"standard"
`aliases`	List<String>	Aliases of the database.	"[]"
`access`	String	`read-write` or `read-only`	"read-write"
`databaseID`	String	The ID for the database.	"CC573A1DF4...."
`serverID`	String	The friendly name or UUID of the server hosting this database.	"server3"
`address`	String	The Bolt address of the server hosting the database.	"localhost:7683"
`role`	String	The cluster role which the server fulfills for this database.	"primary"
`writer`	Boolean	Whether the database accepts writes on this server.	true
`requestedStatus`	String	The state that an operator has requested the database to be in.	"online"
`currentStatus`	String	The state the database is actually in on this server.	"online"
`statusMessage`	String	Error encountered by the server when transitioning the database to `requestedStatus`, if any.	""
`default`	Boolean	Whether this database is the default for this DBMS.	false
`home`	Boolean	Whether this database is the user's home database.	true
`currentPrimariesCount`	Integer	Number of primaries for this database reported as running currently. It is the same as the number of rows where `role`=`primary` and `name`=this database	3
`currentSecondariesCount`	Integer	Number of secondaries for this database reported as running currently. It is the same as the number of rows where `role`=`secondary` and `name`=this database	0

Name	Type	Description	Example value
`requestedPrimariesCount`	Integer	The requested number of primaries for this database. May be lower than current if the DBMS is currently reducing the number of copies of the database, or higher if it is currently increasing the number of copies.	3
`requestedSecondariesCount`	Integer	The requested number of secondaries for this database. May be lower than current if the DBMS is currently reducing the number of copies of the database, or higher if it is currently increasing the number of copies.	1
`creationTime`	Datetime	The timestamp of the creation of this database.	"2022-09-09T12:58:21.923000000Z"
`lastStartTime`	Datetime	The timestamp of the most recent time this database was started It is the same as creation time unless the database has been stopped at some point.	"2022-09-09T12:58:21.923000000Z"
`lastStopTime`	Datetime	The timestamp of the most recent time this database was stopped (`STOP DATABASE`)	null
`store`	String	The store format.	"record-aligned-1.1"
`lastCommittedTxn`	Integer	The latest committed transaction number on this database server. May be different between members when changes have not propagated.	2342
`replicationLag`	Integer	The difference in transaction numbers between this server and the writer of this database. If this is persistently high, there may be a problem.	1
`constituents`	List<String>	A list of alias names making up this composite database, null for non-composite databases.	"[]"

Disaster recovery

Databases can become unavailable for different reasons. For the purpose of this section, an *unavailable database* is defined as a database that is incapable of serving writes, while still may be able to serve reads. Databases not performing as expected for other reasons are not considered unavailable and cannot be helped by this section. This section contains a step-by-step guide on how to recover databases that have become unavailable. By performing the actions described here, the unavailable databases are recovered and made fully operational with as little impact as possible on the other databases in the cluster.

There are many reasons why a database becomes unavailable and it can be caused by issues on different levels in the system. For example, a data-center failover may lead to the loss of multiple serves which in turn may cause a set of databases to become unavailable. It is also possible for databases to become quarantined due to a critical failure in the system which may lead to unavailability even without loss of servers.

> If **all** servers in a Neo4j cluster are lost in a data-center failover, it is not possible to recover the current cluster. A new cluster has to be created and the databases restored. See Deploy a basic cluster and Seed a database for more information.

Faults in clusters

Databases in clusters follow an allocation strategy. This means that they are allocated differently within the cluster and may also have different numbers of primaries and secondaries. The consequence of this is that all servers are different in which databases they are hosting. Losing a server in a cluster may cause some databases to lose a member while others are unaffected. Consequently, in a disaster where multiple servers go down, some databases may keep running with little to no impact, while others may lose all their allocated resources.

Guide to disaster recovery

The following guide defines three stages of disaster recovery. Always run the guide in the order it is described and only move to the next stage once the current one is validated to work properly.

> In this section, an *offline* server is a server that is not running but may be *restartable*. A *lost* server however, is a server that is currently not running and cannot be restarted.

> Disasters may sometimes affect the routing capabilities of the driver and may prevent the use of the `neo4j` scheme for routing. One way to remedy this is to connect directly to the server using `bolt` instead of `neo4j`. See Server-side routing for more information on the `bolt` scheme.

Restore the `system` database

The first step of recovery is to ensure that the `system` database is available. The `system` database is required for clusters to function properly.

1. Start all servers that are *offline*. (If a server is unable to start, inspect the logs and contact support personnel. The server may have to be considered indefinitely lost.)
2. **Validate the `system` database's availability.**
    a. Run `SHOW DATABASE system`. If the response doesn't contain a writer, the `system` database is unavailable and needs to be recovered, continue to step 3.
    b. Run `CREATE USER 'temporaryUser' SET PASSWORD 'temporaryPassword'` to create a temporary user.
        i. Confirm that the query was executed successfully and the temporary user was created as expected, by running `SHOW USERS`, then continue to Recover servers. If not, continue to step 3.

3. Restore the `system` database.

   > Only do the steps below if the `system` database's availability could not be validated by the first two steps in this section.

   The following steps creates a new `system` database from a backup of the current `system` database. This is required since the current `system` database has lost too many members in the server failover.

   a. Shut down the Neo4j process on all servers. Note that this causes downtime for all databases in the cluster.

   b. On each server, run the following `neo4j-admin` command `neo4j-admin dbms unbind-system-db` to reset the `system` database state on the servers. See `neo4j-admin` commands for more information.

   c. On each server, run the following `neo4j-admin` command `neo4j-admin database info system` to find out which server is most up-to-date, ie. has the highest last-committed transaction id.

   d. On the most up-to-date server, take a dump of the current `system` database by running `neo4j-admin database dump system --to-path=[path-to-dump]` and store the dump in an accessible location. See `neo4j-admin` commands for more information.

   e. Ensure there are enough `system` database primaries to create the new `system` database with fault tolerance. Either:

      i. Add completely new servers (see Add a server to the cluster) or

      ii. Change the `system` database mode (`server.cluster.system_database_mode`) on the current `system` database's secondary servers to allow them to be primaries for the new `system` database.

   f. On each server, run `neo4j-admin database load system --from-path=[path-to-dump] --overwrite-destination=true` to load the current `system` database dump.

   g. Ensure that `dbms.cluster.discovery.endpoints` are set correctly on all servers, see Cluster server discovery for more information.

   h. Return to step 1.

Recover servers

Once the `system` database is available, the cluster can be managed. Following the loss of one or more servers, the cluster's view of servers must be updated, ie. the lost servers must be replaced by new servers. The steps here identify the lost servers and safely detach them from the cluster.

1. Run `SHOW SERVERS`. If all servers show health `AVAILABLE` and status `ENABLED` continue to Recover databases.

2. On each `UNAVAILABLE` server, run `CALL dbms.cluster.cordonServer("unavailable-server-id")`.

3. On each `CORDONED` server, run `DEALLOCATE DATABASES FROM SERVER cordoned-server-id`.

4. On each server that failed to deallocate with one of the following messages:

   a. `Could not deallocate server [server]. Can't move databases in single mode [database]`

      or

`Could not deallocate server [server]. Database [database] has lost quorum of servers, only found [existing number of primaries] of [expected number of primaries]. Cannot be safely deallocated. Please drop the database before retrying.`

First ensure that there is a backup for the database in question (see Online backup), and then drop the database by running `DROP DATABASE database-name`. Return to step 3.

b. `Could not deallocate server [server]. Cannot change allocations for database [stopped-db] because it is offline.`

   Try to start the offline database by running `START DATABASE stopped-db WAIT`. If it starts successfully, return to step 3. Otherwise, ensure that there is a backup for the database before dropping it with `DROP DATABASE stopped-db`. Return to step 3.

   > A database can be set to `READ-ONLY`-mode before it is started to avoid updates on a database that is desired to be stopped with the following: `ALTER DATABASE database-name SET ACCESS READ ONLY`.

c. `Could not deallocate server [server]. Reallocation of [database] not possible, no new target found. All existing servers: [existing-servers]. Actual allocated server with mode [mode] is [current-hostings].`

   Add new servers and enable them and then return to step 3, see Add a server to the cluster for more information.

5. Run `SHOW SERVERS YIELD *` once all enabled servers host the requested databases (`hosting`-field contains exactly the databases in the `requestedHosting` field), proceed to the next step. Note that this may take a few minutes.
6. For each deallocated server, run `DROP SERVER deallocated-server-id`.
7. Return to step 1.

Recover databases

Once the `system` database is verified available, and all servers are online, the databases can be managed. The steps here aim to make the unavailable databases available.

1. If you have previously dropped databases as part of this guide, re-create each one from backup. See the Administrative commands section for more information on how to create a database.
2. Run `SHOW DATABASES`. If all databases are in desired states on all servers (`requestedStatus` =`currentStatus`), disaster recovery is complete.

Settings reference

Parameter	Explanation
`initial.server.mode_constraint`	This setting constrains the operating mode of the database to be used only in primary or secondary mode. Default setting is `NONE`, ie. no constraint. As an initial setting, the value set here is used when a server is first enabled. Once enabled, a server's mode constraint can only be changed with `ALTER SERVER 'name' SET OPTIONS {modeConstraint:'PRIMARY'}`. **Example**: a server configured with `initial.server.mode_constraint=SECONDARY` is only allocated databases whose topologies contain 1 or more secondary. This server always only hosts those databases in `SECONDARY` mode.
`server.cluster.system_database_mode`	Every cluster member hosts the `system` database. This config controls what mode a given instance hosts the `system` database in: `PRIMARY` or `SECONDARY`. **Example**: `server.cluster.system_database_mode=SECONDARY` means that this instance holds only a secondary copy of the `system` database. ℹ There should be a relatively high number (5-7) of `system` primaries, spread across availability zones. However, if enabling more than 10 servers, it is recommended to start making the later ones secondaries.
`dbms.cluster.minimum_initial_system_primaries_count`	Minimum number of servers configured with `server.cluster.system_database_mode=PRIMARY` required to form a cluster. **Example**: `dbms.cluster.minimum_initial_system_primaries_count=3` specifies that the cluster is considered bootstrapped and the DBMS online when at least 3 `system` database primaries have discovered one another.

Parameter	Explanation

`dbms.cluster.discovery.type`	This setting specifies the strategy that the instance uses to determine the addresses for other instances in the cluster to contact for *bootstrapping*. Possible values are:
	
	`LIST`
	Treats `dbms.cluster.discovery.endpoints` as a list of addresses of servers to contact for discovery.
	
	`DNS`
	Treats `dbms.cluster.discovery.endpoints` as a domain name to resolve via DNS. Expect DNS resolution to provide A records with hostnames or IP addresses of servers to contact for discovery, on the port specified by `dbms.cluster.discovery.endpoints`.
	
	`SRV`
	Treats `dbms.cluster.discovery.endpoints` as a domain name to resolve via DNS. Expect DNS resolution to provide SRV records with hostnames or IP addresses and ports, of servers to contact for discovery.
	
	`K8S`
	Accesses the Kubernetes list service API to derive addresses of servers to contact for discovery. Requires `dbms.kubernetes.label_selector` to be a Kubernetes label selector for Kubernetes services running a server each and `dbms.kubernetes.service_port_name` to be a service port name identifying the discovery port of cluster servers services. The value of `dbms.cluster.discovery.endpoints` is ignored for this option.
	
	The value of this setting determines how `dbms.cluster.discovery.endpoints` is interpreted. Detailed information about discovery and discovery configuration options is given in Discovery using DNS with multiple records.
	
	Example: `clustering-discovery-dns=DNS` combined with `dbms.cluster.discovery.endpoints=cluster01.example.com:5000` fetch all DNS A records for *cluster01.example.com* and attempt to reach Neo4j instances listening on port `5000` for each A record's IP address.

Parameter	Explanation
`dbms.cluster.discovery.endpoints`	One or more network addresses used to discover other servers in the cluster. The exact method by which endpoints are resolved to other cluster members is determined by the value of `dbms.cluster.discovery.type`. In the default case, the initial discovery members are given as a comma-separated list of address/port pairs, and the default port for the discovery service is `:5000`. It is good practice to set this parameter to the same value on all servers in the cluster. The behavior of this setting can be modified by configuring the setting `dbms.cluster.discovery.type`. This is described in detail in Discovery using DNS with multiple records. **Example:** `dbms.cluster.discovery.type=LIST` combined with `server01.example.com:5000,server02.example.com:5000,server03.example.com:5000` attempt to reach Neo4j instances listening on *server01.example.com*, *server02.example.com* and *server03.example.com*; all on port `5000`.
`server.discovery.advertised_address`	The address/port setting that specifies where the instance advertises that it listens for discovery protocol messages from other members of the cluster. If this server is included in the `discovery.endpoints` of other cluster members, the value there must **exactly** match this advertised address. **Example:** `server.discovery.advertised_address=192.168.33.21:5001` indicates that other cluster members can communicate with this server using the discovery protocol at host `192.168.33.20` and port `5001`.
`server.cluster.raft.advertised_address`	The address/port setting that specifies where the Neo4j server advertises to other members of the cluster that it listens for Raft messages within the cluster. **Example:** `server.cluster.raft.advertised_address=192.168.33.20:7000` listens for cluster communication in the network interface bound to `192.168.33.20` on port `7000`.

Parameter	Explanation
`server.cluster.advertised_address`	The address/port setting that specifies where the instance advertises it listens for requests for transactions in the transaction-shipping catchup protocol. **Example:** `causal_clustering.transaction_advertised_address=192.168.33.20:6001` listens for transactions from cluster members on the network interface bound to `192.168.33.20` on port `6001`.
`server.discovery.listen_address`	The address/port setting that specifies which network interface and port the Neo4j instance binds to for the cluster discovery protocol. **Example:** `server.discovery.listen_address=0.0.0.0:5001` listens for cluster membership communication on any network interface at port `5001`.
`server.cluster.raft.listen_address`	The address/port setting that specifies which network interface and port the Neo4j instance binds to for cluster communication. This setting must be set in coordination with the address this instance advertises it listens at in the setting `server.cluster.raft.advertised_address`. **Example:** `server.cluster.raft.listen_address=0.0.0.0:7000` listens for cluster communication on any network interface at port `7000`.
`server.cluster.listen_address`	The address/port setting that specifies which network interface and port the Neo4j instance binds to for cluster communication. This setting must be set in coordination with the address this instance advertises it listens at in the setting `server.cluster.advertised_address`. **Example:** `server.cluster.listen_address=0.0.0.0:6001` listens for cluster communication on any network interface at port `6001`.

Clustering glossary

Term	Description
Allocator	A component in the cluster that allocates databases to servers according to the topology constraints specified and an allocation strategy.
Asynchronous replication	Enables efficient scale-out of secondary database copies but offers no guarantees under fault conditions. The data present in the secondary copy is not guaranteed to be up-to-date with a majority of the database's primary copies.
Availability	The ability to access data in a database. A database can be available for read-write, read-only, or altogether unavailable. A clustered database is fault-tolerant, i.e. it can maintain both read and write availability if some primaries fail (see Fault tolerance for more information). If the number of failed primaries exceeds the fault tolerance limit, the database becomes read-only. Should all copies fail, the database becomes unavailable.
Bookmark	A marker the client can request from the cluster to ensure that it is able to read its own writes so that the application's state is consistent and only databases that have a copy of the bookmark are permitted to respond.
Causal consistency	When a client (driver) creates a session and executes a query, the responding server issues the client a bookmark. This reflects the state of the database copy on that server at the time the query was executed. The bookmark is passed along and updated by all subsequent queries in the session, regardless of which server executes what query. A bookmark can only be updated monotonically increasing. If a server is behind the state in the bookmark, it waits until it has caught up, or time out the query. Thus, clients executing queries within a session are guaranteed to read their own writes, and only see successively later states of the database. This is sometimes also referred to as session consistency.
Cluster	A collection of servers running Neo4j that are configured to communicate with each other. These may be used to host databases and the databases may be configured to replicate across servers in the cluster thus achieving read scalability or high availability. A minimum of three servers is required for the cluster to be fault-tolerant.

Term	Description
Database	The data store for the nodes, relationships, and properties that make up the graph. Multiple databases can be hosted on a Database Management Server (DBMS).
Database Management Server (DBMS)	The Neo4j services and system database running on an instance of a single server or cluster to provide one or more databases.
Deallocate	An act of safely removing (i.e. without loss of data or reduced fault tolerance) a database from a server, or removing a server from a cluster.
Disaster recovery	A manual intervention to restore availability of a cluster, or databases within a cluster.
Election	In the event that a leader becomes unresponsive, followers automatically trigger an election and vote for a new leader. A majority is required for the vote to be successful.
Fault tolerance	A guarantee that a database can maintain persistence and availability in the event of one or more failures. The number of failures `f` that can be tolerated is dependent on the number of primaries `n` for the database and follows the formula `f = (n-1)/2`. In the event that more than `f` primaries fail, the database can no longer process write transactions and becomes read-only.
Follower	A primary copy of a database acting as a follower, receives and acknowledges synchronous writes from the leader.
Leader	A single primary copy of a database is designated as the leader. It receives all write transactions from clients and replicates writes synchronously to followers and asynchronously to secondary copies of the database. Each database can have a different leader within the cluster.
Primary	A copy of the database that is able to process write transactions and is eligible to be elected as a leader. It participates in fault tolerant writes as it is part of the majority required to acknowledge and commit write transactions.

Term	Description
Read scaling	Adding secondary copies of the database to the cluster can offload read queries from the primary databases and thus reduce the load and aid write performance of the cluster.
Secondary	An asynchronously replicated copy of the database that provides read scaling within the cluster. It is also suitable for running graph analytic workloads in a cluster using Graph Data Science and taking backups without incurring load on the primary.
Seed	A file used to create a copy of a database on a single instance or on a member of a cluster. This can be a database dump or a database backup. *Seed* can also be used as a verb to describe the act seeding a cluster from a backup.
Server	A physical machine, a virtual machine, or a container running Neo4j DBMS. The server can be standalone or part of a cluster.
Session consistency	An alternative name for Neo4j's causal consistency.
Standalone server	A single server, or container, running Neo4j DBMS and not part of a cluster.
Synchronous replication	When attempting to commit a transaction, the leader primary replicates the transaction and block, requiring the follower primaries to acknowledge the replication before allowing the commit to proceed. This blocking replication is known as *synchronous*, and ensures data durability and consistency within the cluster. See also asynchronous replication.
Topology	A configuration that describes how the copies of a database should be spread across the servers in a cluster.

Backup and restore

This chapter describes the following:

- Backup and restore planning — What to consider when designing your backup and restore strategy.
- Backup modes — The supported backup modes.
- Back up an online database — How to back up an online database.
- Aggregate a database backup chain - How to aggregate a backup chain into a single backup.
- 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.

Backup and restore planning

There are two main reasons for backing up your Neo4j databases and storing them in a safe, off-site location:

- to be able to quickly recover your data in case of failure, for example related to hardware, human error, or natural disaster.
- to be able to perform routine administrative operations, such as moving a database from one instance to another, upgrading, or reclaiming space.

Backup and restore strategy

Depending on your particular deployment and environment, it is important to design an appropriate backup and restore strategy.

There are various factors to consider when deciding on your strategy, such as:

- Type of environment – development, test, or production.
- Data volumes.
- Number of databases.
- Available system resources.
- Downtime tolerance during backup and restore.
- Demands on Neo4j performance during backup and restore. This factor might lead your decision towards performing these operations during an off-peak period.
- Tolerance for data loss in case of failure.
- Tolerance for downtime in case of failure. If you have zero tolerance for downtime and data loss, you might want to consider performing an online or even a scheduled backup.
- Frequency of updates to the database.

- Type of backup and restore method (online or offline), which may depend on whether you want to:
  - perform full backups (online or offline).
  - perform differential backups (online only).
  - use SSL/TLS for the backup network communication (online only).
  - keep your databases as archive files (online or offline).
- How many backups you want to keep.
- Where the backups will be stored — drive or remote server, cloud storage, different data center, different location, etc.

> It is recommended to store your database backups on a separate off-site server (drive or remote) from the database files. This ensures that if for some reason your Neo4j DBMS crashes, you will be able to access the backups and perform a restore.

- How you will test recovery routines, and how often.

Backup and restore options

Neo4j supports backing up and restoring both online and offline databases. It uses Neo4j Admin tool commands, 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 database backup/restore` (Enterprise only) -– used for performing online backup (full and differential) and restore operations. The database to be backed up must be in **online** mode. The command produces an immutable artifact, which has an inspectable API to aid management and operability. This command is suitable for production environments, where you cannot afford downtime.

  The command can also be invoked over the network if access is enabled using `server.backup.listen_address`.

  > Make sure to limit access to the backup server port to fully trusted, specific devices. Firewall policies should be considered. For more information, refer to the Server configurations section.

  > When using `neo4j-admin database backup` in a cluster, it is recommended to back up from an external instance as opposed to reuse instances that form part of the cluster.

- `neo4j-admin database dump/load` –- used for performing offline dump and load operations. The database to be dumped must be in **offline** mode. The dump command can only be invoked from the server command line and is suitable for environments, where downtime is not a factor. The command produces an archive file that follows the format *<databasename><timestamp>.dump*.
- `neo4j-admin database copy` –- used for copying an offline database or backup. This command can be used for cleaning up database inconsistencies and reclaiming unused space.

> File system copy-and-paste of databases is not supported and may lead to corruption.

Considerations for backing up and restoring databases in a cluster

Backing up a database in a clustered environment is not essentially different from a standalone backup, apart from the fact that you must know which server in a cluster to connect to. Use `SHOW DATABASE <database>` to learn which servers are hosting the database you want to back up. See Listing a single database for more information.

However, *restoring* a database in a cluster is different since it is not known in advance how a database is going to be allocated to the servers in a cluster. This method relies on the seed already existing on one of the servers. The recommended way to restore a database in a cluster is to seed from URI.

The Neo4j Admin commands `backup`, `restore`, `dump`, `load`, `copy`, and `check-consistency` are not supported for use on composite databases. They must be run directly on the databases that are part of a composite database.

Table 37. The following table describes the commands' capabilities and usage.

Capability/ Usage	`backup/restore`	`dump/load`	`copy`
Neo4j Edition	Enterprise	all	Enterprise
Run from an online Neo4j DBMS	✓	✓	✓
Run from an offline Neo4j DBMS	✗	✓	✓
Run against a user database	✓	✓	✓
Run against the `system` database	✓	✓	✗
Run against a composite databases	✗	✗	✗
Perform full backups	✓	✓	n/a
Perform differential backups	✓	✗	n/a
Applied to an online database	✓	✗	✗
Applied to an offline database	only `restore`	✓	✓
Can be run remotely	only `backup`	✗	✓
Command input	database/archive (.backup)	database/archive (.dump)	database
Command output	archive (.backup)/database	archive (.dump)/database	database; no schema store
Clean up database inconsistencies	✗	✗	✓
Compact data store	✓	✗	✓

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 is backed up independently of one another.

ℹ	It is very important to store a recent backup of your databases, including the `system` database, in a safe location.

Additional files to back up

The following files must be backed up separately from the databases:

- The neo4j.conf file. If you have a cluster deployment, you should back up the configuration file for each cluster member.
- All the files used for encryption, i.e., private key, public certificate, and the contents of the *trusted* and *revoked* directories. The locations of these are described in SSL framework. If you have a cluster, you should back up these files for each cluster member.
- If using custom plugins, make sure that you have the plugins in a safe location.

Storage considerations

For any backup, it is important that you store your data separately from the production system, where there are no common dependencies, and preferably off-site. If you are running Neo4j in the cloud, you may use a different availability zone or even a separate cloud provider. Since backups are kept for a long time, the longevity of archival storage should be considered as part of backup planning.

Backup modes

The backup client can operate in two different modes – a *full backup* and an *differential backup*.

Full backup

A full backup is always required initially for the very first backup into a target location.

ℹ	The full backup can be run against both an **online** (using `neo4j-admin database backup`) and an **offline** (using `neo4j-admin database dump`) database.

Example 66. Full backup against an online database

```
$neo4j-home> export HEAP_SIZE=2G
$neo4j-home> mkdir /mnt/backups
$neo4j-home> bin/neo4j-admin database backup --from=192.168.1.34 --to-path=/mnt/backups/neo4j
--pagecache=4G neo4j
Doing full backup...
2017-02-01 14:09:09.510+0000 INFO  [o.n.c.s.StoreCopyClient] Copying neostore.nodestore.db.labels
2017-02-01 14:09:09.537+0000 INFO  [o.n.c.s.StoreCopyClient] Copied neostore.nodestore.db.labels 8.00
kB
2017-02-01 14:09:09.538+0000 INFO  [o.n.c.s.StoreCopyClient] Copying neostore.nodestore.db
2017-02-01 14:09:09.540+0000 INFO  [o.n.c.s.StoreCopyClient] Copied neostore.nodestore.db 16.00 kB
...
...
...
```

For more information about online backup options and how to control memory usage, see Back up an online database.

A full online database backup creates a full backup artifact in the target location (`--to-path`).

For more information about performing a full backup against an **offline** database, see Back up an offline database.

Differential backup

In the online backup version, after the initial full backup, the subsequent backups attempt to use the differential mode, where just the delta of the transaction logs since the last backup are transferred and used to create a differential backup artifact (stored in the target location). Those differential backup artifacts form a backup chain. If the required transaction logs are not available on the backup server, then the backup client falls back on performing a full backup instead.

Example 67. Differential backup against an online database

```
$neo4j-home> export HEAP_SIZE=2G
$neo4j-home> bin/neo4j-admin database backup --from=192.168.1.34 --to-path=/mnt/backups/neo4j
--pagecache=4G neo4j
Destination is not empty, doing differential backup...
Backup complete.
```

For more information about online backup options and how to control memory usage, see Back up an online database.

Back up an online database

Remember to plan your backup carefully and to back up each of your databases, including the `system` database.

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.

> Neo4j v5.0 introduces a new version of the backup command which produces immutable backup artifacts (as opposed to mutable folders as in previous versions).

Backup artifact

The `neo4j-admin database backup` command produces one backup artifact file per database each time it is run. A backup artifact file is an immutable file containing the backup data of a given database along with some metadata like the database name and id, the backup time, the lowest/highest transaction id etc.

Backup artifacts can be of two types:

1. a *full backup* containing the whole database store or
2. a *differential backup* containing a log of transactions to apply to a database store contained in a full backup artifact.

Backup chain

The first time the backup command is run, a full backup artifact is produced for a given database. On the other hand, differential backup artifacts are produced by the subsequent runs.

A *backup chain* consists of a full backup optionally followed by a sequence of n contiguous differential backups.

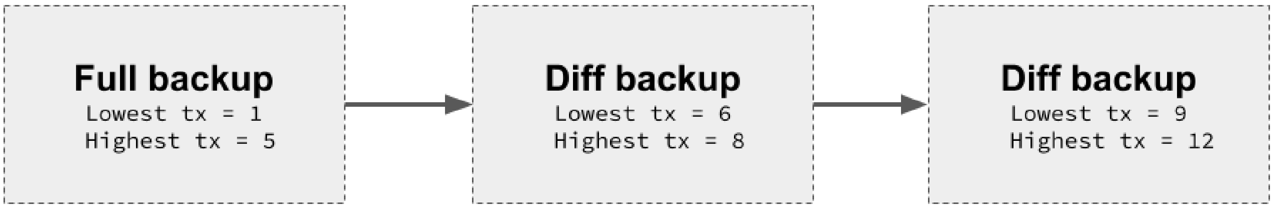


Figure 6. Backup chain

Usage

The `neo4j-admin database backup` command can be used for performing both full and differential backups of an **online** database. The command can be run both locally and remotely. However, it uses a significant amount of resources, such as memory and CPU. Therefore, it is recommended to perform the backup on a separate dedicated machine. The `neo4j-admin database backup` command also supports SSL/TLS. For more information, see Online backup configurations.

> `neo4j-admin database backup` is not supported in Neo4j Aura.

Syntax

```
neo4j-admin database backup     --to-path=<path>
                                [--from=<host:port>[,<host:port>]...]
                                [--type=<type>]
                                [--compress=<true/false>]
                                [--keep-failed=<true/false>]
                                [--pagecache=<size>]
                                [--include-metadata=<all/users/roles>]
                                [--parallel-recovery=<true/false>]
                                [--inspect-path=<path>]
                                [--verbose]
                                [--expand-commands]
                                [--additional-config=<path>]
                                <database>
```

Options

Option	Default	Description
`--to-path`		Directory to place backup in.
`--from`	`localhost:6362`	Comma-separated list of host and port of Neo4j instances, each of which are tried in order.
`--type`	`AUTO`	Type of backup to perform. Possible values are: `FULL`, `DIFF`, `AUTO`.
`--compress`	`true`	Request backup artifact to be compressed. If disabled, backup artifact creation is faster but the size of the produced artifact is approximately equal to the size of backed-up database.
`--keep-failed`	`false`	Request failed backup to be preserved for further post-failure analysis. If enabled, a directory with the failed backup database is preserved.
`--pagecache`	`8m`	The size of the page cache to use for the backup process.

Option	Default	Description
`--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.
`--parallel-recovery`	`false`	Allow multiple threads to apply transactions to a backup in parallel. For some databases and workloads, this may reduce execution times significantly. ℹ `parallel-recovery` is an experimental option. Consult Neo4j support before use.
`--inspect-path`		List and show the metadata of the backup artifact(s). Accepts a folder or a file.
`--verbose`		Enable verbose output.
`--expand-commands`		Allow command expansion in config value evaluation.

Option	Default	Description
`--additional-config`		Configuration file to provide additional or override the existing configuration settings in the *neo4j.conf* file.

Parameters

Parameter	Default	Description
`<database>`		Name of the remote database to back up. The value can contain `*` and `?` for globbing, in which cases all matching databases are backed up. With a single `*` as a value, you can back up all the databases of the DBMS.

Exit codes

Depending on whether the backup was successful or not, `neo4j-admin database backup` exits with different codes. The error codes include details of what error was encountered.

Table 38. Neo4j Admin backup exit codes when backing up one database

Code	Description
`0`	Success.
`1`	Backup failed, or succeeded but encountered problems such as some servers being uncontactable. See logs for more details.

Table 39. Neo4j Admin backup exit codes when backing multiple databases

Code	Description
`0`	All databases are backed up successfully.
`1`	One or several backups failed, or succeeded with problems.

Online backup configurations

Server configuration

The table below lists the basic server parameters relevant to backups. Note that by default, the backup service is enabled but only listens on localhost (127.0.0.1). This needs to be changed if backups are to be taken from another machine.

Table 40. Server parameters for backups

Parameter name	Default value	Description
`server.backup.enabled`	`true`	Enable support for running online backups.
`server.backup.listen_address`	`127.0.0.1:6362`	Listening server for online backups.

Memory configuration

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 database backup` command.

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

Transaction log files

The transaction log files, which keep track of recent changes, are rotated and pruned based on a provided configuration. For example, setting `db.tx_log.rotation.retention_policy=3` files keeps 3 transaction log files in the backup. Because recovered servers do not need all of the transaction log files that have already been applied, it is possible to further reduce storage size by reducing the size of the files to the bare minimum. This can be done by setting `db.tx_log.rotation.size=1M` and db.tx_log.rotation.retention_policy=3 files. You can use the --additional-config parameter to override the configurations in the *neo4j.conf* file.

Removing transaction logs manually can result in a broken backup.

Security configurations

Securing your backup network communication with an SSL policy and a firewall protects your data from unwanted intrusion and leakage. When using the neo4j-admin database backup command, you can configure the backup server to require SSL/TLS, and the backup client to use a compatible policy. For more information on how to configure SSL in Neo4j, see SSL framework.

The default backup port is 6362, configured with key server.backup.listen_address. The SSL configuration policy has the key of dbms.ssl.policy.backup.

As an example, add the following content to your *neo4j.conf* file:

```
dbms.ssl.policy.backup.enabled=true
dbms.ssl.policy.backup.tls_versions=TLSv1.2
dbms.ssl.policy.backup.ciphers=TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
dbms.ssl.policy.backup.client_auth=REQUIRE
```

For a detailed list of recommendations regarding security in Neo4j, see Security checklist.

It is very important to ensure that there is no external access to the port specified by the setting server.backup.listen_address. Failing to protect this port may leave a security hole open by which an unauthorized user can make a copy of the database onto a different machine. In production environments, external access to the backup port should be blocked by a firewall.

Cluster configurations

In a cluster topology, it is possible to take a backup from any server hosting the database to backup, and each server has two configurable ports capable of serving a backup. These ports are configured by server.backup.listen_address and server.cluster.listen_address respectively. Functionally, they are equivalent for backups, but separating them can allow some operational flexibility, while using just a single port can simplify the configuration. It is generally recommended to select secondary servers to act as backup servers, since they are more numerous than primary servers in typical cluster deployments. Furthermore, the possibility of performance issues on a secondary server, caused by a large backup, does not affect the performance or redundancy of the primary servers. If a secondary server is not available, then a primary can be selected based on factors, such as its physical proximity, bandwidth, performance, and liveness.

Use the SHOW DATABASES command to learn which database is hosted on which server.

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.

Targeting multiple servers

It is recommended to provide a list of multiple target servers when taking a backup from a cluster, since that may allow a backup to succeed even if some server is down, or not all databases are hosted on the same servers. If the command finds one or more servers that do not respond, it continues trying to backup from other servers, and continue backing up other requested databases, but the exit code of the command is non-zero, to alert the user to the fact there is a problem. If a name pattern is used for the database together with multiple target servers, all servers contribute to the list of matching databases.

Examples

The following are examples of how to back up a single database, e.g., the default database `neo4j`, and multiple databases, using the `neo4j-admin database backup` command. The target directory */mnt/backups/neo4j* must exist before calling the command and the database(s) must be online.

Example 68. Use `neo4j-admin database backup` *to back up a single database.*

```
bin/neo4j-admin database backup --to-path=/mnt/backups/neo4j neo4j
```

To backup several databases that match database pattern you can use name globbing. For example, to backup all databases that start with n from your three-node cluster you should run:

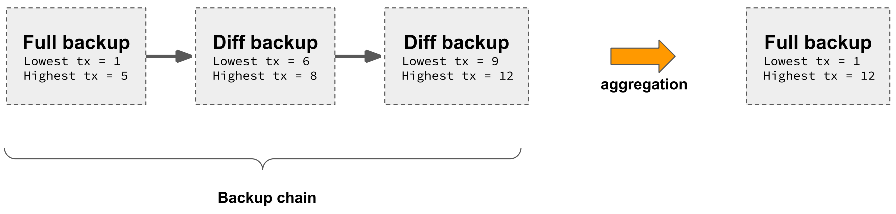
Example 69. Use `neo4j-admin database backup` *to back up multiple databases.*

```
neo4j-admin database backup --from=192.168.1.34,192.168.1.35,192.168.1.36 --to-path
=/mnt/backups/neo4j --pagecache=4G n*
```

Aggregate a database backup chain

Command

The aggregate command turns a backup chain into a single full backup artifact.



The benefits of aggregating a backup chain are notably:

- Reduces the size of backup artifacts in a given backup folder.
- Keeps the recovery time objective (RTO) low by generating a single backup artifact ready to be restored. As part of the aggregation, transactions contained in the differential backups are applied to the store contained in the full backup artifact. This operation is called *recovery* and can be costly.
- Reduces the risk of losing chain's links.

Syntax

```
neo4j-admin database aggregate-backup   --from-path=<path>
                                        [--keep-old-backup[=true|false]]
                                        [--parallel-recovery[=true|false]]
                                        [--verbose]
                                        [--expand-commands]
                                        [--additional-config=<file>]
                                        <database>
```

Options

Option	Default	Description
`--from-path`		Directory where the backup artifacts are located. Directory where the backup artifacts are located, or optionally the path to a specific *.backup file* forcing `aggregate` to find the backup chain for that specific backup (and guarding against the risk of identifying the wrong backup chain, if more than one exists in a given directory). When a file is supplied, the *<database>* parameter should be omitted. The option to supply a file is only available in Neo4j 5.2 and later.
`--keep-old-backup`	`false`	If set to true, the old backup chain is not removed.
`--parallel-recovery`	`false`	Allow multiple threads to apply pulled transactions to a backup in parallel. For some databases and workloads this may reduce aggregate times significantly. `parallel-recovery` is an experimental option. Consult Neo4j support before use.
`--verbose`		Enable verbose output.
`--expand-commands`		Allow command expansion in config value evaluation.
`--additional-config`		Configuration file to provide additional or override the existing configuration settings in the *neo4j.conf* file.

Parameters

Parameter	Default	Description
`<database>`	`neo4j`	Name of the database for which to aggregate the artifacts. Can contain `*` and `?` for globbing.

Examples

An example of how to perform aggregation of a set of backups located in a given folder for the `neo4j` *database.*

```
bin/neo4j-admin database aggregate-backup --from-path=/mnt/backups/ neo4j
```

The command first looks inside the `/mnt/backups/` directory for a backup chain for the database `neo4j`. If found, it is then aggregated into a single backup artifact.

An example of how to perform aggregation of a set of backups identified using a given backup file for the `neo4j` *database.*

```
bin/neo4j-admin database aggregate-backup --from-path=/mnt/backups/neo4j-2022-10-18T13-00-07.backup
```

The command checks the `/mnt/backups/` directory for a backup chain including the file *neo4j-2022-10-18T13-00-07.backup*, for the database `neo4j`. If found, it is then aggregated into a single backup artifact. This option is only available in Neo4j 5.2 and later.

Restore a database backup

Command

A database backup artifact 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 database restore` must be invoked as the `neo4j` user to ensure the appropriate file permissions. For more information, see Administrative commands.

> When restoring a backup chain, the transaction log contained in the differential backup artifacts need first to be replayed. This recovery operation is resource intensive and can be decoupled from the restore operation by using the aggregate command.

Syntax

```
neo4j-admin database restore    --from-path=<path>[,<path>...]
                                [--overwrite-destination=<true/false>]
                                [--restore-until=<recoveryCriteria>]
                                [--to-path-data=<path>]
                                [--to-path-txn=<path>]
                                [--verbose]
                                [--expand-commands]
                                [--force]
                                <database>
```

Options

Option	Default	Description
`--from-path`		A single path or a comma-separated list of paths pointing to a backup artifact file. An artifact file can be 1) a full backup, in which case it is restored directly or, 2) a differential backup, in which case the command tries first to find in the folder a backup chain ending at that specific differential backup and then restores that chain.
`--overwrite-destination`		Replace an existing database. This option is not safe on a cluster since clusters have additional state that would be inconsistent with restored database. In a cluster, restore to a new database to avoid this problem.

Option	Default	Description
`--restore-until`		Differential backup artifacts contains transaction logs that can be replayed and applied to stores contained in full backup artifacts when restoring a backup chain. The database applies logs until the recovery predicate is satisfied. Currently supported predicates are: `<transaction>` and `<date>`. • to restore a database up to a transaction id, the required transaction predicate should look like: `--restore-until=123`, where 123 is a provided transaction id. The restore recovers transaction logs up to - but not including - transaction 123. • to restore a database up to a specific date, the required date predicate should look like: `--restore-until=2021-09-11 10:15:30`, where n2021-09-11 10:15:30 is a UTC date. The restore recovers transactions that were committed before the provided date.
`--to-path-data`		Base directory for databases. Usage of this option is only allowed if the `--from-path` parameter points to one directory.
`--to-path-txn`		Base directory for transaction logs. Usage of this option is only allowed if the `--from-path` parameter points to one directory.
`--verbose`		Enable verbose output.

Option	Default	Description
`--expand-commands`		Allow command expansion in config value evaluation.
`--additional-config`		Configuration file to provide additional or override the existing configuration settings in the *neo4j.conf* file.

Parameters

Parameter	Default	Description
`<database>`		Name for the restored database.

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 database restore` command.

```
bin/neo4j-admin database restore --from-path=/mnt/backups/neo4j-2021-11-05T11-26-38.backup --overwrite
-destination neo4j
```

Unless you are replacing an existing database, you must create the database (using `CREATE DATABASE` against the `system` database) after the restore operation finishes.

If you have backed up a database with the option `--include-metadata`, you can manually restore the users and roles metadata.

From the *<neo4j-home>* directory, you run the Cypher script *data/scripts/databasename/restore_metadata.cypher*, which the `neo4j-admin database restore` command outputs, using Cypher Shell:

Using `cat` (UNIX)

```
cat data/scripts/databasename/restore_metadata.cypher | bin/cypher-shell -u user -p password -a
ip_address:port -d system --param "database => 'databasename'"
```

Using `type` (Windows)

```
type data\scripts\databasename\restore_metadata.cypher | bin\cypher-shell.bat -u user -p password -a
ip_address:port -d system --param "database => 'databasename'"
```

Back up an offline database

Remember to plan your backup carefully and to back up each of your databases, including the `system` database.

Command

A Neo4j database can be backed up in offline mode using the `dump` command of `neo4j-admin`. If the database is hosted in a cluster, make sure that the database is stopped on the server you are connected to.

Usage

The `neo4j-admin database dump` command can be used for performing a full backup of an **offline** database. It dumps a database into a single-file archive, called *<database>.dump*. Alternatively, `neo4j-admin database dump` can stream dump to standard output, enabling the output to be piped to another program, for example to `neo4j-admin database load`.

The command can be run only locally from an online or an offline Neo4j DBMS.

It does not support SSL/TLS.

Syntax

```
neo4j-admin database dump   [--verbose]
                            [--expand-commands]
                            [--overwrite-destination[=true|false]]
                            [--to-path=<path> | --to-stdout]
                            <database>
```

<database> — Name of the database to dump. Can contain `*` and `?` for globbing.

Options

Option	Default	Description
`--verbose`		Enable verbose output.
`--expand-commands`		Allow command expansion in config value evaluation.
`--overwrite-destination`	`false`	Overwrite any existing dump file in the destination folder.
`--to-path`		Destination for the database dump. It must point to an existing directory.

Option	Default	Description
`--to-stdout`		Use standard output as the destination for the database dump.

Example

The following is an example of how to create a dump of the default database `neo4j` using the `neo4j-admin database dump` command. The target directory */dumps/neo4j* must exist before running the command and the database must be offline.

```
bin/neo4j-admin database dump neo4j --to-path=/dumps/neo4j
```

`neo4j-admin database dump` cannot be applied to composite databases. It must be run directly on the databases that are part of a composite database.

Restore a database dump

A database dump can be loaded to a Neo4j instance using the `load` command of `neo4j-admin`.

Command

The `neo4j-admin database load` command loads a database from an archive created with the `neo4j-admin database dump` command. Alternatively, `neo4j-admin database load` can accept a dump from standard input, enabling it to accept input from `neo4j-admin database dump` or another source.

The command can be run from an online or an offline Neo4j DBMS.

If you are replacing an existing database, you have to shut it down before running the command. If you are not replacing an existing database, you must create the database (using `CREATE DATABASE` against the `system` database) after the load operation finishes.

`neo4j-admin database load` must be invoked as the `neo4j` user to ensure the appropriate file permissions.

Syntax

```
neo4j-admin database load   [-h]
                            [--verbose]
                            [--expand-commands]
                            [--info]
                            [--overwrite-destination[=true|false]]
                            [--from-path=<path> | --from-stdin]
                            [--additional-config=<file>]
                            <database>
```

<database> — Name of the database to load. Can contain `*` and `?` for globbing.

Options

Option	Default	Description
`--verbose`		Enable verbose output.
`--expand-commands`		Allow command expansion in config value evaluation.
`--info`		Print meta-data information about the archive file, such as file count, byte count, and format of the load file.
`--overwrite-destination`	`false`	If an existing database should be replaced.
`--from-path`		Path to the directory containing archive(s) created with the `neo4j-admin database dump` command.
`--from-stdin`		Read dump from standard input.
`--additional-config`	`<file>`	Configuration file with additional configuration.

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 database load` command. When replacing an existing database, you have to shut it down before running the command.

```
bin/neo4j-admin database load --from-path=/dumps/neo4j neo4j --overwrite-destination
```

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 cluster, and a previous version of the database exists, you must delete it (using `DROP DATABASE`) first. Alternatively, you can stop the Neo4j instance and unbind it from the cluster using `neo4j-admin server unbind` to remove its cluster state data. If you fail to DROP or unbind before loading the dump, that database's store files will be out of sync with its cluster state, potentially leading to logical corruptions. For more information, see Seed a cluster.

`neo4j-admin database load` cannot be applied to composite databases. It must be run directly on the databases that are part of a composite database.

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 database copy` is not supported for use on the `system` database.

`neo4j-admin database copy` is not supported for use on composite databases. It must be run directly on the databases that are part of a composite database.

It is important to note that `neo4j-admin database 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 database copy` command takes can be made based on the following:

- Neo4j, like many other databases, does IO in 8K pages.
- Your disc manufacturer will have a value for the maximum IOPS it can process.

For example, if your disc manufacturer has provided a maximum of 5000 IOPS, you can reasonably expect up to 5000 such page operations a second. Therefore, the maximal theoretical throughput you can expect is 40MB/s (or 144 GB/hour) on that disc. You may then assume that the best-case scenario for running `neo4j-admin database copy` on that 5000 IOPS disc is that it takes at least 1 hour to process a 144 GB database. [12]

However, it is important to remember that the process must read 144 GB from the source database, and must also write to the target store (assuming the target store is of comparable size). Additionally, there are internal processes during the copy that reads/modifies/writes the store multiple times. Therefore, with an additional 144 GB of both read and write, the best-case scenario for running `neo4j-admin database copy` on a 5000 IOPS disc is that it takes **at least 3 hours to process a 144 GB database.**

Finally, it is also important to consider that in almost all Cloud environments, the published IOPS value may not be the same as the actual value, or be able to continuously maintain the maximum possible IOPS. The real processing time for this example *could* be well above that estimation of 3 hours.

For detailed information about supported methods of upgrade and migration, see the Neo4j Upgrade and Migration Guide.

Command

`neo4j-admin database copy` copies the *data store* of an existing **offline** database to a new database.

Usage

The `neo4j-admin database copy` command can be used to clean up database inconsistencies, compact stores, and do a direct migration from Neo4j 4.4 to any 5.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 database copy` copies the *data store* and creates the node label and relationship type lookup indexes. Any additional *schema* (indexes and constraints) defined are not included. However, 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.

`neo4j-admin database copy` preserves the node IDs; however, the relationships get new IDs.

Syntax

```
neo4j-admin database copy   [--verbose]
                            [--from-path-data=<path> --from-path-txn=<path>]
                            [--to-path-data=<path> --to-path-txn=<path>]
                            [--to-path-schema=<path>]
                            [--force]
                            [--compact-node-store[=true|false]]
                            [--to-format=<format>]
                            [--ignore-nodes-with-labels=<label>[,<label>...]]
                            [--ignore-relationships-with-types=<type>[,<type>...]]
                            [--copy-only-node-properties=<label.property>[,<label.property>...]]
                            [--copy-only-nodes-with-labels=<label>[,<label>...]]
                            [--copy-only-relationship-
properties=<relationship.property>[,<relationship.property>...]]
                            [--copy-only-relationships-with-types=<type>[,<type>...]]
                            [--skip-labels=<label>[,<label>...]]
                            [--skip-node-properties=<label.property>[,<label.property>...]]
                            [--skip-properties=<property>[,<property>...]]
                            [--skip-relationship-
properties=<relationship.property>[,<relationship.property>...]]
                            [--from-pagecache=<size>]
                            <fromDatabase>
                            <toDatabase>
```

`<toDatabase>` — Name of the target database.

`<fromDatabase>` — Name of the source database.

Options

Option	Description
`--verbose`	Enable verbose output.
`--from-path-data`	Path to the */databases* directory, containing the directory of the source database. It can be used to target databases outside of the installation. Default: `server.directories.data`/databases
`--from-path-txn`	Path to the */transactions* directory, containing the transaction directory of the source database. Default: `server.directories.transaction.logs.root`
`--to-path-data=<path>`	Path to the */databases* directory, containing the directory of the target database. Default: `server.directories.data`/databases
`--to-path-txn`	Path to the */transactions* directory, containing the transaction directory of the target database. Default: `server.directories.transaction.logs.root`
`--to-path-schema`	Path to the directory where the schema commands file will be created. Default is the current directory.
`--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.

Option	Description
`--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` or `aligned`, there is no validation that the data will fit. Default: The format of the source database.
`--ignore-nodes-with-labels`	A comma-separated list of labels. Nodes that have *any* of the specified labels will not be included in the copy. Cannot be combined with `--copy-only-nodes-with-labels`.
`--ignore-relationships-with-types`	A comma-separated list of relationship types. Relationships with *any* of the specified relationship types will not be included in the copy. Cannot be combined with `--copy-only-relationships-with-types`.
`--copy-only-node-properties`	A comma-separated list of property keys to include in the copy for nodes with the specified label. Nodes whose labels are not explicitly mentioned in the list will have all their properties included in the copy. Cannot be combined with `--skip-properties` or `--skip-node-properties`.
`--copy-only-nodes-with-labels`	A comma-separated list of labels. All nodes that have *any* of the specified labels will be included in the copy. Cannot be combined with `--ignore-nodes-with-labels`.
`--copy-only-relationship-properties`	A comma-separated list of property keys to include in the copy for relationships with the specified type. Relationship types that are not explicitly mentioned will have all their properties included in the copy. Cannot be combined with `--skip-properties` or `--skip-relationship-properties`.

Option	Description
`--copy-only-relationships-with-types=<type>[,<type>…]`	A comma-separated list of relationship types. All relationships with *any* of the specified types will be included in the copy. Cannot be combined with `--ignore-relationships-with-types`.
`--skip-labels`	A comma-separated list of labels to ignore during the copy.
`--skip-node-properties`	A comma-separated list of property keys to ignore for nodes with the specified label. Cannot be combined with `--skip-properties` or `--copy-only-node-properties`.
`--skip-properties`	A comma-separated list of property keys to ignore during the copy. Cannot be combined with `--skip-node-properties`, `--copy-only-node-properties`, `--skip-relationship-properties`, and `--copy-only-relationship-properties`.
`--skip-relationship-properties`	A comma-separated list of property keys to ignore for relationships with the specified type. Cannot be combined with `--skip-properties` or `--copy-only-relationship-properties`.
`--from-pagecache`	The size of the page cache to use for reading.

You can use the `--from-pagecache` option to speed up the copy operation by specifying how much cache to allocate when reading the source. The `--from-pagecache` should be assigned whatever memory you can spare since Neo4j does random reads from the source.

Examples

Example 70. Use `neo4j-admin database 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 database copy neo4j 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 schema commands saved in the file *<database-name>-schema.cypher*.

   `--to-path-schema` can be used to specify a different directory for the schema file.

Example 71. Use `neo4j-admin database 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 database copy neo4j copy --ignore-nodes-with-labels="Cat,Dog"
```

The command creates a copy of the database `neo4j` but without the nodes with the labels `:Cat` and `:Dog`.

Labels are processed independently, i.e., the filter ignores any node with a label `:Cat`, `:Dog`, or both.

For a detailed example of how to use `neo4j-admin database copy` to filter out data for sharding a database, see Sharding data with the `copy` command.

[12] The calculations are based on `MB/s = (IOPS * B) ÷ 10^6`, where `B` is the block size in bytes; in the case of Neo4j, this is `8000`. GB/hour can then be calculated from `(MB/s * 3600) ÷ 1000`.

Authentication and authorization

This section helps you ensure that your Neo4j deployment adheres to your company's information security guidelines by setting up the appropriate authentication and authorization rules.

The following topics are covered:

- Introduction
- Built-in roles
- Fine-grained access control
- Integration with LDAP directory services
- Integration with Single Sign-On (SSO) services
- Manage procedure and user-defined function permissions
- Terminology

> The functionality described in this section is applicable to Enterprise Edition. A limited set of user management functions are also available in Community Edition. Native roles overview gives a quick overview of these.

Introduction

This page provides an overview of authentication and authorization in Neo4j. Authorization is managed using role-based access control (RBAC). Permissions that define access control are assigned to roles, which are in turn assigned to users.

Neo4j has the following auth providers, that can perform user authentication and authorization:

Native auth provider

Neo4j provides a native auth provider that stores user and role information in the `system` database. The following parameters control this provider:

* `dbms.security.auth_enabled` (Default: `true`) — Enable auth requirement to access Neo4j.

  > If you need to disable authentication, for example, to recover an `admin` user password or assign a user to the `admin` role, make sure you block all network connections during the recovery phase so users can connect to Neo4j only via `localhost`. For more information, see Password and user recovery.

* `dbms.security.auth_lock_time` (Default: `5s`) — The amount of time a user account is locked after a configured number of unsuccessful authentication attempts.
* `dbms.security.auth_max_failed_attempts` (Default: `3`) — The maximum number of unsuccessful authentication attempts before imposing a user lock for a configured amount of time.
  When triggered, Neo4j logs an error containing a timestamp and the message `failed to log in: too many failed attempts` in the *security.log*.

The Cypher commands to manage users, roles, and permissions are described in detail in Cypher Manual → 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.

Single Sign-On provider

Some clients wish to centrally manage authentication and authorization for their systems in an identity provider which provides single sign-on for their other systems. Neo4j supports the popular OpenID Connect mechanism for integrating with identity providers and configuration for that is described in Integration with Single Sign-On Services.

Custom-built plugin auth providers

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

Kerberos authentication and single sign-on

In addition to LDAP, Native and custom providers, Neo4j supports Kerberos for authentication and single sign-on. Kerberos support is provided via the Neo4j Kerberos Add-On.

Built-in roles

Neo4j provides built-in roles with default privileges. The built-in roles and the default privileges are:

`PUBLIC`

* Access to the home database.

* Allows executing procedures with the users own privileges.
* Allows executing user-defined functions with the users own privileges.

`reader`

* Access to all databases.
* Traverse and read on the data graph (all nodes, relationships, properties).
* Show indexes and constraints along with any other future schema constructs.

`editor`

* Access to all databases.
* Traverse, read, and write on the data graph.
* Write access limited to creating and changing **existing** property keys, node labels, and relationship types of the graph. In other words, the `editor` role cannot add to the schema but can only make changes to already existing objects.
* Show indexes and constraints along with any other future schema constructs.

`publisher`

* Access to all databases.
* Traverse, read, and write on the data graph.
* Show indexes and constraints along with any other future schema constructs.

`architect`

* Access to all databases.
* Traverse, read, and write on the data graph.
* Create/drop/show indexes and constraints along with any other future schema constructs.

`admin`

* Access to all databases.
* Traverse, read, and write on the data graph.
* 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.

ℹ	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	✓	✓	✓	✓	✓	✓	✓
View own details	✓	✓	✓	✓	✓	✓	✓
View own transactions	✓	✓	✓	✓	✓	✓	✓
Terminate own transactions	✓	✓	✓	✓	✓	✓	✓
View own privileges	✓	✓	✓	✓	✓	✓	
View all databases	✓	✓	✓	✓	✓	✓	✓
Access home database	✓	✓	✓	✓	✓	✓	✓
Access all databases	✓	✓	✓	✓	✓		✓
Read data	✓	✓	✓	✓	✓		✓
View index/constraint	✓	✓	✓	✓	✓		✓
Write/update/delete existing data		✓	✓	✓	✓		✓
Create new types of properties key			✓	✓	✓		✓
Create new types of nodes labels			✓	✓	✓		✓

Action	`reader`	`editor`	`publisher`	`architect`	`admin`	`PUBLIC`	Available in Community Edition
Create new types of relationship types			✓	✓	✓		✓
Create/drop index/constraint				✓	✓		✓
Create/delete user					✓		✓
Change another user’s name					✓		✓
Change another user’s password					✓		✓
Change another user’s home database					✓		
Suspend/activate user					✓		
Create/drop roles					✓		
Change role names					✓		
Assign/remove role to/from user					✓		
Create/drop/alter databases					✓		
Start/stop databases					✓		
Manage database access					✓		
Grant/deny/revoke privileges					✓		
View all users					✓		✓
View all roles					✓		
View all roles for a user					✓		
View all users for a role					✓		
View another user’s privileges					✓		

Action	`reader`	`editor`	`publisher`	`architect`	`admin`	`PUBLIC`	Available in Community Edition
View all transactions					✓		✓
Terminate all transactions					✓		✓
Execute procedures					✓	✓	✓
Execute functions					✓	✓	✓
Execute admin procedures					✓		✓
Dynamically change configuration [13]					✓		

More information about the built-in roles and their privileges can be found in Neo4j Cypher Manual.

Fine-grained access control

When creating a database, administrators may want to establish which users have the ability to access certain information.

As described in Built-in roles, Neo4j already offers preset roles configured to specific permissions (i.e. read, edit, or write). While these built-in roles cover many common daily scenarios, it is also possible to create custom roles for specific needs.

This page contains an example that illustrates various aspects of security and fine-grained access control.

Healthcare use case

To demonstrate the application of these tools, consider an example of a *healthcare* database which could be relevant in a medical clinic or hospital.

For simplicity reasons, only three labels are used to represent the following entities:

`(:Patient)`
: Patients that visit the clinic because they have some symptoms. Information specific to patients can be captured in properties:

- `name`
- `ssn`
- `address`
- `dateOfBirth`

`(:Symptom)`

A set of symptoms found in a catalog of known illnesses. They can be described using the properties:

- `name`
- `description`

`(:Disease)`

Known illnesses mapped in a catalog found in the database. They can be described using the properties:

- `name`
- `description`

These entities are modelled as nodes, and connected by relationships of the following types:

`(:Patient)-[:HAS]→(:Symptom)`

When a patient reports to the clinic, they describe their symptoms to the nurse or the doctor. The nurse or doctor then enters this information into the database in the form of connections between the patient node and a graph of known symptoms. Possible properties of interest on this relationship could be:

- `date` - date when symptom was reported.

`(:Symptom)-[:OF]→(:Disease)`

Symptoms are a subgraph in the graph of known diseases. The relationship between a symptom and a disease can include a probability factor for how likely or common it is for people with that disease to express that symptom. This will make it easier for the doctor to make a diagnosis using statistical queries.

- `probability` - probability of symptom matching disease.

`(:Patient)-[:DIAGNOSIS]→(:Disease)`

The doctor can use the graph of diseases and their symptoms to perform an initial investigation into the most likely diseases to match the patient. Based on this, and their own assessment of the patient, the doctor may make a diagnosis which they would persist to the graph through the addition of this relationship with appropriate properties:

- `by`: doctor’s name
- `date`: date of diagnosis
- `description`: additional doctors' notes

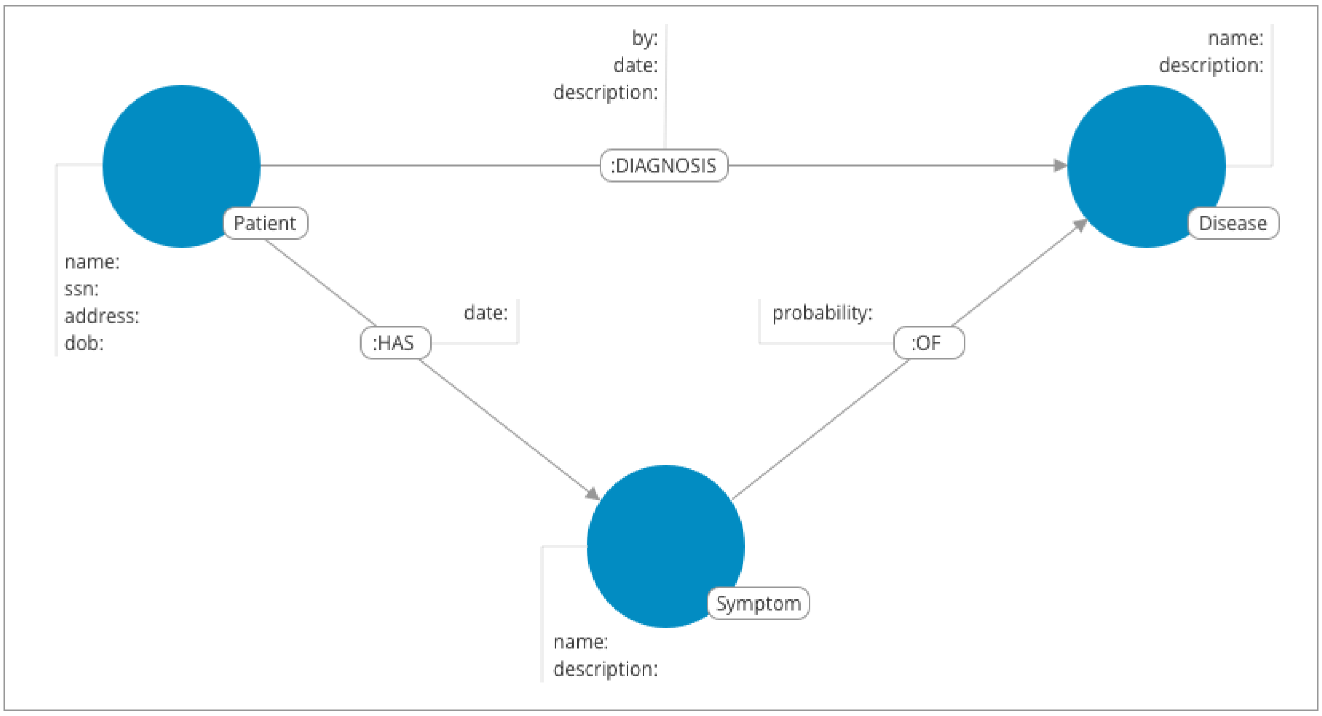


Figure 7. Healthcare use case

This same database would be used by a number of different users, each with different access needs:

- **Doctors** who need to diagnose patients.
- **Nurses** who need to treat patients.
- **Receptionists** who need to identify and record patient information.
- **Researchers** who need to perform statistical analysis of medical data.
- **IT administrators** who need to manage the database, to create and assign users for example.

Security

Unlike applications which often require users to be modeled within the application itself, databases provide user management resources such as roles and privileges. This allows users to be created entirely within the database security model, a strategy that allows the separation of access to the data and the data itself. For more information, see Cypher Manual → Access control.

The following examples show two different approaches to using Neo4j security features to support the *healthcare* database application. The first approach uses Built-in roles, whereas the second uses more advanced resources with fine-grained privileges for sub-graph access control.

In this example, consider five users of the *healthcare* database:

- Alice, the doctor.
- Daniel, the nurse.
- Bob, the receptionist.
- Charlie, the researcher.
- Tina, the IT administrator.

These users can be created using the `CREATE USER` command (from the `system` database):

Example 72. 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 these capabilities need to be granted by using roles. There are two different ways of doing this, either by using the built-in roles, or through more fine-grained access control using privileges and custom roles.

Access control using built-in roles

Neo4j comes with built-in roles that cover a number of common needs:

- `PUBLIC` - All users have this role, can by default access the home database, and run all procedures and user-defined functions.
- `reader` - Can read data from all databases.
- `editor` - Can read and update all databases, but not expand the schema with new labels, relationship types or property names.
- `publisher` - Can read and edit, as well as add new labels, relationship types, and property names.
- `architect` - Has all the capabilities of the publisher as well as the ability to manage indexes and constraints.
- `admin` - Can perform architect actions as well as manage databases, users, roles, and privileges.

Consider Charlie from the example of users. As a researcher, they do not need write access to the database, so they are assigned the `reader` role.

On the other hand, Alice (the doctor), Daniel (the nurse), and Bob (the receptionist) all need to update the database with new patient information, but do not need to expand the schema with new labels, relationship types, property names or indexes. For this reason, they are all assigned the `editor` role.

Tina, the IT administrator who installs and manages the database, needs to be assigned the `admin` role.

Here is how to grant roles to the users:

Example 73. 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;
```

Sub-graph access control using privileges]

A limitation of the previously described approach is that it does allow all users to see all the data on the database. In many real-world scenarios though, it would be preferable to establish some access restrictions.

For example, you may want to limit the researcher’s access to the patients' personal information or restrict the receptionist from writing new labels on the database. While these restrictions could be coded into the application layer, it is possible and **more secure** to enforce fine-grained restrictions directly within the Neo4j security model by creating custom roles and assigning specific privileges to them.

Since new custom roles will be created, it is important to first revoke the current roles from the users assigned to them:

```
REVOKE ROLE reader FROM charlie;
REVOKE ROLE editor FROM alice;
REVOKE ROLE editor FROM daniel;
REVOKE ROLE editor FROM bob;
REVOKE ROLE admin FROM tina;
```

Now you can create custom roles based on the concept of *privileges*, which allows more control over what each user is capable of doing. To properly assign those privileges, start by identifying each type of user:

Doctor

Should be able to read and write most of the graph, but be prevented from reading the patients' address. Has the permission to save *diagnoses* to the database, but not expand the schema with new concepts.

Receptionist

Should be able to read and write all patient data, but not be able to see the symptoms, diseases, or diagnoses.

Researcher

Should be able to perform statistical analysis of all data, except patients' personal information, to which they should have restricted access. To illustrate two different ways of setting up the same effective privileges, two roles are created for comparison.

Nurse

Should be able to perform all tasks that both the doctor and the receptionist can do. Granting both roles (doctor and receptionist) to the nurse does not work as expected. This is explained in the section dedicated to the creation of the `nurse` role.

Junior nurse

While the senior nurse is able to save diagnoses just as a doctor can, some (junior) nurses might not be allowed to do that. Creating another role from scratch is an option, but the same output can be achieved by combining the `nurse` role with a new `disableDiagnoses` role that specifically restricts that activity.

IT administrator

This role is very similar to the built-in `admin` role, except that it should not allow access to the patients' `SSN` or be able to save a diagnosis, a privilege restricted to medical professionals. To achieve this, the built-in `admin` role can be copied and modified accordingly.

User manager

This user should have similar access as the IT administrator, but with more restrictions. To achieve that, a new role can be created from scratch and only specific administrative capabilities can be assigned to it.

Before creating the new roles and assigning them to Alice, Bob, Daniel, Charlie, and Tina, it is important to define the privileges each role should have. Since all users need `ACCESS` privilege to the `healthcare` database, this can be set through the `PUBLIC` role instead of all the individual roles:

```
GRANT ACCESS ON DATABASE healthcare TO PUBLIC;
```

=== Privileges of `itadmin`

This role can be created as a copy of the built-in `admin` role:

```
CREATE ROLE itadmin AS COPY OF admin;
```

Then you need to **deny** the two specific actions this role is not supposed to perform:

- Read any patients' social security number (`SSN`).
- Submit medical diagnoses.

```
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 AS COMMANDS;
```

```
+---------------------------------------------------------------------------+
| command                                                                   |
+---------------------------------------------------------------------------+
| "GRANT ACCESS ON DATABASE * TO `itadmin`"                                 |
| "GRANT MATCH {*} ON GRAPH * NODE * TO `itadmin`"                          |
| "GRANT MATCH {*} ON GRAPH * RELATIONSHIP * TO `itadmin`"                  |
| "GRANT WRITE ON GRAPH * TO `itadmin`"                                     |
| "GRANT INDEX MANAGEMENT ON DATABASE * TO `itadmin`"                       |
| "GRANT CONSTRAINT MANAGEMENT ON DATABASE * TO `itadmin`"                  |
| "GRANT NAME MANAGEMENT ON DATABASE * TO `itadmin`"                        |
| "GRANT START ON DATABASE * TO `itadmin`"                                  |
| "GRANT STOP ON DATABASE * TO `itadmin`"                                   |
| "GRANT TRANSACTION MANAGEMENT (*) ON DATABASE * TO `itadmin`"             |
| "GRANT ALL DBMS PRIVILEGES ON DBMS TO `itadmin`"                          |
| "DENY READ {ssn} ON GRAPH `healthcare` NODE Patient TO `itadmin`"         |
| "DENY CREATE ON GRAPH `healthcare` RELATIONSHIP DIAGNOSIS TO `itadmin`"   |
+---------------------------------------------------------------------------+
```

Privileges that were granted or denied earlier can be revoked using the `REVOKE` command.

To provide the IT administrator `tina` these privileges, they must be assigned the new role `itadmin`:

```
neo4j@system> GRANT ROLE itadmin TO tina;
```

To demonstrate that Tina is not able to see the patients' `SSN`, you can login to `healthcare` as `tina` and run the following query:

```
MATCH (n:Patient)
 WHERE n.dateOfBirth < date('1972-06-12')
RETURN n.name, n.ssn, n.address, n.dateOfBirth;
```

```
+---------------------------------------------------------------------+
| n.name          | n.ssn | n.address                | n.dateOfBirth |
+---------------------------------------------------------------------+
| "Mary Stone"    | NULL  | "1 secret way, downtown" | 1970-01-15    |
| "Ally Anderson" | NULL  | "1 secret way, downtown" | 1970-08-20    |
| "Sally Stone"   | NULL  | "1 secret way, downtown" | 1970-03-12    |
| "Jane Stone"    | NULL  | "1 secret way, downtown" | 1970-07-21    |
| "Ally Svensson" | NULL  | "1 secret way, downtown" | 1971-08-15    |
| "Jane Svensson" | NULL  | "1 secret way, downtown" | 1972-05-12    |
| "Ally Svensson" | NULL  | "1 secret way, downtown" | 1971-07-30    |
+---------------------------------------------------------------------+
```

The results make it seem as if these nodes do not even have an `SSN` field. This is a key feature of the security model: users cannot tell the difference between data that does not exist and data that is hidden using fine-grained read privileges.

Now recall that the `itadmin` role was denied the ability to save diagnoses (as this is a critical medical function reserved for only doctors and senior medical staff), you can test this by trying to create `DIAGNOSIS` relationships:

```
MATCH (n:Patient), (d:Disease)
CREATE (n)-[:DIAGNOSIS]->(d);
```

```
Create relationship with type 'DIAGNOSIS' is not allowed for user 'tina' with roles [PUBLIC,
itadmin].
```

Restrictions to reading data do not result in errors, they only make it appear as if the data is not there. However, restrictions on updating the graph do output an appropriate error when the user attempts to perform an action they are not allowed to.

=== Privileges of `researcher`

The researcher Charlie was previously a read-only user. To assign them the desired permissions, you can do something similar to what was done with the `itadmin` role, this time copying and modifying the `reader` role.

Another way to do it is by creating a new role from scratch and then either granting or denying a list of privileges:

- **Denying privileges:**

  You can grant the role `researcher` the ability to find all nodes and read all properties (much like the `reader` role), but deny read access to the `Patient` properties. This way, the researcher is unable to see patients' information such as `name`, `SSN`, and `address`. This approach has a problem though: if more properties are added to the `Patient` nodes *after* the restrictions were assigned to the `researcher` role, these new properties will automatically be visible to the researcher — a possibly undesirable outcome.

  To avoid that, you can rather deny *specific* privileges:

  ```
  // First create the role
  CREATE ROLE researcherB;
  // Then grant access to everything
  GRANT MATCH {*}
      ON GRAPH healthcare
      TO researcherB;
  // And deny read on specific node properties
  DENY READ {name, address, ssn}
      ON GRAPH healthcare
      NODES Patient
      TO researcherB;
  // And finally deny traversal of the doctors diagnosis
  DENY TRAVERSE
      ON GRAPH healthcare
      RELATIONSHIPS DIAGNOSIS
      TO researcherB;
  ```

- **Granting privileges:**

  Another alternative is to only provide specific access to the properties the researcher is allowed to see. This way, the addition of new properties (for instance, to a `Patient` node) does not automatically make them visible to users assigned with this role. In case you wish to make them visible though, you need to explicitly grant read access:

```
// Create the role first
CREATE ROLE researcherW
// Allow the researcher to find all nodes
GRANT TRAVERSE
    ON GRAPH healthcare
    NODES *
    TO researcherW;
// Now only allow the researcher to traverse specific relationships
GRANT TRAVERSE
    ON GRAPH healthcare
    RELATIONSHIPS HAS, OF
    TO researcherW;
// Allow reading of all properties of medical metadata
GRANT READ {*}
    ON GRAPH healthcare
    NODES Symptom, Disease
    TO researcherW;
// Allow reading of all properties of the disease-symptom relationship
GRANT READ {*}
    ON GRAPH healthcare
    RELATIONSHIPS OF
    TO researcherW;
// Only allow reading dateOfBirth for research purposes
GRANT READ {dateOfBirth}
    ON GRAPH healthcare
    NODES Patient
    TO researcherW;
```

In order to test that the researcher Charlie now has the specified privileges, assign them the `researcherB` role (with specifically denied privileges):

```
GRANT ROLE researcherB TO charlie;
```

You can also use a version of the `SHOW PRIVILEGES` command to see Charlie's access rights, which are a combination of those assigned to the `researcherB` and `PUBLIC` roles:

```
neo4j@system> SHOW USER charlie PRIVILEGES AS COMMANDS;
```

```
+-----------------------------------------------------------------------+
| command                                                               |
+-----------------------------------------------------------------------+
| "GRANT ACCESS ON HOME DATABASE TO $role"                              |
| "GRANT ACCESS ON DATABASE `healthcare` TO $role"                      |
| "GRANT EXECUTE PROCEDURE * ON DBMS TO $role"                          |
| "GRANT EXECUTE FUNCTION * ON DBMS TO $role"                           |
| "GRANT MATCH {*} ON GRAPH `healthcare` NODE * TO $role"               |
| "GRANT MATCH {*} ON GRAPH `healthcare` RELATIONSHIP * TO $role"       |
| "DENY TRAVERSE ON GRAPH `healthcare` RELATIONSHIP DIAGNOSIS TO $role" |
| "DENY READ {address} ON GRAPH `healthcare` NODE Patient TO $role"     |
| "DENY READ {name} ON GRAPH `healthcare` NODE Patient TO $role"        |
| "DENY READ {ssn} ON GRAPH `healthcare` NODE Patient TO $role"         |
+-----------------------------------------------------------------------+
```

Now when Charlie logs into the `healthcare` database and tries to run a command similar to the one previously used by the `itadmin`, they will see different results:

```
MATCH (n:Patient)
 WHERE n.dateOfBirth < date('1972-06-12')
RETURN n.name, n.ssn, n.address, n.dateOfBirth;
```

```
+---------------------------------------------+
| n.name | n.ssn | n.address | n.dateOfBirth |
+---------------------------------------------+
| NULL   | NULL  | NULL      | 1971-05-31    |
| NULL   | NULL  | NULL      | 1971-04-17    |
| NULL   | NULL  | NULL      | 1971-12-27    |
| NULL   | NULL  | NULL      | 1970-02-13    |
| NULL   | NULL  | NULL      | 1971-02-04    |
| NULL   | NULL  | NULL      | 1971-05-10    |
| NULL   | NULL  | NULL      | 1971-02-21    |
+---------------------------------------------+
```

Only the date of birth is available, so that the researcher Charlie may perform statistical analysis, for example. Another query Charlie could try is to find the ten diseases a patient younger than 25 is most likely to be diagnosed with, listed by probability:

```
WITH datetime() - duration({years:25}) AS timeLimit
MATCH (n:Patient)
WHERE n.dateOfBirth > date(timeLimit)
MATCH (n)-[h:HAS]->(s:Symptom)-[o:OF]->(d:Disease)
WITH d.name AS disease, o.probability AS prob
RETURN disease, sum(prob) AS score ORDER BY score DESC LIMIT 10;
```

```
+-------------------------------------------+
| disease              | score              |
+-------------------------------------------+
| "Acute Argitis"      | 95.05395287286318 |
| "Chronic Someitis"   | 88.7220337139605  |
| "Chronic Placeboitis" | 88.43609533058974 |
| "Acute Whatitis"     | 83.23493746472457 |
| "Acute Otheritis"    | 82.46129768949129 |
| "Chronic Otheritis"  | 82.03650063794025 |
| "Acute Placeboitis"  | 77.34207326583929 |
| "Acute Yellowitis"   | 76.34519967465832 |
| "Chronic Whatitis"   | 73.73968070128234 |
| "Chronic Yellowitis" | 71.58791287376775 |
+-------------------------------------------+
```

If the `researcherB` role is revoked to Charlie, but `researcherW` is granted, when re-running these queries, the same results will be obtained.

Privileges that were granted or denied earlier can be revoked using the `REVOKE` command.

=== Privileges of `doctor`

Doctors should be given the ability to read and write almost everything, except the patients' `address` property, for instance. This role can be built from scratch by assigning full read and write access, and then specifically denying access to the `address` property:

```
CREATE ROLE doctor;
GRANT TRAVERSE ON GRAPH healthcare TO doctor;
GRANT READ {*} ON GRAPH healthcare TO doctor;
GRANT WRITE ON GRAPH healthcare TO doctor;
DENY READ {address} ON GRAPH healthcare NODES Patient TO doctor;
DENY SET PROPERTY {address} ON GRAPH healthcare NODES Patient TO doctor;
```

To allow the doctor Alice to have these privileges, grant them this new role:

```
neo4j@system> GRANT ROLE doctor TO alice;
```

To demonstrate that Alice is not able to see patient addresses, log in as `alice` to `healthcare` and run the following query:

```
MATCH (n:Patient)
 WHERE n.dateOfBirth < date('1972-06-12')
RETURN n.name, n.ssn, n.address, n.dateOfBirth;
```

```
+---------------------------------------------------------+
| n.name          | n.ssn   | n.address | n.dateOfBirth |
+---------------------------------------------------------+
| "Jack Anderson" | 1234647 | NULL      | 1970-07-23    |
| "Joe Svensson"  | 1234659 | NULL      | 1972-06-07    |
| "Mary Jackson"  | 1234568 | NULL      | 1971-10-19    |
| "Jack Jackson"  | 1234583 | NULL      | 1971-05-04    |
| "Ally Smith"    | 1234590 | NULL      | 1971-12-07    |
| "Ally Stone"    | 1234606 | NULL      | 1970-03-29    |
| "Mark Smith"    | 1234610 | NULL      | 1971-03-30    |
+---------------------------------------------------------+
```

As a result, the doctor has the expected privileges, including being able to see the patients' `SSN`, but not their address.

The doctor is also able to see all other node types:

```
MATCH (n) WITH labels(n) AS labels
RETURN labels, count(*);
```

```
+------------------------+
| labels      | count(*) |
+------------------------+
| ["Patient"] | 101      |
| ["Symptom"] | 10       |
| ["Disease"] | 12       |
+------------------------+
```

In addition, the doctor can traverse the graph, finding symptoms and diseases connected to patients:

```
MATCH (n:Patient)-[:HAS]->(s:Symptom)-[:OF]->(d:Disease)
  WHERE n.ssn = 1234657
RETURN n.name, d.name, count(s) AS score ORDER BY score DESC;
```

The resulting table shows which are the most likely diagnoses based on symptoms. The doctor can use this table to facilitate further questioning and testing of the patient in order to decide on the final diagnosis.

```
+---------------------------------------------------+
| n.name          | d.name                | score |
+---------------------------------------------------+
| "Sally Anderson" | "Chronic Otheritis"   | 4     |
| "Sally Anderson" | "Chronic Yellowitis"  | 3     |
| "Sally Anderson" | "Chronic Placeboitis" | 3     |
| "Sally Anderson" | "Acute Whatitis"      | 2     |
| "Sally Anderson" | "Acute Yellowitis"    | 2     |
| "Sally Anderson" | "Chronic Someitis"    | 2     |
| "Sally Anderson" | "Chronic Argitis"     | 2     |
| "Sally Anderson" | "Chronic Whatitis"    | 2     |
| "Sally Anderson" | "Acute Someitis"      | 1     |
| "Sally Anderson" | "Acute Argitis"       | 1     |
| "Sally Anderson" | "Acute Otheritis"     | 1     |
+---------------------------------------------------+
```

Once the doctor has investigated further, they would be able to decide on the diagnosis and save that result to the database:

```
WITH datetime({epochmillis:timestamp()}) AS now
WITH now, date(now) as today
MATCH (p:Patient)
  WHERE p.ssn = 1234657
MATCH (d:Disease)
  WHERE d.name = "Chronic Placeboitis"
MERGE (p)-[i:DIAGNOSIS {by: 'Alice'}]->(d)
  ON CREATE SET i.created_at = now, i.updated_at = now, i.date = today
  ON MATCH SET i.updated_at = now
RETURN p.name, d.name, i.by, i.date, duration.between(i.created_at, i.updated_at) AS updated;
```

This allows the doctor to record their diagnosis as well as take note of previous diagnoses:

```
+-----------------------------------------------------------------------------------------+
| p.name          | d.name                | i.by    | i.date     | updated               |
+-----------------------------------------------------------------------------------------+
| "Sally Anderson" | "Chronic Placeboitis" | "Alice" | 2020-05-29 | P0M0DT213.076000000S |
+-----------------------------------------------------------------------------------------+
```

Creating the `DIAGNOSIS` relationship for the first time requires the privilege to create new types. This is also true for the property names `doctor`, `created_at`, and `updated_at`. It can be fixed by either granting the doctor `NAME MANAGEMENT` privileges or by pre-creating the missing types. The latter would be more precise and can be achieved by running, as an administrator, the procedures `db.createRelationshipType` and `db.createProperty` with appropriate arguments.

=== Privileges of `receptionist`

Receptionists should only be able to manage patient information. They are not allowed to find or read any other parts of the graph. In addition, they should be able to create and delete patients, but not any other nodes:

```
CREATE ROLE receptionist;
GRANT MATCH {*} ON GRAPH healthcare NODES Patient TO receptionist;
GRANT CREATE ON GRAPH healthcare NODES Patient TO receptionist;
GRANT DELETE ON GRAPH healthcare NODES Patient TO receptionist;
GRANT SET PROPERTY {*} ON GRAPH healthcare NODES Patient TO receptionist;
```

It would have been simpler to grant global `WRITE` privileges to the receptionist Bob. However, this would have the unfortunate side effect of allowing them the ability to create other nodes, like new `Symptom` nodes, even though they would subsequently be unable to find or read those same nodes. While there are use cases in which it is desirable to have roles able to create data they cannot read, that is not the case of this model.

With that in mind, grant the receptionist Bob their new `receptionist` role:

```
neo4j@system> GRANT ROLE receptionist TO bob;
```

With these privileges, if Bob tries to read the entire database, they will still only see the patients:

```
MATCH (n) WITH labels(n) AS labels
RETURN labels, count(*);
```

```
+------------------------+
| labels      | count(*) |
+------------------------+
| ["Patient"] | 101      |
+------------------------+
```

However, Bob is able to see all fields of the patients' records:

```
MATCH (n:Patient)
 WHERE n.dateOfBirth < date('1972-06-12')
RETURN n.name, n.ssn, n.address, n.dateOfBirth;
```

```
+-----------------------------------------------------------------------+
| n.name          | n.ssn   | n.address                | n.dateOfBirth |
+-----------------------------------------------------------------------+
| "Mark Stone"    | 1234666 | "1 secret way, downtown" | 1970-08-04    |
| "Sally Jackson" | 1234633 | "1 secret way, downtown" | 1970-10-21    |
| "Bob Stone"     | 1234581 | "1 secret way, downtown" | 1972-02-16    |
| "Ally Anderson" | 1234582 | "1 secret way, downtown" | 1970-05-13    |
| "Mark Svensson" | 1234594 | "1 secret way, downtown" | 1970-01-16    |
| "Bob Anderson"  | 1234597 | "1 secret way, downtown" | 1970-09-23    |
| "Jack Svensson" | 1234599 | "1 secret way, downtown" | 1971-02-13    |
| "Mark Jackson"  | 1234618 | "1 secret way, downtown" | 1970-03-28    |
| "Jack Jackson"  | 1234623 | "1 secret way, downtown" | 1971-04-02    |
+-----------------------------------------------------------------------+
```

With the `receptionist` role, Bob can delete any new patient nodes they have just created, but they are not able to delete patients that have already received diagnoses since those are connected to parts of the graph that Bob cannot see. Here is a demonstration of both scenarios:

```
CREATE (n:Patient {
  ssn:87654321,
  name: 'Another Patient',
  email: 'another@example.com',
  address: '1 secret way, downtown',
  dateOfBirth: date('2001-01-20')
})
RETURN n.name, n.dateOfBirth;
```

```
+-----------------------------------+
| n.name            | n.dateOfBirth |
+-----------------------------------+
| "Another Patient" | 2001-01-20    |
+-----------------------------------+
```

The receptionist is able to modify any patient record:

```
MATCH (n:Patient)
WHERE n.ssn = 87654321
SET n.address = '2 streets down, uptown'
RETURN n.name, n.dateOfBirth, n.address;
```

```
+-----------------------------------------------------------------+
| n.name            | n.dateOfBirth | n.address                 |
+-----------------------------------------------------------------+
| "Another Patient" | 2001-01-20    | "2 streets down, uptown" |
+-----------------------------------------------------------------+
```

The receptionist is also able to delete this recently created patient because it is not connected to any other records:

```
MATCH (n:Patient)
 WHERE n.ssn = 87654321
DETACH DELETE n;
```

However, if the receptionist attempts to delete a patient that has existing diagnoses, this will fail:

```
MATCH (n:Patient)
 WHERE n.ssn = 1234610
DETACH DELETE n;
```

```
org.neo4j.graphdb.ConstraintViolationException: Cannot delete node<42>, because it still has
relationships. To delete this node, you must first delete its relationships.
```

The reason why this query fails is that, while Bob can find the `(:Patient)` node, they do not have sufficient traverse rights to find nor delete the outgoing relationships from it. Either they need to ask Tina the `itadmin` for help for this task, or more privileges can be added to the `receptionist` role:

```
GRANT TRAVERSE ON GRAPH healthcare NODES Symptom, Disease TO receptionist;
GRANT TRAVERSE ON GRAPH healthcare RELATIONSHIPS HAS, DIAGNOSIS TO receptionist;
GRANT DELETE ON GRAPH healthcare RELATIONSHIPS HAS, DIAGNOSIS TO receptionist;
```

Privileges that were granted or denied earlier can be revoked using the `REVOKE` command.

=== Privileges of nurses

Nurses should have the capabilities of both doctors and receptionists, but assigning them both the `doctor` and `receptionist` roles might not have the expected effect. If those two roles were created with `GRANT` privileges only, combining them would be simply cumulative. But if the `doctor` role contains some `DENY` privileges, these always overrule `GRANT`. This means that the nurse will still have the same restrictions as a doctor, which is not what is intended here.

To demonstrate this, you can assign the `doctor` role to the nurse Daniel:

```
neo4j@system> GRANT ROLE doctor, receptionist TO daniel;
```

Daniel should now have a combined set of privileges:

```
SHOW USER daniel PRIVILEGES AS COMMANDS;
```

```
+--------------------------------------------------------------------------+
| command                                                                  |
+--------------------------------------------------------------------------+
| "GRANT ACCESS ON HOME DATABASE TO $role"                                 |
| "GRANT ACCESS ON DATABASE `healthcare` TO $role"                         |
| "GRANT EXECUTE PROCEDURE * ON DBMS TO $role"                             |
| "GRANT EXECUTE FUNCTION * ON DBMS TO $role"                              |
| "GRANT TRAVERSE ON GRAPH `healthcare` NODE * TO $role"                   |
| "GRANT TRAVERSE ON GRAPH `healthcare` RELATIONSHIP * TO $role"           |
| "GRANT READ {*} ON GRAPH `healthcare` NODE * TO $role"                   |
| "GRANT READ {*} ON GRAPH `healthcare` RELATIONSHIP * TO $role"           |
| "GRANT MATCH {*} ON GRAPH `healthcare` NODE Patient TO $role"            |
| "GRANT WRITE ON GRAPH `healthcare` TO $role"                             |
| "GRANT SET PROPERTY {*} ON GRAPH `healthcare` NODE Patient TO $role"     |
| "GRANT CREATE ON GRAPH `healthcare` NODE Patient TO $role"               |
| "GRANT DELETE ON GRAPH `healthcare` NODE Patient TO $role"               |
| "DENY READ {address} ON GRAPH `healthcare` NODE Patient TO $role"        |
| "DENY SET PROPERTY {address} ON GRAPH `healthcare` NODE Patient TO $role" |
+--------------------------------------------------------------------------+
```

Privileges that were granted or denied earlier can be revoked using the `REVOKE` command.

Now the intention is that a nurse can perform the actions of a receptionist, which means they should be able to read and write the `address` field of the `Patient` nodes. To do so, the nurse can run the following query:

```
MATCH (n:Patient)
 WHERE n.dateOfBirth < date('1972-06-12')
RETURN n.name, n.ssn, n.address, n.dateOfBirth;
```

Which returns these results:

```
+---------------------------------------------------------+
| n.name          | n.ssn   | n.address | n.dateOfBirth |
+---------------------------------------------------------+
| "Jane Anderson" | 1234572 | NULL      | 1971-05-26    |
| "Mark Stone"    | 1234586 | NULL      | 1972-06-07    |
| "Joe Smith"     | 1234595 | NULL      | 1970-12-28    |
| "Joe Jackson"   | 1234603 | NULL      | 1970-08-31    |
| "Jane Jackson"  | 1234628 | NULL      | 1972-01-31    |
| "Mary Anderson" | 1234632 | NULL      | 1971-01-07    |
| "Jack Svensson" | 1234639 | NULL      | 1970-01-06    |
+---------------------------------------------------------+
```

As expected, the `address` field is invisible to the nurse. This happens because, as previously described, `DENY` privileges *always* overrule `GRANT`. Since both roles `doctor` and `receptionist` were assigned to the nurse, the `DENIED` privileges of the `doctor` role are overruling the `GRANTED` privileges of the `receptionist`. Even if the nurse tries to write the address field, they would receive an error, and that is not what is desired here. To correct that, you can:

- Redefine the `doctor` role with only grants and define each `Patient` property the doctor should be able to read.
- Redefine the `nurse` role with the actual intended behavior.

The second option is simpler if you consider that the nurse is essentially the doctor without the `address` restrictions. In this case, you need to create a `nurse` role from scratch:

```
CREATE ROLE nurse
GRANT TRAVERSE ON GRAPH healthcare TO nurse;
GRANT READ {*} ON GRAPH healthcare TO nurse;
GRANT WRITE ON GRAPH healthcare TO nurse;
```

Now you assign the `nurse` role to the nurse Daniel, but remember to revoke the `doctor` and the `receptionist` roles so there are no privileges being overridden:

```
REVOKE ROLE doctor FROM daniel;
REVOKE ROLE receptionist FROM daniel;
GRANT ROLE nurse TO daniel;
```

This time, when the nurse Daniel takes another look at the patient records, they will see the address fields:

```
MATCH (n:Patient)
 WHERE n.dateOfBirth < date('1972-06-12')
RETURN n.name, n.ssn, n.address, n.dateOfBirth;
```

```
+-----------------------------------------------------------------------+
| n.name           | n.ssn   | n.address                | n.dateOfBirth |
+-----------------------------------------------------------------------+
| "Jane Anderson"  | 1234572 | "1 secret way, downtown" | 1971-05-26    |
| "Mark Stone"     | 1234586 | "1 secret way, downtown" | 1972-06-07    |
| "Joe Smith"      | 1234595 | "1 secret way, downtown" | 1970-12-28    |
| "Joe Jackson"    | 1234603 | "1 secret way, downtown" | 1970-08-31    |
| "Jane Jackson"   | 1234628 | "1 secret way, downtown" | 1972-01-31    |
| "Mary Anderson"  | 1234632 | "1 secret way, downtown" | 1971-01-07    |
| "Jack Svensson"  | 1234639 | "1 secret way, downtown" | 1970-01-06    |
+-----------------------------------------------------------------------+
```

The other main action that the nurse role should be able to perform is the primary doctor action of saving a diagnosis to the database:

```
WITH date(datetime({epochmillis:timestamp()})) AS today
MATCH (p:Patient)
  WHERE p.ssn = 1234657
MATCH (d:Disease)
  WHERE d.name = "Chronic Placeboitis"
MERGE (p)-[i:DIAGNOSIS {by: 'Daniel'}]->(d)
  ON CREATE SET i.date = today
RETURN p.name, d.name, i.by, i.date;
```

```
+-------------------------------------------------------------------+
| p.name            | d.name                | i.by     | i.date     |
+-------------------------------------------------------------------+
| "Sally Anderson"  | "Chronic Placeboitis" | "Daniel" | 2020-05-29 |
+-------------------------------------------------------------------+
```

Performing this action, otherwise reserved for the doctor role, involves more responsibility for the nurse. There might be nurses that should not be entrusted with this option, which is why you can divide the nurse role into *senior* and *junior* nurses, for example. Currently, Daniel is a senior nurse.

=== Privileges of junior nurses

Previously, creating the nurse role by combining the doctor and receptionist roles led to an undesired scenario as the DENIED privileges of the doctor role overrode the GRANTED privileges of the receptionist. In that case, the objective was to enhance the permissions of the *senior* nurse, but when it comes to the *junior* nurse, they should be able to perform the same actions as the *senior*, except adding diagnoses to the database.

To achieve this, you can create a special role that contains specifically only the additional restrictions:

```
CREATE ROLE disableDiagnoses;
DENY CREATE ON GRAPH healthcare RELATIONSHIPS DIAGNOSIS TO disableDiagnoses;
```

And then assign this new role to the nurse Daniel, so you can test the behavior:

```
GRANT ROLE disableDiagnoses TO daniel;
```

If you check what privileges Daniel has now, it is the combination of the two roles `nurse` and `disableDiagnoses`:

```
neo4j@system> SHOW USER daniel PRIVILEGES AS COMMANDS;
```

```
+-----------------------------------------------------------------------+
| command                                                               |
+-----------------------------------------------------------------------+
| "GRANT ACCESS ON HOME DATABASE TO $role"                              |
| "GRANT ACCESS ON DATABASE `healthcare` TO $role"                      |
| "GRANT EXECUTE PROCEDURE * ON DBMS TO $role"                          |
| "GRANT EXECUTE FUNCTION * ON DBMS TO $role"                           |
| "GRANT TRAVERSE ON GRAPH `healthcare` NODE * TO $role"                |
| "GRANT TRAVERSE ON GRAPH `healthcare` RELATIONSHIP * TO $role"        |
| "GRANT READ {*} ON GRAPH `healthcare` NODE * TO $role"                |
| "GRANT READ {*} ON GRAPH `healthcare` RELATIONSHIP * TO $role"        |
| "GRANT WRITE ON GRAPH `healthcare` TO $role"                          |
| "DENY CREATE ON GRAPH `healthcare` RELATIONSHIP DIAGNOSIS TO $role" |
+-----------------------------------------------------------------------+
```

Daniel can still see the address fields, and can even perform the diagnosis investigation that the `doctor` can perform:

```
MATCH (n:Patient)-[:HAS]->(s:Symptom)-[:OF]->(d:Disease)
WHERE n.ssn = 1234650
RETURN n.ssn, n.name, d.name, count(s) AS score ORDER BY score DESC;
```

```
+-----------------------------------------------------------+
| n.ssn   | n.name       | d.name                 | score |
+-----------------------------------------------------------+
| 1234650 | "Mark Smith" | "Chronic Whatitis"    | 3     |
| 1234650 | "Mark Smith" | "Chronic Someitis"    | 3     |
| 1234650 | "Mark Smith" | "Acute Someitis"      | 2     |
| 1234650 | "Mark Smith" | "Chronic Otheritis"   | 2     |
| 1234650 | "Mark Smith" | "Chronic Yellowitis"  | 2     |
| 1234650 | "Mark Smith" | "Chronic Placeboitis" | 2     |
| 1234650 | "Mark Smith" | "Acute Otheritis"     | 2     |
| 1234650 | "Mark Smith" | "Chronic Argitis"     | 2     |
| 1234650 | "Mark Smith" | "Acute Placeboitis"   | 2     |
| 1234650 | "Mark Smith" | "Acute Yellowitis"    | 1     |
| 1234650 | "Mark Smith" | "Acute Argitis"       | 1     |
| 1234650 | "Mark Smith" | "Acute Whatitis"      | 1     |
+-----------------------------------------------------------+
```

But when they try to save a diagnosis to the database, they will be denied that action:

```
WITH date(datetime({epochmillis:timestamp()})) AS today
MATCH (p:Patient)
  WHERE p.ssn = 1234650
MATCH (d:Disease)
  WHERE d.name = "Chronic Placeboitis"
MERGE (p)-[i:DIAGNOSIS {by: 'Daniel'}]->(d)
  ON CREATE SET i.date = today
RETURN p.name, d.name, i.by, i.date;
```

```
Create relationship with type 'DIAGNOSIS' is not allowed for user 'daniel' with roles [PUBLIC,
disableDiagnoses, nurse].
```

To promote Daniel back to senior nurse, revoke the role that introduced the restriction:

```
REVOKE ROLE disableDiagnoses FROM daniel;
```

=== Building a custom administrator role

The `itadmin` role was originally created by copying the built-in `admin` role and adding restrictions. However, there might be cases in which having `DENY`s can be less convenient than only having `GRANT`s. Instead, you can build the administrator role from the ground up.

The IT administrator Tina is able to create new users and assign them to the product roles as an `itadmin`, but you can create a more restricted role called `userManager` and grant it only the appropriate privileges:

```
CREATE ROLE userManager;
GRANT USER MANAGEMENT ON DBMS TO userManager;
GRANT ROLE MANAGEMENT ON DBMS TO userManager;
GRANT SHOW PRIVILEGE ON DBMS TO userManager;
```

Test the new behavior by revoking the `itadmin` role from Tina and grant them the `userManager` role instead:

```
REVOKE ROLE itadmin FROM tina
GRANT ROLE userManager TO tina
```

These are the privileges granted to `userManager`:

- `USER MANAGEMENT` allows creating, updating, and dropping users.
- `ROLE MANAGEMENT` allows creating, updating, and dropping roles as well as assigning roles to users.
- `SHOW PRIVILEGE` allows listing the users' privileges.

Listing Tina’s new privileges should now show a much shorter list than when they were a more powerful administrator with the `itadmin` role:

```
neo4j@system> SHOW USER tina PRIVILEGES AS COMMANDS;
```

```
+---------------------------------------------------+
| command                                           |
+---------------------------------------------------+
| "GRANT ACCESS ON HOME DATABASE TO $role"          |
| "GRANT ACCESS ON DATABASE `healthcare` TO $role"  |
| "GRANT EXECUTE PROCEDURE * ON DBMS TO $role"      |
| "GRANT EXECUTE FUNCTION * ON DBMS TO $role"       |
| "GRANT ROLE MANAGEMENT ON DBMS TO $role"          |
| "GRANT USER MANAGEMENT ON DBMS TO $role"          |
| "GRANT SHOW PRIVILEGE ON DBMS TO $role"           |
+---------------------------------------------------+
```

No other privilege management privileges were granted here. How much power this role should have would depend on the requirements of the system. Refer to the section Cypher Manual → The `admin` role for a complete list of privileges to consider.

Now Tina should be able to create new users and assign them to roles:

```
CREATE USER sally SET PASSWORD 'secret' CHANGE REQUIRED;
GRANT ROLE receptionist TO sally;
SHOW USER sally PRIVILEGES AS COMMANDS;
```

```
+---------------------------------------------------------------------+
| command                                                             |
+---------------------------------------------------------------------+
| "GRANT ACCESS ON HOME DATABASE TO $role"                            |
| "GRANT ACCESS ON DATABASE `healthcare` TO $role"                    |
| "GRANT EXECUTE PROCEDURE * ON DBMS TO $role"                        |
| "GRANT EXECUTE FUNCTION * ON DBMS TO $role"                         |
| "GRANT MATCH {*} ON GRAPH `healthcare` NODE Patient TO $role"       |
| "GRANT SET PROPERTY {*} ON GRAPH `healthcare` NODE Patient TO $role" |
| "GRANT CREATE ON GRAPH `healthcare` NODE Patient TO $role"          |
| "GRANT DELETE ON GRAPH `healthcare` NODE Patient TO $role"          |
+---------------------------------------------------------------------+
```

Integration with LDAP directory services

This page describes Neo4j support for integrating with LDAP systems. The following topics are covered:

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.

LDAP dynamic configuration settings

The following configuration settings can be updated while the database is running, see Update dynamic settings. Altering any of these settings clears the authentication and authorization cache.

Parameter name	Default value	Description
dbms.security.ldap.authentication.user_dn_template	`uid={0},ou=users,dc=example,dc=com`	Convert usernames into LDAP-specific fully qualified names required for logging in.
dbms.security.ldap.authorization.user_search_base	`ou=users,dc=example,dc=com`	Set the base object or named context to search for user objects.
dbms.security.ldap.authorization.user_search_filter	`(&(objectClass=*)(uid={0}))`	Set an LDAP search filter for a user principal.
dbms.security.ldap.authorization.group_membership_attributes	`memberOf`	List attribute names of a user object that contains groups to be used for mapping to roles. Common values: `memberOf` and `gidNumber`.
dbms.security.ldap.authorization.nested_groups_enabled	`false`	This setting determines whether multiple LDAP search results will be processed. This must be set to `true` in order to resolve nested group membership.
dbms.security.ldap.authorization.group_to_role_mapping		List an authorization mapping from groups to the pre-defined built-in roles `admin`, `architect`, `publisher`, `editor`, and `reader`, or to any other custom-defined roles.
dbms.security.ldap.authentication.attribute	`samaccountname`	Set the attribute to search for users with a system account.
dbms.security.ldap.authorization.access_permitted_group		Set an LDAP group of users with access rights. Users passing authentication are mapped to at least the `PUBLIC` role in addition to any roles assigned by the group to role mapping and have access to the database that those roles provide. If this attribute is set, users not part of this LDAP group will fail authentication, even if their credentials are correct.

All settings are defined at server startup time in the default configuration file neo4j.conf or can be modified at runtime using `dbms.setConfigValue()`.

Set Neo4j to use LDAP

First, you configure Neo4j to use LDAP as an authentication and authorization provider.

1. Uncomment the setting `dbms.security.auth_enabled=false` and change its value to `true` to turn on the security feature.

2. Uncomment the settings `dbms.security.authentication_providers` and `dbms.security.authorization_providers` and change their value to `ldap`. This way, the LDAP connector is used as a security provider for both authentication and authorization.

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 74. Example of LDAP groups to Neo4j roles mapping

```
dbms.security.ldap.authorization.group_to_role_mapping=\
    "cn=Neo4j Read Only,cn=users,dc=example,dc=com"      = reader;    \ 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.

Configure Neo4j to use Active Directory

You configure Neo4j to use the LDAP security provider to access and manage your Active Directory. There are three alternative ways to do that depending on your specific use case.

Configure Neo4j to support LDAP user ID authentication

This option allows users to log in with their LDAP user ID.

In the *neo4j.conf* file, uncomment and configure the following settings:

1. Configure LDAP to point to the AD server:

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

2. Provide details on the user structure of the LDAP directory:

```
dbms.security.ldap.authentication.user_dn_template=cn={0},cn=Users,dc=example,dc=com
dbms.security.ldap.authorization.user_search_base=cn=Users,dc=example,dc=com
dbms.security.ldap.authorization.user_search_filter=(&(objectClass=*)(cn={0}))
dbms.security.ldap.authorization.group_membership_attributes=memberOf
```

3. Map the groups in the LDAP system to the Neo4j built-in and custom roles. See Map the LDAP groups to the Neo4j roles.

Configure Neo4j to support attribute authentication

This is an alternative configuration for Active Directory that allows users to log in by providing an attribute to search for, by default `sAMAccountName`. The attribute has to be unique to be used as a lookup. You create a system account that has read-only access to the parts of the LDAP directory that you want. However, it does not need to have access rights to Neo4j or any other systems.

In the *neo4j.conf* file, uncomment and configure the following settings:

1. Configure LDAP to point to the AD server:

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

2. Provide details on the user structure of the LDAP directory (replacing `myattribute` with the actual attribute name):

```
dbms.security.ldap.authorization.user_search_base=cn=Users,dc=example,dc=com
dbms.security.ldap.authorization.user_search_filter=(&(objectClass=*)(myattribute={0}))
dbms.security.ldap.authorization.group_membership_attributes=memberOf
```

3. Map the groups in the LDAP system to the Neo4j built-in and custom roles. See Map the LDAP groups to the Neo4j roles.

4. Configure Neo4j to use a system account with read access to all users and groups in the LDAP server.

   a. Set `dbms.security.ldap.authorization.use_system_account` value to `true`.

   b. Set `dbms.security.ldap.authorization.system_username` value to the full Distinguished Name (DN) as the `dbms.security.ldap.authentication.user_dn_template` will not be applied to this username. For example,

```
dbms.security.ldap.authorization.system_username=cn=search-account,cn=Users,dc=example,dc=com
```

   c. Configure the LDAP system account password.

```
dbms.security.ldap.authorization.system_password=mypassword
```

   d. Configure which attribute to search for by adding the following lines to the *neo4j.conf* file (replacing `myattribute` with the actual attribute name):

```
dbms.security.ldap.authentication.search_for_attribute=true
dbms.security.ldap.authentication.attribute=myattribute
```

e. (Optional) Create an LDAP group to restrict authentication against the database to a subset of LDAP users:

```
dbms.security.ldap.authorization.access_permitted_group=cn=Neo4j Access,cn=users,dc=example,dc=com
```

Configure Neo4j to support `sAMAccountName` authentication by setting `user_dn_template`

This is an alternative configuration for Active Directory that allows all users from the specified domain to log in using `sAMAccountName`. With this option, you do not have to create a system account and store a system username/password in the config file. Instead, you set `{0}@example.com` as a value of the `user_dn_template` to enable the authentication to start at the root domain. This way, the whole tree is checked to find the user, regardless of where it is located within the LDAP directory tree.

In the *neo4j.conf* file, uncomment and configure the following settings:

1. Configure LDAP to point to the AD server:

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

2. Provide details on the user structure of the LDAP directory:

   ```
   dbms.security.ldap.authentication.user_dn_template={0}@example.com
   dbms.security.ldap.authorization.user_search_base=dc=example,dc=com
   dbms.security.ldap.authorization.user_search_filter=(&(objectClass=user)(sAMAccountName={0}))
   dbms.security.ldap.authorization.group_membership_attributes=memberOf
   ```

3. Map the groups in the LDAP system to the Neo4j built-in and custom roles. For more information, see Map the LDAP groups to the Neo4j roles.

The setting `dbms.security.ldap.authentication.search_for_attribute` should be set to the default value of false.

Configure Neo4j to perform nested group lookup

When a user is a member of a group (e.g. `engineers`) and that group is a member of another group (e.g. `employees`), Active Directory can be configured to perform a nested search such that a user in the group `engineers` would also be a member of the group `employees`. This in turn means that it is possible to configure a group to role mapping for `employees` which will transitively apply to `engineers`.

Active Directory facilitates nested search via the extensible match operator `LDAP_MATCHING_RULE_IN_CHAIN` (whose Object Identifier is 1.2.840.113556.1.4.1941). This operator walks the chain of ancestry in objects all the way to the root.

To set up nested search in the *neo4j.conf* file, configure the following settings:

1. Enable nested groups.

```
dbms.security.ldap.authorization.nested_groups_enabled=true
```

2. Provide details on the user structure of the LDAP directory:

```
dbms.security.ldap.authentication.user_dn_template=cn={0},cn=users,dc=example,dc=com
dbms.security.ldap.authorization.user_search_base=dc=example,dc=com
dbms.security.ldap.authorization.user_search_filter=(&(objectClass=*)(uid={0}))
```

3. Provide the nested groups search filter.
   This is the filter which will be used to perform the nested lookup of the user's groups. It should contain the placeholder token `\{0}`, which will be substituted with the user's Distinguished Name (which is found for the specified user principle using `dbms.security.ldap.authorization.user_search_filter`). This example features Active Directory's `LDAP_MATCHING_RULE_IN_CHAIN` (aka `1.2.840.113556.1.4.1941`) implementation:

```
dbms.security.ldap.authorization.nested_groups_search_filter=(&(objectclass=group)(member:1.2.840.1135
56.1.4.1941:={0}))
```

4. Provide group to role mappings, including ancestor groups if required:

```
dbms.security.ldap.authorization.group_to_role_mapping=\
"cn=engineers,cn=users,dc=example,dc=com"=procedures;\
"cn=employees,cn=users,dc=example,dc=com"=reader
```

In contrast to a non-nested-LDAP lookup, a nested group lookup does not perform an attribute-based lookup on the user object. Instead, the `dbms.security.ldap.authorization.group_membership_attributes` setting is ignored and the `dbms.security.ldap.authorization.user_search_filter` is only used to determine the Distinguished Name of the user. This is then substituted into the `dbms.security.ldap.authorization.nested_groups_search_filter` to perform a separate, nested lookup of the user's groups.

Configure Neo4j to use OpenLDAP

You configure the LDAP security provider to access and manage your OpenLDAP directory service.

In the neo4j.conf file, uncomment and configure the following settings:

1. Configure LDAP to point to the OpenLDAP server:

```
dbms.security.ldap.host=myopenldap.example.com
```

2. Provide details on the user structure of the LDAP directory:

```
dbms.security.ldap.authentication.user_dn_template=cn={0},ou=users,dc=example,dc=com
dbms.security.ldap.authorization.user_search_base=ou=users,dc=example,dc=com
dbms.security.ldap.authorization.user_search_filter=(&(objectClass=*)(uid={0}))
dbms.security.ldap.authorization.group_membership_attributes=gidNumber
```

3. (Optional) Create an LDAP group to restrict authentication against the database to a subset of LDAP users:

   ```
   dbms.security.ldap.authorization.access_permitted_group=501
   ```

4. Map the groups in the LDAP system to the Neo4j built-in and custom roles. For more information, see Map the LDAP groups to the Neo4j roles.

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
```

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()
```

Available methods of encryption

Specifying the `dbms.security.ldap.host` parameter configures using LDAP without encryption. Not specifying the protocol or port results in `ldap` being used over the default port `389`.

```
dbms.security.ldap.host=myactivedirectory.example.com
dbms.security.ldap.host=myactivedirectory.example.com:389
dbms.security.ldap.host=ldap://myactivedirectory.example.com
dbms.security.ldap.host=ldap://myactivedirectory.example.com:389
```

Use LDAP with encryption via StartTLS

To configure Active Directory with encryption via StartTLS, set the following parameters:

```
dbms.security.ldap.use_starttls=true
dbms.security.ldap.host=ldap://myactivedirectory.example.com
```

Use LDAP with encrypted LDAPS

To configure Active Directory with encrypted LDAPS, set `dbms.security.ldap.host` to one of the following. If you do not specify the port, the default one `636` is used.

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

Use a self-signed certificate (SSL) in a test environment

Production environments should always use an SSL certificate issued by a Certificate Authority for secure access to the LDAP server. However, there are scenarios, for example in test environments, where you may want to use an SSL certificate on the LDAP server.

To configure an SSL certificate on LDAP server, enter the details of the certificate using `server.jvm.additional` in *neo4j.conf*. The path to the certificate file `MyCert.jks` is an absolute path to the Neo4j server.

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

Integration with Single Sign-On Services

Neo4j supports OpenID Connect (OIDC), which allows for integration with many identity providers including Okta, Microsoft Azure Active Directory, and Google. This integration permits federated users, managed by the identity provider, to access Neo4j instead of, or in addition to the native users and roles. For examples with different providers and troubleshooting, see the SSO configuration tutorial.

OIDC configuration settings

Neo4j supports multiple OIDC identity providers at the same time, as such each provider configuration must be assigned a prefix to differentiate it from others. In the configuration examples below the provider-specific prefix is represented by `<provider>`, which should be replaced with a name representing your provider. For example, if you are using Okta as your identity provider you might use `okta` in the place of `<provider>` below.

The following configuration settings are important to consider when configuring single sign-on. For a more detailed overview of the single sign-on configuration options, see Configuration settings. Some of these settings can also be updated while the database is running, see Dynamic settings for more information on how to do this. Altering any of these settings causes users to re-authenticate as their permissions may have changed as a result.

Parameter name	Default value	Dynamic	Description
dbms.security.oidc.<provider>.display_name		false	The display name for the provider. This is displayed in clients such as Neo4j Browser and Bloom.
dbms.security.oidc.<provider>.auth_flow	pkce	true	The OIDC auth_flow for clients such as Neo4j Browser and Bloom to use. Supported values are `pkce` and `implicit`.
dbms.security.oidc.<provider>.well_known_discovery_uri		true	The OpenID Connect Discovery URL for the provider.
dbms.security.oidc.<provider>.auth_endpoint		true	URL of the provider’s Authorization Endpoint.

Parameter name	Default value	Dynamic	Description
dbms.security.oidc.<provider>.auth_params		true	Optional parameters that clients may require with the Authorization Endpoint. The map is a semicolon-separated list of key-value pairs. For example: k1=v1;k2=v2.
dbms.security.oidc.<provider>.token_endpoint		true	URL of the provider's OAuth 2.0 Token Endpoint.
dbms.security.oidc.<provider>.token_params		true	Option parameters that clients may require with the Token Endpoint. The map is a semicolon-separated list of key-value pairs. For example: k1=v1;k2=v2.
dbms.security.oidc.<provider>.jwks_uri		true	URL of the provider's JSON Web Key Set.
dbms.security.oidc.<provider>.user_info_uri		true	URL of the provider's UserInfo Endpoint.
dbms.security.oidc.<provider>.issuer		true	URL that the provider asserts as its issuer identifier. This will be checked against the `iss` claim in the token.
dbms.security.oidc.<provider>.audience		true	The expected value for the `aud` claim.
dbms.security.oidc.<provider>.client_id		true	The `client_id` of this client as issued by the provider.
dbms.security.oidc.<provider>.params		true	Option parameters that clients may require. The map is a semicolon-separated list of key-value pairs. For example: k1=v1;k2=v2.
dbms.security.oidc.<provider>.config		true	Option additional configuration that clients may require. The map is a semicolon-separated list of key-value pairs. For example: k1=v1;k2=v2.
dbms.security.oidc.<provider>.get_groups_from_user_info	false	true	Whether to fetch the groups claim from the user info endpoint on the identity provider. The default is `false`, to read the claim from the token.
dbms.security.oidc.<provider>.get_username_from_user_info	false	true	Whether to fetch the username claim from the user info endpoint on the identity provider. The default is `false`, to read the claim from the token.
dbms.security.oidc.<provider>.claims.username	sub	true	The claim to use for the database username.
dbms.security.oidc.<provider>.claims.groups		true	The claim to use for the database roles.
dbms.security.oidc.<provider>.authorization.group_to_role_mapping		true	List an authorization mapping from groups to the pre-defined built-in roles `admin`, `architect`, `publisher`, `editor`, and `reader`, or to any custom-defined roles.
dbms.security.logs.oidc.jwt_claims_at_debug_level_enabled	false	false	When set to `true`, it logs the claims from the JWT into the security log (provided the security log level is also set to `DEBUG`).

Configure Neo4j to use OpenID Connect

First, you configure Neo4j to use OpenID Connect as an authentication and authorization provider.

1. Uncomment the setting `dbms.security.auth_enabled=false` and change its value to `true` to enable the security feature.
2. Uncomment the settings `dbms.security.authentication_providers` and `dbms.security.authorization_providers` and change their value to `oidc-<provider>`, where `<provider>` maps to the provider name used in the configuration settings. This way, the OIDC connector is used as a security provider for both authentication and authorization. These configuration values are comma-separated lists, so if you wish to continue to use native authentication and authorization alongside SSO, then these providers can be added to the existing `native` provider:

   Example 75. Configuration

   ```
   dbms.security.authentication_providers=oidc-newsso,oidc-oldsso,native
   dbms.security.authorization_providers=oidc-newsso,oidc-oldsso,native
   ```

   This example has two OpenID Connect providers configured, as well as Neo4j native authorization and authentication.

Map the Identity Provider Groups to the Neo4j Roles

Before identity provider managed groups can be used with Neo4j, you have to decide on an approach for mapping identity provider groups to Neo4j roles. The simplest approach is to create identity provider groups with the same names as Neo4j roles. If you decide to go this way, no mapping configuration is necessary. Assuming, however, that identity provider groups do not directly map 1:1 to the desired Neo4j roles, it is necessary to map the identity provider groups to the Neo4j built-in and custom-defined roles. To do that, you need to know what privileges the Neo4j roles have, and based on these privileges, create the mapping to the groups defined in the identity provider. The map must be formatted as a semicolon-separated list of key-value pairs, where the key is a comma-separated list of the identity provider 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 76. Example of identity provider groups to Neo4j roles mapping

```
dbms.security.oidc.mysso.authorization.group_to_role_mapping=\
    neo4j_readonly      = reader;    \ ①
    neo4j_rw      = editor,publisher; \ ②
    neo4j_rw,neo4j_create     = publisher; \ ③
    neo4j_create,neo4j_schema = architect; \
    neo4j_dba  = admin; \
    neo4j_exec       = rolename ④
```

① Mapping of an identity provider group to a Neo4j built-in role.

② Mapping of an identity provider group to two Neo4j built-in roles.

③ Mapping of two identity provider groups to a Neo4j built-in role.

④ Mapping of an identity provider group to a custom-defined role. Custom-defined roles, such as `rolename`, must be explicitly created using the `CREATE ROLE rolename` command before they can be used to grant privileges. See the Cypher Manual → Creating roles.

Configure Neo4j to use an OpenID Connect Identity Provider

This option allows users to log in through an OIDC compliant identity provider by offering a token from the provider instead of a username and password. Typically, these tokens take the form of a signed JSON Web Token (JWT). In the configuration examples below, we are using `mysso` as our provider name. It is recommended to use a name describing the provider that is being integrated.

OpenID Connect Using JWT Claims

In this configuration, Neo4j receives a JWT from the identity provider containing claims representing the database username (e.g. email), and the Neo4j roles.

1. Set a Display Name

   In the *neo4j.conf* file, uncomment and configure the following settings:

   ```
   dbms.security.oidc.mysso.display_name=SSO Provider
   ```

   This is displayed on a button on the login page of clients such as Neo4j Browser and Bloom, so that users can identify the provider they are using to login.

2. Configure Discovery:

   Uncomment and configure the following settings:

   ```
   dbms.security.oidc.mysso.well_known_discovery_uri=https://my-idp.example.com/.well-known/openid-
   configuration
   ```

   The `well_known_discovery` endpoint of the identity provider supplies the OpenID Provider Metadata to allow Neo4j to interact with a provider. It is also possible to configure the provider settings manually:

```
dbms.security.oidc.mysso.auth_endpoint=https://my-idp.example.com/openid-connect/auth
dbms.security.oidc.mysso.token_endpoint=https://my-idp.example.com/openid-connect/token
dbms.security.oidc.mysso.jwks_uri=https://my-idp.example.com/openid-connect/certs
dbms.security.oidc.mysso.user_info_uri=https://my-idp.example.com/openid-connect/userinfo
dbms.security.oidc.mysso.issuer=abcd1234
```

Manual settings always take priority over those retrieved from the discovery endpoint.

3. Configure Audience:

   Provide the expected value for the audience(`aud`) claim:

   ```
   dbms.security.oidc.mysso.claims.audience=myaudience
   ```

   In some situations there may be multiple values for the `aud` claim. In this situation, the id_token should contain an authorized party(`azp`) claim containing the client id, which is configured as follows:

   ```
   dbms.security.oidc.mysso.claims.client_id=myclientid
   ```

4. Configure Claims:

   Provide the name of the claims that map to the database username and roles. `username` is expected to be a string claim and `roles` is expected to be a list of strings.

   ```
   dbms.security.oidc.mysso.claims.username=sub
   dbms.security.oidc.mysso.claims.groups=roles
   ```

5. Optionally, map the groups in the OIDC groups claim to the Neo4j built-in and custom roles. See Map the Identity Provider Groups to the Neo4j Roles.

OpenID Connect Fetching Claims from Provider

In this configuration, Neo4j receives a token from the identity provider and uses that token to call back to the identity provider using its UserInfo endpoint to retrieve claims for the database username and Neo4j roles.

1. Configure as for JWT Claims.

   Configure Neo4j for OpenID Connect Using JWT Claims.

2. Configure the claims to fetch from the UserInfo endpoint:

   ```
   dbms.security.oidc.mysso.get_username_from_user_info=true
   dbms.security.oidc.mysso.get_groups_from_user_info=true
   ```

   It is possible to fetch just the username, just the groups, or both from the userinfo endpoint.

Use a self-signed certificate (SSL) in a test environment

Production environments should always use an SSL certificate issued by a Certificate Authority for secure access to the identity provider. However, there are scenarios, for example in test environments, where you may want to use a self-signed SSL certificate on the identity provider server.

To configure a self-signed SSL certificate used on an identity provider server, enter the details of a Java keystore containing the relevant certificates using `server.jvm.additional` in *neo4j.conf*. The path to the certificate file `MyCert.jks` is an absolute path to the Neo4j server.

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

Debug logging of JWT claims

While setting up an OIDC integration, it is sometimes necessary to perform troubleshooting. In these cases, it can be useful to view the claims contained in the JWT supplied by the identity provider. To enable the logging of these claims at `DEBUG` level in the security log, set dbms.security.logs.oidc.jwt_claims_at_debug_level_enabled to be `true` and the security log level to `DEBUG`.

> Make sure to set dbms.security.logs.oidc.jwt_claims_at_debug_level_enabled back to `false` for production environments to avoid unwanted logging of potentially sensitive information. Also, bear in mind that the set of claims provided by an identity provider in the JWT can change over time.

Manage procedure and user-defined function permissions

To be able to run a procedure or user-defined function, the user needs to have the corresponding execute privilege. Procedures and user-defined functions are executed according to the same security rules as regular Cypher statements, e.g. a procedure performing writes will fail if called by a user that only has *read* privileges.

Procedures and user-defined functions can also be run with privileges exceeding the users own privileges. This is called *execution boosting*. The elevated privileges only apply within the procedure or user-defined function; any operation performed outside will still use the users original privileges.

> The steps below assume that the procedure or user-defined function is already developed and installed.
>
> Please refer to Java Reference → Extending Neo4j for a description on creating and using user-defined procedures and functions.

Manage procedure permissions

Procedure permissions can be managed using the native execute privileges. These control whether the user is allowed to both execute a procedure, and which set of privileges apply during the execution.

A procedure may be run using the `EXECUTE PROCEDURE` privilege.

This allows the user to execute procedures that match the globbed procedures.

Example 77. Grant privilege to execute procedure

```
GRANT EXECUTE PROCEDURE db.schema.visualization ON DBMS TO visualizer
```

This will allow any user with the `visualizer` role to execute the `db.schema.visualization`. E.g. a user that also have the following privileges:

```
GRANT TRAVERSE ON GRAPH * NODES A, B TO role
GRANT TRAVERSE ON GRAPH * RELATIONSHIP R1 TO role
```

When calling the `db.schema.visualization` procedure that user will only see the `A` and `B` nodes and `R1` relationships, even though there might exist other nodes and relationships.

A procedure may also be executed with elevated privileges using the `EXECUTE BOOSTED PROCEDURE` privilege.

The `EXECUTE BOOSTED PROCEDURE` privilege only controls the privileges used during the execution and not the execution itself. The user needs both `EXECUTE PROCEDURE` and `EXECUTE BOOSTED PROCEDURE` to execute the procedure with elevated privileges.

This allows the user to successfully execute procedures that would otherwise fail during execution with their assigned roles. The user is given full privileges for the procedure, during the execution of the procedure only.

Example 78. Grant privilege to use elevated privileges during procedure execution

```
GRANT EXECUTE BOOSTED PROCEDURE db.schema.visualization ON DBMS TO visualizer
```

This will allow any user with the `visualizer` role to execute the `db.schema.visualization` with elevated privileges. When calling the `db.schema.visualization` procedure that user will see all nodes and relationships that exist in the graph, even though they have no traversal privileges.

Manage user-defined function permissions

User-defined function permissions can be managed using the native execute privileges. These control if the user is both allowed to execute a user-defined function, and which set of privileges apply during the execution.

A user-defined function may be executed using the `EXECUTE USER DEFINED FUNCTION` privilege.

This allows the user to execute user-defined functions that match the globbed user-defined function.

Example 79. Grant privilege to execute user-defined function

```
GRANT EXECUTE USER DEFINED FUNCTION apoc.any.properties ON DBMS TO custom
```

This will allow any user with the `custom` role to execute the `apoc.any.properties`. E.g. a user that also have the following privilege:

```
GRANT MATCH {visibleProp} ON GRAPH * NODES A TO role
```

When calling the user-defined function `MATCH (a:A) RETURN apoc.any.properties(a) AS properties`, they will only see the `visibleProp` even though there might exist other properties.

A user-defined function may also be executed with elevated privileges using the `EXECUTE BOOSTED USER DEFINED FUNCTION` privilege.

The `EXECUTE BOOSTED USER DEFINED FUNCTION` privilege only controls the privileges used during the execution and not the execution itself. The user needs both `EXECUTE USER DEFINED FUNCTION` and `EXECUTE BOOSTED USER DEFINED FUNCTION` to execute the user-defined function with elevated privileges.

This allows the user to successfully execute user-defined functions that would otherwise fail during execution with their assigned roles. The user is given full privileges for the user-defined function, during the execution of the function only.

Example 80. Grant privilege to use elevated privileges during user-defined function execution

```
GRANT EXECUTE BOOSTED USER DEFINED FUNCTION apoc.any.properties ON DBMS TO custom
```

This will allow any user with the `custom` role to execute the `apoc.any.properties` with elevated privileges. E.g. a user that also 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.

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.

[13] For more information, see Update dynamic settings

Security

This section describes how to ensure physical data security according to industry best practices with regards to server and network security:

- Securing extensions
- SSL framework
- Credentials handling in Neo4j Browser
- Security checklist

Additionally, logs can be useful for continuous analysis, or for specific investigations. Facilities are available for producing security event logs as well as query logs as described in Monitoring.

Refer to Authentication and authorization for information on how to manage users and their authentication and authorization.

Securing extensions

Neo4j can be extended by writing custom code which can be invoked directly from Cypher, as described in Java Reference → User-defined functions. This page describes how to ensure the security of these additions.

Allow listing

Allow listing can be used to allow the loading of only a few extensions from a larger library.

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 81. 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.

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`, and `7000`)
- `backups` (port - `6362`)

This page describes how to set up SSL within your environment, how to view, validate, and test the certificates.

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. See table below.

- 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 41.77.Final, which requires tcnative 2.0.52. Only Neo4j 3.5 still uses older versions of Netty. See the table below for detailed information:

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.

Certificates and private keys

Certificates

The SSL configuration requires SSL certificates to be issued by a Certificate Authority (CA). All certificates must follow the X.509 standard and be saved in a PEM-encoded file.

Valid trusted certificates can be generated for free using non-profit CAs such as Let's Encrypt.

Example public.crt file

```
-----BEGIN CERTIFICATE-----
MIIDojCCAoqgAwIBAgIBATANBgkqhkiG9w0BAQsFADBhMQswCQYDVQQGEwJTRTEQ
     ...
xsUBvcQuyxewlvWRS18YB51J+yu0Xg==
-----END CERTIFICATE-----
```

The instructions on this page assume that you have already obtained the required certificates from the CA.

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.

Transformations

Neo4j requires all SSL certificates to be in the PEM format. If your certificate is in the binary DER format, you must transform it into PEM format.

Transform DER format certificate to PEM format

```
openssl x509 -in cert.der -inform der -outform pem -out cert.crt
```

Private keys

Private keys must be in the standard format PKCS #8 and saved as a PEM-encoded file.

Example private.key file

```
-----BEGIN PRIVATE KEY-----
MIICdQIBADANBgkqhkiG9w0BAQEFAASCAl8wggJbAgEAAoGBAN5D0I4bgdQK4In6
     ...
oaMe91ZPQ1JI
-----END PRIVATE KEY-----
```

Private keys can also be encrypted with a passphrase according to the PKCS #5 standard.

Example private.key file with passphrase encryption

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
MIICojAcBgoqhkiG9w0BDAEEMA4ECL3eSAoRlJ18AgIIAASCAoCj7WDyjsgcawdv
     ...
lYeSjVah
-----END ENCRYPTED PRIVATE KEY-----
```

Transformation

If the private key is encoded with the old PKCS #1 format, the file will typically start with the line:

```
-----BEGIN RSA PRIVATE KEY-----
```

You can convert it to PKCS #8 format with the following command:

Convert a PKCS #1 key to a PKCS #8 key

```
openssl pkcs8 -topk8 -in pkcs1.key -out pkcs8.key
```

An unencrypted private key could be PKCS #1 or PKCS #8. It can be encrypted with the following command:

Convert an unencrypted key to an encrypted PKCS #8 key using 256bit AES in cipher-block-chaining (CBC) mode

```
openssl pkcs8 -topk8 -v2 aes-256-cbc -v2prf hmacWithSHA512 -in pkcs1or8.key -out pkcs8.encrypted.key
```

Supported encryption arguments to openssl are:

- `-v1 PBE-MD5-DES`
- `-v1 PBE-SHA1-3DES`
- `-v1 PBE-SHA1-RC2-40`
- `-v1 PBE-SHA1-RC2-128`
- `-v1 PBE-SHA1-RC4-40`
- `-v1 PBE-SHA1-RC4-128`
- `-v1 PBE-SHA1-2DES`
- `-v2 aes-128-cbc -v2prf hmacWithSHA1`
- `-v2 aes-128-cbc -v2prf hmacWithSHA224`

- `-v2 aes-128-cbc -v2prf hmacWithSHA256`
- `-v2 aes-128-cbc -v2prf hmacWithSHA384`
- `-v2 aes-128-cbc -v2prf hmacWithSHA512`
- `-v2 aes-256-cbc -v2prf hmacWithSHA1`
- `-v2 aes-256-cbc -v2prf hmacWithSHA224`
- `-v2 aes-256-cbc -v2prf hmacWithSHA256`
- `-v2 aes-256-cbc -v2prf hmacWithSHA384`
- `-v2 aes-256-cbc -v2prf hmacWithSHA512`

It is highly recommended to use one of the `-v2` variants, because they offer more robust encryption.

Versions before Neo4j 5.0 allow keys to be stored with the old PKCS #1 standard. You can identify them by the line `-----BEGIN RSA PRIVATE KEY-----` at the beginning of the file. While Neo4j 5.0 can load and use those keys, they are considered deprecated and will be removed in a future version.

Validate the key and the certificate

If you need, you can validate the key file and the certificate as follows:

Validate the key

```
openssl rsa -in private.key -check
```

Validate certificate in the PEM format

```
openssl x509 -in public.crt -text -noout
```

Connectors

Before enabling SSL support, you must ensure the following connector configurations to avoid errors:

- Set `server.https.enabled` to `true` when using HTTPS.
- Set `server.bolt.tls_level` to `REQUIRED` or `OPTIONAL` when using Bolt.

For more information on configuring connectors, see Configure connectors.

Configuration

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

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

- `scope` is the name of the communication channel, such as `bolt`, `https`, `cluster`, and `backup`.
- `setting-suffix` can be any of the following:

Setting suffix	Description	Default value
	Basic	
`enabled`	Setting this to `true` enables this policy.	`false`
`base_directory`	The base directory under which cryptographic objects are searched for by default.	`certificates/<scope>`
`private_key`	The private key used for authenticating and securing this instance.	`private.key`
`private_key_password`	The passphrase to decode the private key. Only applicable for encrypted private keys.	
`public_certificate`	A public certificate matching the private key signed by a CA.	`public.crt`
`trusted_dir`	A directory populated with certificates of trusted parties.	`trusted/`
`revoked_dir`	A directory populated with certificate revocation lists (CRLs).	`revoked/`
	Advanced	
`verify_hostname`	Enabling this setting turns on client-side hostname verification. After receiving the server's public certificate, the client compares the address it uses against the certificate Common Name (CN) and Subject Alternative Names (SAN) fields. If the address does not match those fields, the client disconnects.	`false`
`ciphers`	A comma-separated list of ciphers suites allowed during cipher negotiation. Valid values depend on the current JRE 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. Bolt connector is used by Cypher Shell, Neo4j Browser, and by the officially supported language drivers. Bolt connector is enabled by default but its encryption is disabled. To enable the encryption over Bolt, create the folder structure and place the key file and the certificates under those. Then, you need to configure the SSL Bolt policies in the *neo4j.conf* file.

1. Enable the Bolt connector to enable SSL over Bolt:

   ```
   server.bolt.enabled=true (default is true)
   ```

2. Set up the *bolt* folder under *certificates*.

   a. Create a directory *bolt* under *<neo4j-home>/certificates* folder:

      ```
      mkdir certificates/bolt
      ```

   b. Create a directory *trusted* and *revoked* under *<neo4j-home>/certificates/bolt* folder:

      ```
      mkdir certificates/bolt/trusted
      mkdir certificates/bolt/revoked
      ```

3. Place the certificates *private.key* and the *public.crt* files under *<neo4j-home>/certificates/bolt* folder:

   ```
   cp /path/to/certs/private.key certificates/bolt
   cp /path/to/certs/public.crt certificates/bolt
   ```

4. Place the *public.crt* file under the *<neo4j-home>/certificates/bolt/trusted* folder.

   ```
   cp /path/to/certs/public.crt certificates/bolt/trusted
   ```

5. (Optional) If a particular certificate is revoked, then place it under *<neo4j-home>/certificates/bolt/revoked* folder.

   ```
   cp /path/to/certs/public.crt certificates/bolt/revoked
   ```

   The folder structure should look like this with the right file permissions and the groups and ownerships:

Path	Directory/File	Owner	Group	Permission	Unix/Linux View
/data/neo4j/certificates/bolt	Directory	`neo4j`	`neo4j`	`0755`	`drwxr-xr-x`
/data/neo4j/certificates/bolt/public.crt	File	`neo4j`	`neo4j`	`0644`	`-rw-r--r--`
/data/neo4j/certificates/bolt/private.key	File	`neo4j`	`neo4j`	`0400`	`-r--------`
/data/neo4j/certificates/bolt/trusted	Directory	`neo4j`	`neo4j`	`0755`	`drwxr-xr-x`
/data/neo4j/certificates/bolt/trusted/public.crt	File	`neo4j`	`neo4j`	`0644`	`-rw-r--r--`
/data/neo4j/certificates/bolt/revoked	Directory	`neo4j`	`neo4j`	`0755`	`drwxr-xr-x`

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:

   ```
   server.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:

```
server.https.enabled=true (default is false)
```

2. Set up the *https* folder under *certificates*.

   a. Create a directory *https* under *<neo4j-home>/certificates* folder:

   ```
   mkdir certificates/https
   ```

   b. Create a directory *trusted* and *revoked* under *<neo4j-home>/certificates/https* folder:

   ```
   mkdir certificates/https/trusted
   mkdir certificates/https/revoked
   ```

3. Place the certificates *private.key* and the *public.crt* files under *<neo4j-home>/certificates/https* folder:

```
cp /path/to/certs/private.key certificates/https
cp /path/to/certs/public.crt certificates/https
```

4. Place the *public.crt* file under the *<neo4j-home>/certificates/https/trusted* folder.

```
cp /path/to/certs/public.crt certificates/https/trusted
```

5. (Optional) If a particular certificate is revoked, then place it under *<neo4j-home>/certificates/https/revoked* folder.

```
cp /path/to/certs/public.crt certificates/https/revoked
```

The folder structure should look like this with the right file permissions and the groups and ownerships:

Path	Directory/File	Owner	Group	Permission	Unix/Linux View
/data/neo4j/certificates/https	Directory	`neo4j`	`neo4j`	`0755`	`drwxr-xr-x`
/data/neo4j/certificates/https/public.crt	File	`neo4j`	`neo4j`	`0644`	`-rw-r—r--`
/data/neo4j/certificates/https/private.key	File	`neo4j`	`neo4j`	`0400`	`-r--------`
/data/neo4j/certificates/https/trusted	Directory	`neo4j`	`neo4j`	`0755`	`drwxr-xr-x`
/data/neo4j/certificates/https/trusted/public.crt	File	`neo4j`	`neo4j`	`0644`	`-rw-r—r--`
/data/neo4j/certificates/https/revoked	Directory	`neo4j`	`neo4j`	`0755`	`drwxr-xr-x`

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
   ```

   > If the certificate is a different path outside of NEO4J_HOME, then set the absolute path for the certificates directory.

   c. Set the HTTPS client authentication to `NONE` to disable the mutual authentication:

   ```
   dbms.ssl.policy.https.client_auth=NONE
   ```

   d. Disable HTTP connector:

   ```
   server.http.enabled=false
   ```

7. Test the SSL connection to the specified host and HTTPS port and view the certificate:

   ```
   openssl s_client -connect my_domain.com:7473
   ```

Configure SSL for intra-cluster communications

Intra-cluster encryption is the security solution for the cluster communication. The Neo4j cluster communicates on 3 ports:

- 5000 - Discovery management
- 6000 - Transactions
- 7000 - Raft communications

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 `server.cluster.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 suites that will be allowed during cipher negotiation. Valid values depend on the current JRE and SSL provider. For Oracle JRE here is the list of supported ones - https://docs.oracle.com/en/java/javase/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:

```
server.bolt.ocsp_stapling_enabled=true (default = false)
```

SSL logs

All information related to SSL can be found in the *debug.log* file. You can also enable additional debug logging for SSL by adding the following configuration to the *neo4j.conf* file and restarting Neo4j.

```
server.jvm.additional=-Djavax.net.debug=ssl:handshake
```

This will log additional information in the *neo4j.log* file. In some installations done using `rpm` based installs, *neo4j.log* is not created. To get the contents of this, since *neo4j.log* just contains `STDOUT` content, look for the `neo4j` service log contents using `journalctl`:

```
neo4j@ubuntu:/var/log/neo4j$ journalctl -u neo4j -b > neo4j.log
neo4j@ubuntu:/var/log/neo4j$ vi neo4j.log
```

Terminology

The following terms are relevant to SSL support within Neo4j:

Certificate Authority (CA)

A trusted entity that issues electronic documents that can verify the identity of a digital entity. The term commonly refers to globally recognized CAs, but can also include internal CAs that are trusted inside of an organization. The electronic documents are digital certificates. They are an essential part of secure communication, and play an important part in the Public Key Infrastructure.

Certificate Revocation List (CRL)

In the event of a certificate being compromised, that certificate can be revoked. This is done by means of a list (located in one or several files) spelling out which certificates are revoked. The CRL is always issued by the CA which issues the corresponding certificates.

cipher

An algorithm for performing encryption or decryption. In the most general implementation of encrypted communications, Neo4j makes implicit use of ciphers that are included as part of the Java platform. The configuration of the SSL framework also allows for the explicit declaration of allowed ciphers.

communication channel

A means for communicating with the Neo4j database. Available channels are:

- Bolt client traffic
- HTTPS client traffic

- intra-cluster communication
- backup traffic

cryptographic objects

A term denoting the artifacts private keys, certificates and CRLs.

configuration parameters

These are the parameters defined for a certain ssl policy in *neo4j.conf*.

certificate

SSL certificates are issued by a trusted certificate authority (CA). The public key can be obtained and used by anyone to encrypt messages intended for a particular recipient. The certificate is commonly stored in a file named *<file name>.crt*. This is also referred to as the public key.

SAN

SAN is an acronym for *Subject Alternative Names*. It is an extension to certificates that one can include optionally. When presented with a certificate that includes SAN entries, it is recommended that the address of the host is checked against this field. Verifying that the hostname matches the certificate SAN helps prevent attacks where a rogue machine has access to a valid key pair.

SSL

SSL is an acronym for *Secure Sockets Layer*, and is the predecessor of TLS. It is common to refer to SSL/TLS as just SSL. However, the modern and secure version is TLS, which is also the default in Neo4j.

SSL policy

An SSL policy in Neo4j consists of a a digital certificate and a set of configuration parameters defined in *neo4j.conf*.

PKCS #1

PKCS #1 is the first family of standards called Public-Key Cryptography Standards (PKCS). It provides the basic definitions and recommendations for implementing the RSA algorithm for public-key cryptography. It defines the mathematical properties of public and private keys, primitive operations for encryption and signatures, secure cryptographic schemes, and related ASN.1 syntax representations.

PKCS #5

PKCS #5 contains recommendations for implementating password-based cryptography, covering key derivation functions, encryption schemes, message authentication schemes, and *ASN.1* syntax, identifying the techniques.

PKCS #8

PKCS #8 is a standard syntax for storing private key information. The PKCS #8 private key may be encrypted with a passphrase using the PKCS #5 standards, which support multiple ciphers. The main difference from PKCS #1 is that it allows more algorithms than RSA and supports stronger encryption of the private key.

private key

The private key ensures that encrypted messages can be deciphered only by the intended recipient.

The private key is commonly stored in a file named *<file name>.key*. It is important to protect the private key to ensure the integrity of encrypted communication.

Public Key Infrastructure (PKI)

A set of roles, policies, and procedures needed to create, manage, distribute, use, store, and revoke digital certificates and manage public-key encryption.

public key

The public key can be obtained and used by anyone to encrypt messages intended for a particular recipient. This is also referred to as the certificate.

TLS protocol

The cryptographic protocol that provides communications security over a computer network. The Transport Layer Security (TLS) protocol and its predecessor, the Secure Sockets Layer (SSL) protocol, are both frequently referred to as "SSL".

TLS version

A version of the TLS protocol.

X.509

X.509 is an International Telecommunication Union (ITU) standard defining the format of public key certificates.

Browser credentials handling

Neo4j Browser has two mechanisms for avoiding users having to repeatedly enter their Neo4j credentials. This page explains how to control how credentials are handled.

First, while the Browser is open in a web browser tab, it ensures that the existing database session is kept alive. This is subject to a timeout. The timeout is configured in the setting `browser.credential_timeout`. The timeout is reset whenever there is user interaction with the Browser.

Second, the Browser can also cache the user's Neo4j credentials locally. When credentials are cached, they are stored unencrypted in the web browser's local storage. If the web browser tab is closed and then re-opened, the session is automatically re-established using the cached credentials. This local storage is also subject to the timeout configured in the setting `browser.credential_timeout`. In addition, caching credentials in browser local storage can be disabled altogether. To disable credentials caching, set `browser.retain_connection_credentials=false` in the server configuration.

If the user issues a `:server disconnect` command then any existing session is terminated and the credentials are cleared from local storage.

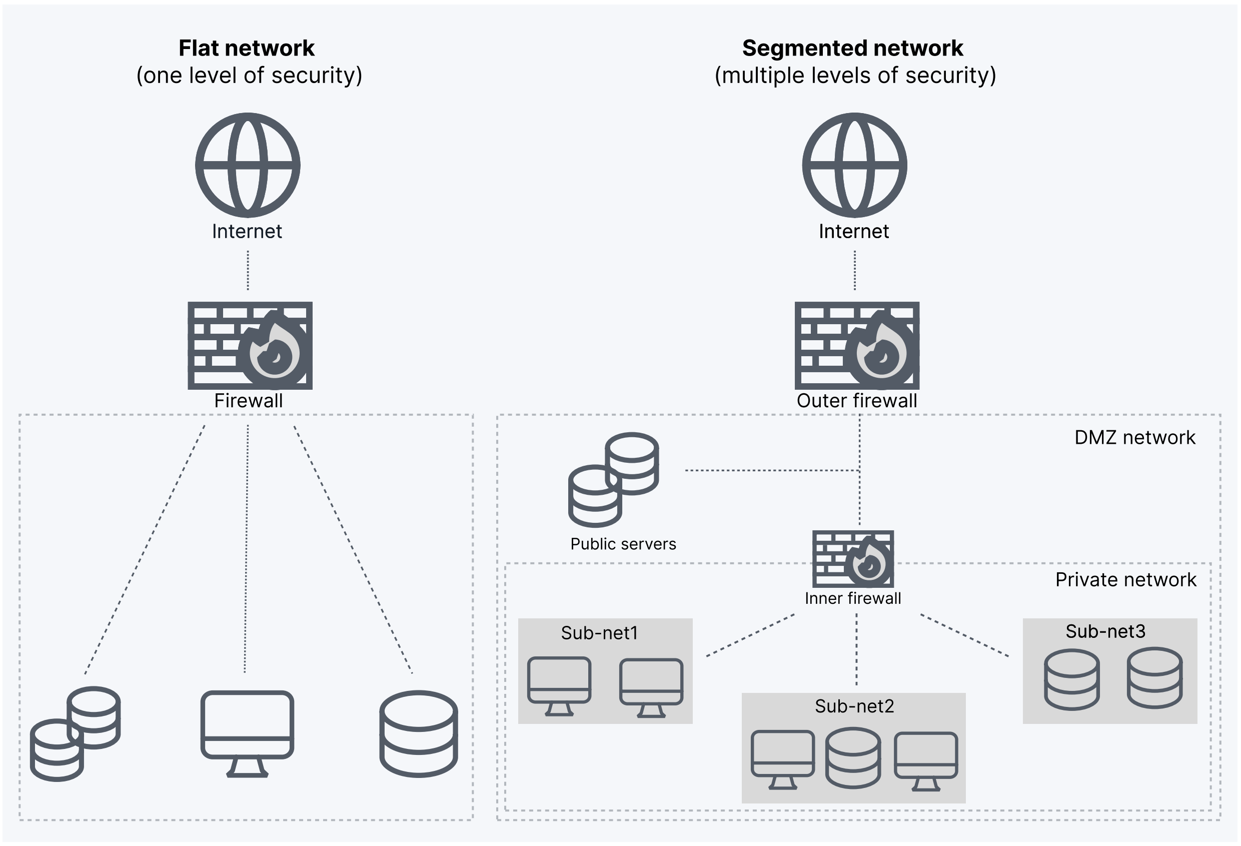
For more information on how to administer and use Neo4j Browser, see the Neo4j Browser manual → Browser operations.

Security checklist

The following checklist highlights the specific areas within Neo4j that may need extra attention to ensure the appropriate level of security for your application after Neo4j is deployed.

1. Deploy Neo4j on safe servers in secure networks:

   a. Use subnets and firewalls to segment the network.



   b. Open only the ports that you need. For a list of relevant ports, see Ports.

      In particular, ensure that there is no external access to the port specified by the setting `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 clustering, configure and use encryption for intra-cluster communication. For details, see Intra-cluster encryption.

   e. If using clustering, configure and use encryption for backups. This ensures that only servers with the specified SSL policy and SSL certificates can access the server and perform the backup.

   f. For configuring your Bolt and HTTPS connectors, refer to Configure connectors.

   g. If using LDAP, configure your LDAP system with encryption via StartTLS. For more information, see Use LDAP with encryption via StartTLS.

4. Be on top of the security for custom extensions:

   a. Validate any custom code you deploy (procedures and unmanaged extensions) and ensure that they do not unintentionally expose any parts of the product or data.

   b. Survey the settings `dbms.security.procedures.unrestricted` and `dbms.security.procedures.allowlist` to ensure that they exclusively contain intentionally exposed extensions.

5. Make sure you have the 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.

Performance

This section describes factors that affect operational performance and how to tune Neo4j for optimal throughput. The following topics are covered:

- Memory configuration — How to configure memory settings for efficient operations.
- Index configuration — How to configure indexes.
- Garbage collector — How to configure the Java Virtual Machine's garbage collector.
- Bolt thread pool configuration — How to configure the Bolt thread pool.
- Linux file system tuning — How to configure the Linux file system.
- Locks and deadlocks — Information about locks and deadlocks in Neo4j.
- Disks, RAM and other tips — Disks, RAM and other tips.
- Statistics and execution plans — How schema statistics and execution plans affect Cypher query performance.
- Space reuse — Data deletion and storage space reuse.

Memory configuration

This page describes the different aspects of Neo4j memory configuration and use. The RAM of the Neo4j server has a number of usage areas, with some sub-areas:

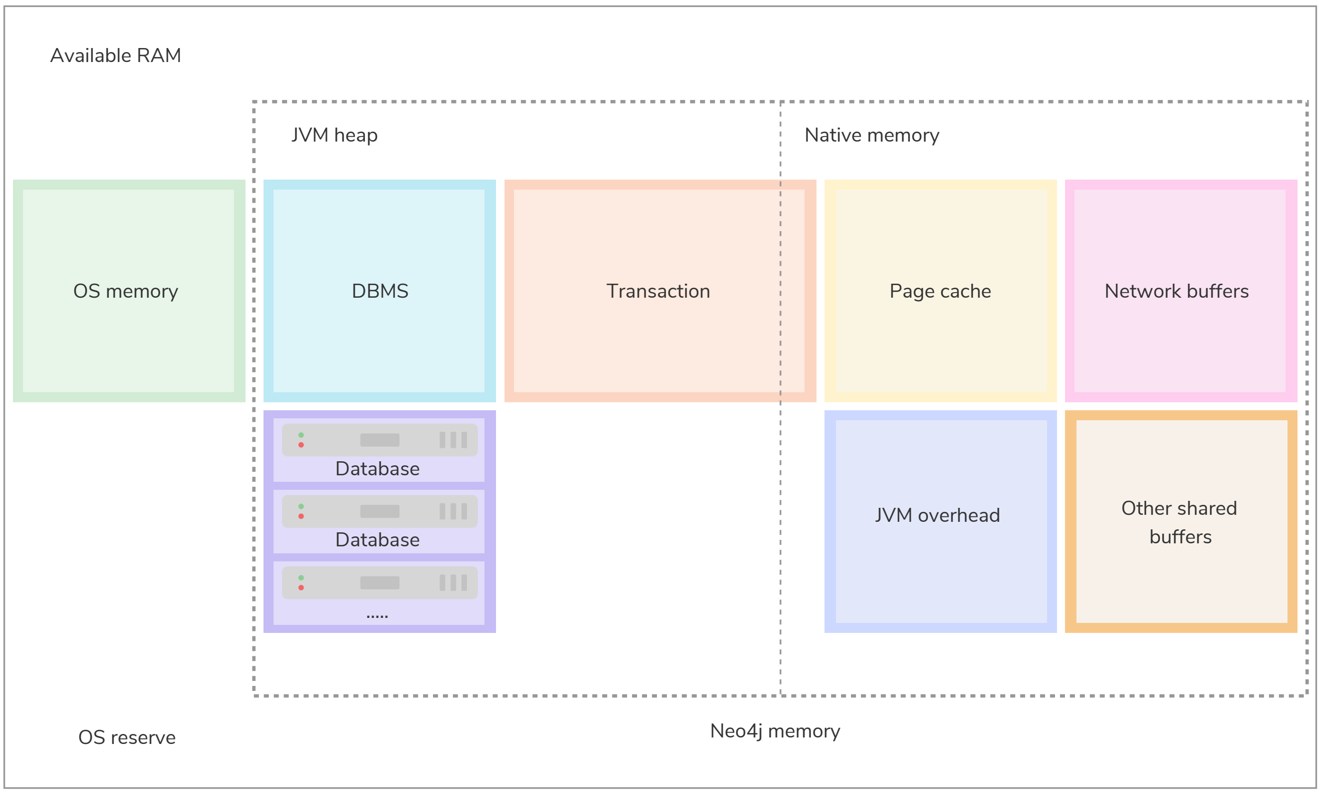


Figure 8. Neo4j memory management

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 `server.memory.heap.initial_size` and `server.memory.heap.max_size`. It is recommended to set these two parameters to the same value to avoid unwanted full garbage collection pauses.

Generally, to aid performance, you should configure a large enough heap to sustain concurrent operations.

Native memory

Native memory, sometimes referred to as off-heap memory, is memory directly allocated by Neo4j from the OS. This memory will grow dynamically as needed and is not subject to the garbage collector.

DBMS

The database management system, or DBMS, contains the global components of the Neo4j instance. For example, the bolt server, logging service, monitoring service, etc.

Database

Each database in the system comes with an overhead. In deployments with multiple databases, this overhead needs to be accounted for.

Transaction

When executing a transaction, Neo4j holds not yet committed data, the result, and intermediate states of the queries in memory. The size needed for this is very dependent on the nature of the usage of Neo4j. For example, long-running queries, or very complicated queries, are likely to require more memory. Some parts of the transactions can optionally be placed off-heap, but for the best performance, it is recommended to keep the default with everything on-heap.

This memory group can be limited with the setting `dbms.memory.transaction.total.max`.

Page cache

The page cache is used to cache the Neo4j data stored on disk. The caching of graph data and indexes into memory helps avoid costly disk access and result in optimal performance.

The parameter for specifying how much memory Neo4j is allowed to use for the page cache is: `server.memory.pagecache.size`.

Network buffers

Direct buffers are used by Neo4j to send and receive data. Direct byte buffers are important for improving performance because they allow native code and Java code to share data without copying it. However, they are expensive to create, which means byte buffers are usually reused once they are created.

Other shared buffers

This includes unspecified shared direct buffers.

JVM overhead

The JVM will require some memory to function correctly. For example, this can be:

- **Thread stacks** – Each thread has its own call stack. The stack stores primitive local variables and object references along with the call stack (list of method invocations) itself. The stack is cleaned up as stack frames move out of context, so there is no GC performed here.
- **Metaspace** – Metaspace stores the java class definitions and some other metadata.
- **Code cache** – The JIT compiler stores the native code it generates in the code cache to improve performance by reusing it.

For more details and means of limiting the memory used by the JVM please consult your JVM documentation.

Considerations

Always use explicit configuration

To have good control of the system behavior, it is recommended to always define the page cache and heap size parameters explicitly in *neo4j.conf*. Otherwise, Neo4j computes some heuristic values at startup based on the available system resources.

Initial memory recommendation

Use the `neo4j-admin server memory-recommendation` command to get an initial recommendation for how to distribute a certain amount of memory. The values may need to be adjusted to cater for each specific use case.

Inspect the memory settings of all databases in a DBMS

The `neo4j-admin server memory-recommendation` command is useful for inspecting the current distribution of data and indexes.

Example 82. Use `neo4j-admin server memory-recommendation` to inspect the memory settings of all your databases

Estimate the total size of the database files.

```
$neo4j-home> bin/neo4j-admin server memory-recommendation
...
...
...
# Total size of lucene indexes in all databases: 6690m
# Total size of data and native indexes in all databases: 17050m
```

You can see that the Lucene indexes take up approximately 6.7GB of data, and that the data volume and native indexes combined take up approximately 17GB.

Using this information, you can do a sanity check of your memory configuration:

- Compare the value for data volume and native indexes to the value of `server.memory.pagecache.size`.
- For cases when *off-heap* transaction state is used, estimate transactional workload and how much memory is left to the value of `dbms.tx_state.max_off_heap_memory`.
- Compare the value for Lucene indexes to how much memory is left after assigning `server.memory.pagecache.size` and `server.memory.heap.initial_size`.

In some production systems the access to memory is limited and must be negotiated between different areas. Therefore, it is recommended to perform a certain amount of testing and tuning of these settings to figure out the optimal division of the available memory.

Limit transaction memory usage recommendation

The measured heap usage of all transactions is only an estimate and the actual heap utilization may be slightly larger or slightly smaller than the estimated value. In some cases, limitations of the estimation algorithm to detect shared objects at a deeper level of the memory graph could lead to overestimations. This is because a conservative estimate is given based on aggregated estimations of memory usage, where the identities of all contributing objects are not known, and cannot be assumed to be shared. For example, when you use `UNWIND` on a very large list, or expand a variable length or shortest path pattern, where many relationships are shared between the computed result paths.

In these cases, if you experience problems with a query that gets terminated, you can execute the same query with the transaction memory limit disabled. If the actual heap usage is not too large, it might succeed without triggering an out-of-memory error.

Capacity planning

In many use cases, it is advantageous to try to cache as much of the data and indexes as possible. The following examples illustrate methods for estimating the page cache size, depending on whether you are already running in production or planning for a future deployment:

Example 83. 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 server memory-recommendation
...
...
...
# Total size of lucene indexes in all databases: 6690m
# Total size of data and native indexes in all databases: 35050m
```

You can see that the data volume and native indexes combined take up approximately 35GB. In your specific use case, you estimate that 20% will provide sufficient head room for growth.

`server.memory.pagecache.size` = 1.2 * (35GB) = 42GB

You configure the page cache by adding the following to *neo4j.conf*:

```
server.memory.pagecache.size=42GB
```

Example 84. Estimate page cache for a new Neo4j database

When planning for a future database, it is useful to run an import with a fraction of the data, and then multiply the resulting store size delta by that fraction plus some percentage for growth.

1. Run the `memory-recommendation` command to see the total size of the data and indexes in all current databases.

   ```
   $neo4j-home> bin/neo4j-admin server memory-recommendation
   ...
   ...
   ...
   # Total size of lucene indexes in all databases: 6690m
   # Total size of data and native indexes in all databases: 35050m
   ```

2. Import 1/100th of the data and again measure the data volume and native indexes of all databases.

   ```
   $neo4j-home> bin/neo4j-admin server memory-recommendation
   ...
   ...
   ...
   # Total size of lucene indexes in all databases: 6690m
   # Total size of data and native indexes in all databases: 35400m
   ```

   You can see that the data volume and native indexes combined take up approximately 35.4GB.

3. Multiply the resulting store size delta by that fraction.

   35.4GB - 35GB = 0.4GB * 100 = 40GB

4. Multiply that number by 1.2 to size up the result, and allow for 20% growth.

   `server.memory.pagecache.size` = 1.2 * (40GB) = 48GB

5. Configure the page cache by adding the following to *neo4j.conf*:

   ```
   server.memory.pagecache.size=48G
   ```

Limit transaction memory usage

By using the `dbms.memory.transaction.total.max` setting you can configure a global maximum memory usage for all of the transactions running on the server. This setting must be configured low enough so that you do not run out of memory. If you are experiencing `OutOfMemory` messages during high transaction load, try to lower this limit.

Neo4j also offers the following settings to provide fairness, which can help improve stability in multi-tenant deployments.

- The setting `db.memory.transaction.total.max` limits the transaction memory usage per database.
- The setting `db.memory.transaction.max` constrains each transaction.

When any of the limits are reached, the transaction is terminated without affecting the overall health of the database.

To help configure these settings you can use the following commands to list the current usage:

```
CALL dbms.listPools()
SHOW TRANSACTIONS
```

Or alternatively, you can monitor the memory usage of each query in the query.log.

Index configuration

This page describes how to configure Neo4j indexes to enhance search performance and enable full-text search. The supported index types are:

- Range
- Point
- Text
- Full-text
- Token lookup

All types of indexes can be created and dropped using Cypher and they can also all be used to index both nodes and relationships. The token lookup index is the only index present by default in the database.

Range, point, text, and full-text indexes provide mapping from a property value to an entity (node or relationship). Token lookup indexes are different and provide mapping from labels to nodes, or from relationships types to relationships, instead of between properties and entities.

When you write a Cypher query, you do not need to specify which indexes to use. Cypher's query planner decides which of the available indexes to use.

The rest of this page provides information on the available indexes and their configuration aspects. For further details on creating, querying, and dropping indexes, see Cypher Manual → Indexes to support full-text search.

The type of an index can be identified according to the table below:

Index type	Cypher command	Core API
Range index	`SHOW INDEXES#RANGE`	`org.neo4j.graphdb.schema.IndexType#RANGE`
Point index	`SHOW INDEXES#POINT`	`org.neo4j.graphdb.schema.IndexType#POINT`
Text index	`SHOW INDEXES#TEXT`	`org.neo4j.graphdb.schema.IndexType#TEXT`
Full-text index	`SHOW INDEXES#FULLTEXT`	`org.neo4j.graphdb.schema.IndexType#FULLTEXT`

Index type	Cypher command	Core API
Token lookup index	`SHOW INDEXES#LOOKUP`	`org.neo4j.graphdb.schema.IndexType#LOOKUP`

You cannot have indexes of the same type over the same properties.

Range indexes

Range indexes can be used for exact lookups on all types of values, range scans, full scans, and prefix searches.

Range indexes are the most general purpose of the property indexes, as they support all value types and wide range of operations.

Limitations on key size

Range has a key size limit of around 8kB.

If a transaction reaches the key size limit for one or more of its changes, that transaction fails before committing any changes. If the limit is reached during index population, the resulting index is in a failed state, and as such is not usable for any queries.

Workarounds to address limitations

Since text index has key size limit of around 32kB, the key size limit of range index can be worked around by using a text index instead. However, text index is not a general purpose index like range index, so this workaround cannot be applied to all cases. For more information, see Text indexes.

Point indexes

Point indexes are a type of highly-specialized, single-property index and they only index properties with Point values, unlike range indexes.

Point indexes are designed to speed up spatial queries, specifically the `distance` and `bounding box` queries. Exact lookups are the only non-spatial query that this index type supports.

For more information on the queries a point index can be used for, refer to Cypher Manual → Query Tuning → The use of indexes.

Point index optionally accepts configuration properties for tuning the behavior of spatial search. For more information on configuring point index, refer to Cypher Manual → Indexes for search performance.

Text indexes

Text indexes are a type of single-property index. Unlike range indexes, text indexes index only properties with string values.

Text indexes are specifically designed to deal with `ENDS WITH` or `CONTAINS` queries efficiently. They are used through Cypher and they support a smaller set of string queries. Even though text indexes do support other text queries, `ENDS WITH` or `CONTAINS` queries are the only ones for which this index type provides an advantage over a range index.

For more information on the queries a text index can be used for, refer to Cypher Manual → Query Tuning → The use of indexes.

For more information on the different index types, refer to Cypher Manual → Indexes for search performance.

Neo4j 5.1 introduces an improved index provider, `text-2.0`, that performs significantly better. New Text indexes will use the `text-2.0` provider by default. Pre-existing indexes will continue to use `text-1.0`. You must re-create these indexes to benefit from this improvement.

Limitations

Text indexes only index single property strings.

The index has a key size limit for single property strings of around 32kB. If a transaction reaches the key size limit for one or more of its changes, that transaction fails before committing any changes. If the limit is reached during index population, the resulting index is in a failed state, and as such is not usable for any queries.

Full-text indexes

Full-text indexes are optimized for indexing and searching texts.

Even though text and full-text indexes might seem to solve very similar problems, there are essential differences. Unlike text indexes, which index only single property strings, full-text indexes can index any kind of string data. Text indexes solve substring match and exact string match according to the semantics defined by the Cypher language. While, full-text indexes use pluggable analyzers, many of which provide language-specific processing of the text that allows for more sophisticated queries than a simple substring match. Depending on which analyzer is used, the full-text index can be used for different text search types, such as exact matches, relevance matches, phrase queries, autocompletion, and many others. Additionally, the results are ordered by relevance.

An example of a use case for full-text indexes is parsing a book for a certain term and taking advantage of the knowledge that the book is written in a certain language. The use of an *analyzer* for that language enables the exclusion of stop words, such as "*if*" and "*and*", and the inclusion of word forms.

Another use case example is indexing the various address fields and text data in a corpus of emails. Using the `email` analyzer, you can find all emails that are sent from/to or mention a specific email account.

In contrast to range and text indexes, full-text indexes are queried using built-in procedures. They are however created and dropped using Cypher. The use of full-text indexes does require familiarity with how those indexes operate.

Full-text indexes are powered by the Apache Lucene indexing and search library. A full description on how to create and use full-text indexes is provided in the Cypher Manual → Indexes to support full-text search.

Configuring a full-text index

The following options are available for configuring full-text indexes. For a complete list of Neo4j procedures, see Operations Manual → Procedures.

`db.index.fulltext.default_analyzer`
: The name of the default analyzer when creating a new Full-text index. Once created, the index's analyzer is not affected by this setting.

`db.index.fulltext.eventually_consistent`
: The default consistency model when creating a new full-text index. Once created, the index's consistency model is not affected by this setting.

  Indexes are normally fully consistent, and the committing of a transaction does not return until both the store and indexes are updated. Eventually consistent full-text indexes, on the other hand, are not updated as part of a commit but instead have their updates queued up and applied in a background thread. This means that there can be a short delay between committing a change and that change becoming visible via any eventually consistent full-text indexes. This delay is just an artifact of the queueing and is usually relatively small since eventually consistent indexes are updated "as soon as possible".

  By default, this is turned off, and full-text indexes are fully consistent.

`db.index.fulltext.eventually_consistent_index_update_queue_max_length`
: Eventually consistent full-text indexes have their updates queued up and applied in a background thread, and this setting determines the maximum size of that update queue. If the maximum queue size is reached, then committing transactions block and wait until there is more room in the queue before adding more updates to it.

  This setting applies to all eventually consistent full-text indexes, and they all use the same queue. The maximum queue length must be at least 1 index update and no more than 50 million due to heap space usage considerations.

  The default maximum queue length is 10.000 index updates.

Selecting an analyzer

By default, the full-text index uses the `standard-no-stop-words` analyzer, specified in `db.index.fulltext.default_analyzer` configuration setting. This analyzer is the same as the Lucene's `StandardAnalyzer` , except no stop-words are filtered out.

To specify another analyzer, use the `OPTIONS` clause of the full-text index creation command. The list of all possible analyzers is available via the `db.index.fulltext.listAvailableAnalyzers()` Cypher procedure.

By default, the analyzer analyzes both the indexed values and query string. In some cases, however, using different analyzers for the indexed values and query string is more appropriate. You can do that by specifying an analyzer for the query string when using the full-text search procedures.

For a detailed information on how to create and use full-text indexes, see the Cypher Manual → Indexes to support full-text search.

Per-property analyzer

A full-text index can be created over multiple properties. If different analyzers for different properties are required, the standard approach in Lucene is to create a custom composite analyzer. The Lucene project provides `PerFieldAnalyzerWrapper` that can associate analyzers with specific fields. For more information, see the Lucene official documentation.

Token lookup indexes

Token lookup indexes are used to look up nodes with a specific label or relationships of a specific type. They are always created over all labels or relationship types. Therefore, a databases can have a maximum of two token lookup indexes - one for nodes and one for relationships.

Use and significance

Token lookup indexes are the most important indexes as they significantly speed up the population of other indexes. They are also essential for the Cypher queries execution and Core API operations. Therefore, dropping them should be carefully considered.

The node label lookup index is important for queries that match a node by one or more labels. It can also be used for matching labels and properties of a node when there are no suitable indexes available. Likewise, the relationship type lookup index is important for queries that match relationships by their types.

Most queries are executed by matching nodes and expanding their relationships. Hence, the node label lookup index is slightly more significant than the relationship type lookup index.

Both node and relationship type lookup index are present by default in all databases created in 4.3 and onwards.

Databases created before 4.3

Databases created before 4.3 do not get relationship lookup index automatically, in order to preserve backwards compatibility and performance characteristics of such databases.

If needed, such databases can get a relationship type lookup index by creating it explicitly through Cypher.

> Creating relationship type lookup index on a large database can take significant amount of time, as all relationships need to be scanned when populating such index.

Tuning of the garbage collector

This page discusses the effects of the Java Virtual Machine’s garbage collector with regards to Neo4j performance. In this setting, the heap is separated into an *old generation* and a *young generation*, while new objects are allocated in the young generation, and then later moved to the old generation, if they stay live (in use) for long enough.

When a generation fills up, the garbage collector performs a collection, during which all other threads in the process are paused. The young generation is quick to collect since the pause time correlates with the *live* set of objects. In the old generation, pause times roughly correlates with the size of the heap. For this reason, the heap should ideally be sized and tuned such that transaction and query state never makes it to the old generation.

The heap size is configured with the `server.memory.heap.initial_size` (in MBs) setting in the *neo4j.conf* file. The initial size of the heap is specified by the `server.memory.heap.initial_size` setting, or with the `-Xms???m` flag, or chosen heuristically by the JVM itself if left unspecified. The JVM will automatically grow the heap as needed, up to the maximum size. The growing of the heap requires a full garbage collection cycle. It is recommended to set the initial heap size and the maximum heap size to the same value. This way the pause that happens when the garbage collector grows the heap can be avoided.

If the new generation is too small, short-lived objects may be moved to the old generation too soon. This is called premature promotion and will slow the database down by increasing the frequency of old generation garbage collection cycles. If the new generation is too big, the garbage collector may decide that the old generation does not have enough space to fit all the objects it expects to promote from the new to the old generation. This turns new generation garbage collection cycles into old generation garbage collection cycles, again slowing the database down. Running more concurrent threads means that more allocations can take place in a given span of time, in turn increasing the pressure on the new generation in particular.

The *Compressed OOPs* feature in the JVM allows object references to be compressed to use only 32 bits. The feature saves a lot of memory but is only available for heaps up to 32 GB. The maximum applicable size varies from platform and JVM version. The `-XX:+UseCompressedOops` option can be used to verify whether the system can use the *Compressed OOPs* feature. If it cannot, this will will be logged in the default process output stream.

How to tune the specific garbage collection algorithm depends on both the JVM version and the workload. It is recommended to test the garbage collection settings under realistic load for days or weeks. Problems like heap fragmentation can take a long time to surface.

To gain good performance, these are the things to look into first:

- Make sure the JVM is not spending too much time performing garbage collection. The goal is to have a large enough heap to make sure that heavy/peak load will not result in so called GC-trashing. Performance can drop as much as two orders of magnitude when GC-trashing happens. Having too large heap may also hurt performance so you may have to try some different heap sizes.
- Neo4j needs enough heap memory for the transaction state and query processing, plus some head-room for the garbage collector. As heap memory requirements are so workload-dependent, it is common to see heap memory configurations from 1 GB, up to 32 GB.

Edit the following properties:

Table 43. neo4j.conf JVM tuning properties

Property Name	Meaning
`server.memory.heap.initial_size`	initial heap size (in MB)
`server.memory.heap.max_size`	maximum heap size (in MB)
`server.jvm.additional`	additional literal JVM parameter

Bolt thread pool configuration

The Bolt connector is backed by a thread pool on the server side, whereas the thread pool is constructed as part of the server startup process. This page discusses the thread pool infrastructure and how it can be configured.

How thread pooling works

The Bolt thread pool has a minimum and a maximum capacity. It starts with a minimum number of threads available, and grows up to the maximum count depending on the workload. Threads that sit idle for longer than a specified time period are stopped and removed from the pool in order to free up resources. However, the size of the pool will never go below the minimum.

Each connection being established is assigned to the connector's thread pool. Idle connections do not consume any resources on the server side, and they are monitored against messages arriving from the client. Each message arriving on a connection triggers the scheduling of a connection on an available thread in the thread pool. If all the available threads are busy, and there is still space to grow, a new thread is created and the connection is handed over to it for processing. If the pool capacity is filled up, and no threads are available to process, the job submission is rejected and a failure message is generated to notify the client of the problem.

The default values assigned to the Bolt thread pool will fit most workloads, so it is generally not necessary to configure the connection pool explicitly. If the maximum pool size is set too low, an exception will be thrown with an error message indicating that there are no available threads to serve. The message will also be written to neo4j.log.

Any connection with an active explicit, or implicit, transaction will stick to the thread that starts the transaction, and will not return that thread to the pool until the transaction is closed. Therefore, in applications that are making use of explicit transactions, it is important to close the transactions appropriately. To learn more about transactions, refer to the Neo4j Driver manuals.

Configuration options

The following configuration options are available for configuring the Bolt connector:

Table 44. Thread pool options

Option name	Default	Description
`server.bolt.thread_pool_min_size`	`5`	The minimum number of threads that will always be up even if they are idle.

Option name	Default	Description
server.bolt.thread_pool_max_size	400	The maximum number of threads that will be created by the thread pool.
server.bolt.thread_pool_keep_alive	5m	The duration that the thread pool will wait before killing an idle thread from the pool. However, the number of threads will never go below server.bolt.thread_pool_min_size.

How to size your Bolt thread pool

Select values for thread pool sizing based on your workload. Since each active transaction will borrow a thread from the pool until the transaction is closed, it is basically the minimum and maximum active transaction at any given time that determine the values for pool configuration options. You can use the monitoring capabilities (see Monitoring) of the database to discover more about your workload.

Configure server.bolt.thread_pool_min_size based on your minimum or average workload. Since there will always be this many amount of threads in the thread pool, sticking with lower values may be more resource-friendly than having too many idle threads waiting for job submissions.

Configure server.bolt.thread_pool_max_size based on your maximum workload. This should basically be set after the maximum number of active transactions that is expected on the server. You should also account for non-transaction operations that will take place on the thread pool, such as connection and disconnection of clients.

Example 85. 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.

```
server.bolt.thread_pool_min_size=10
server.bolt.thread_pool_max_size=100
server.bolt.thread_pool_keep_alive=10m
```

Linux file system tuning

Databases often produce many small and random reads when querying data, and few sequential writes when committing changes. For maximum performance, it is recommended to store database and transaction logs on separate physical devices. This page covers Neo4j I/O behavior and how to optimize for operations on disk.

Often, recommended practice is to disable file and directory access time updates. This way, the file system won’t have to issue writes that update this meta-data, thus improving write performance.

Since databases can put a high and consistent load on a storage system for a long time, it is recommended to use a file system that has good aging characteristics. The EXT4 and XFS file systems are both supported.

💡	While EXT4 and XFS file systems are both supported, XFS can provide marginal performance benefits (up to 10%) in some workloads, but uses more disk space (up to 2x) compared to EXT4. Therefore, EXT4 is generally recommended.

A high read and write I/O load can also degrade SSD performance over time. The first line of defense against SSD wear is to ensure that the working dataset fits in RAM. A database with a high write workload will, however, still cause wear on SSDs. The simplest way to combat this is to over-provision; use SSDs that are at least 20% larger than you strictly need them to be.

⚠	Neo4j does not recommend and support the usage of NFS or NAS as database storage.

Locks and deadlocks

Neo4j is fully ACID compliant. This means that all database operations which access graphs, indexes, or schemas must be performed in a transaction. When a write transaction occurs, Neo4j takes locks to preserve data consistency while updating.

Locks

Locks are taken automatically by the queries that users run. They ensure that a node/relationship is locked to one particular transaction until that transaction is completed. In other words, a lock on a node or a relationship by one transaction will pause additional transactions which seek to concurrently modify the same node or relationship. As such, locks prevent concurrent modifications of shared resources between transactions.

Locks are used in Neo4j to ensure data consistency and isolation levels. They not only protect logical entities (such as nodes and relationships), but also the integrity of internal data structures.

The default isolation is read-committed isolation level. It is, however, possible to manually acquire write locks on nodes and relationships. For more information on how to manually acquire write locks, see Neo4j Java Reference Manual → Transaction management.

Lock contention

Lock contention may arise if an application needs to perform concurrent updates on the same nodes/relationships. In such a scenario, transactions must wait for locks held by other transactions to be released in order to be completed. If two or more transactions attempt to modify the same data concurrently, it will increase the likelihood of a deadlock (explained in more detail below). In larger graphs, it is less likely that two transactions modify the same data concurrently, and so the likelihood of a deadlock is reduced. That said, even in large graphs, a deadlock can occur if two or more transactions are attempting to modify the same data concurrently.

Locks in practice

Table 45. Locks taken for specific graph modifications

Modification	Lock taken
Creating a node	No lock
Updating a node label	Node lock
Updating a node property	Node lock
Deleting a node	Node lock
Creating a relationship*	If node is sparse: node lock. If node is dense: node delete prevention lock.**
Updating a relationship property	Relationship lock
Deleting a relationship*	If node is sparse: node lock. If node is dense: node delete prevention lock. Relationship lock for both sparse and dense nodes.

**Applies for both source nodes and target nodes.*

***A node is considered dense if it at any point has had 50 or more relationships (i.e. it will still be considered dense even it comes to have less than 50 relationships at any point in the future). A node is considered sparse if it has never had more than 50 relationships.*

Additional locks are often taken to maintain indexes and other internal structures, depending on how other data in the graph is affected by a transaction. For these additional locks, no assumptions or guarantees can be made with regard to which lock will or will not be taken.

Locks and dense nodes

When creating or deleting relationships in Neo4j, dense nodes are not exclusively locked during a transaction (a node is considered dense if, at any point in time, it has had more than 50 relationships). Rather, internally shared locks prevent the deletion of nodes, and shared degree locks are acquired for synchronizing with concurrent label changes for those nodes (to ensure correct count updates).

At commit time, relationships are inserted into their relationship chains at places that are currently uncontested (i.e. not currently modified by another transaction), and the surrounding relationships are exclusively locked.

In other words, relationship modifications acquires coarse-grained shared node locks when doing the operation in the transaction, and then acquires precise exclusive relationship locks during commit.

The locking is very similar for sparse and dense nodes. The biggest contention for sparse nodes is the update of the degree (i.e. number of relationships) for the node. Dense nodes store this data in a concurrent data structure, and so can avoid exclusive node locks in almost all cases for relationship modifications.

Deadlocks

A deadlock occurs when two transactions are blocked by each other because they are attempting to concurrently modify a node or a relationship that is locked by the other transaction. In such a scenario, neither of the transactions will be able to proceed.

When Neo4j detects a deadlock, the transaction is terminated (with the transient error message code `Neo.TransientError.Transaction.DeadlockDetected`).

For example, running the following two queries in Cypher-shell at the same time, will result in a deadlock because they are attempting to modify the same node properties concurrently:

Transaction A

```
:begin
MATCH (n:Test) SET n.prop = 1
WITH collect(n) as nodes
CALL apoc.util.sleep(5000)
MATCH (m:Test2) SET m.prop = 1;
```

Transaction B

```
:begin
MATCH (n:Test2) SET n.prop = 1
WITH collect(n) as nodes
CALL apoc.util.sleep(5000)
MATCH (m:Test) SET m.prop = 1;
```

The following error message is thrown:

```
The transaction will be rolled back and terminated. Error: ForsetiClient[transactionId=6698, clientId=1]
can't acquire ExclusiveLock{owner=ForsetiClient[transactionId=6697, clientId=3]} on NODE(27), because
holders of that lock are waiting for ForsetiClient[transactionId=6698, clientId=1].
 Wait list:ExclusiveLock[
Client[6697] waits for [ForsetiClient[transactionId=6698, clientId=1]]]
```

The Cypher clause `MERGE` takes locks out of order to ensure uniqueness of the data, and this may prevent Neo4j's internal sorting operations from ordering transactions in a way which avoids deadlocks. When possible, users are, therefore, encouraged to use the Cypher clause `CREATE` instead, which does not take locks out of order.

Avoiding deadlocks

Most likely, a deadlock will be resolved by retrying the transaction. This will, however, negatively impact the total transactional throughput of the database, so it is useful to know about strategies to avoid deadlocks.

Neo4j assists transactions by internally sorting operations (see below for more information about internal locks). However, this internal sorting only applies for the locks taken when creating or deleting relationships. Users are, therefore, encouraged to sort their operations in cases where Neo4j does not internally assist, such as when locks are taken for property updates. This is done by ensuring that updates occur in the same order. For example, if the three locks `A`, `B`, and `C` are always taken in the same order (e.g. `A→B→C`), then a transaction will never hold lock `B` while waiting for lock `A` to be released, and so a deadlock will not occur.

Another option is to avoid lock contention by not modifying the same entities concurrently.

For more information about deadlocks, see Neo4j Java Reference Manual → Transaction management.

Internal lock types

To avoid deadlocks, internal locks should be taken in the following order:

Lock type	Locked entity	Description
`LABEL` or `RELATIONSHIP_TYPE`	Token id	Schema locks, which lock indexes and constraints on the particular label or relationship type.
`SCHEMA_NAME`	Schema name	Lock a schema name to avoid duplicates. Note, collisions are possible because the hash is stringed (this only affects concurrency and not correctness).
`NODE_RELATIONSHIP_GROUP_DELETE`	Node id	Lock taken on a node during the transaction creation phase to prevent deletion of said node and/or relationship group. This is different from the `NODE` lock to allow concurrent label and property changes together with relationship modifications.
`NODE`	Node id	Lock on a node, used to prevent concurrent updates to the node records (i.e. add/remove label, set property, add/remove relationship). Note that updating relationships will only require a lock on the node if the head of the relationship chain/relationship group chain must be updated, since that is the only data part of the node record.
`DEGREES`	Node id	Used to lock nodes to avoid concurrent label changes when a relationship is added or deleted. Such an update would otherwise lead to an inconsistent count store.
`RELATIONSHIP_DELETE`	Relationship id	Lock a relationship for exclusive access during deletion.

Lock type	Locked entity	Description
`RELATIONSHIP_GROUP`	Node id	Lock the full relationship group chain for a given dense node.* This will not lock the node, in contrast to the lock `NODE_RELATIONSHIP_GROUP_DELETE`.
`RELATIONSHIP`	Relationship	Lock on a relationship, or more specifically a relationship record, to prevent concurrent updates.

**A node is considered dense if it at any point has had 50 or more relationships (i.e. it will still be considered dense even it comes to have less than 50 relationships at any point in the future).*

Note that these lock types may change without any notification between different Neo4j versions.

Disks, RAM and other tips

As with any persistence solution, performance depends a lot on the persistence media used. In general, the faster storage you have, and the more of your data you can fit in RAM, the better performance you will get. This page provides an overview of performance considerations for disk and RAM when running Neo4j.

Storage

There are many performance characteristics to consider for your storage solutions. The performance can vary hugely in orders of magnitude. Generally, having all your data in RAM achieves maximum performance.

If you have multiple disks or persistence media available, it may be a good idea to divide the store files and transaction logs across those disks. Keeping the store files on disks with low seek time can do wonders for read operations.

Use tools like `dstat` or `vmstat` to gather information when your application is running. If the swap or paging numbers are high, that is a sign that the database does not quite fit in memory. In this case, database access can have high latencies.

To achieve maximum performance, it is recommended to provide Neo4j with as much RAM as possible to avoid hitting the disk.

Page cache

When Neo4j starts up, its page cache is empty and needs to warm up. The pages, and their graph data contents, are loaded into memory on demand as queries need them. This can take a while, especially for large stores. It is not uncommon to see a long period with many blocks being read from the drive, and high IO wait times. This will show up in the page cache metrics as an initial spike in page faults. The page fault spike is then followed by a gradual decline of page fault activity, as the probability of queries needing a page that is not yet in memory drops.

Active page cache warmup Enterprise edition

Neo4j Enterprise Edition has a feature called *active page cache warmup*, which is enabled by default via the `db.memory.pagecache.warmup.enable` configuration setting.

How it works

It shortens the page fault spike and makes the page cache warm up faster. This is done by periodically recording *cache profiles* of the store files while the database is running. These profiles contain information about what data is and is not in memory and are stored in the *data/databases/mydatabase/profiles* directory. When Neo4j is restarted next time, it looks for these cache profiles and loads the same data that was in memory when the profile was created. The profiles are also copied as part of the online backup and cluster store-copy operations and help warm up new databases that join a cluster.

The setting should remain enabled for most scenarios. However, when the workload changes after the database restarts, the setting can be disabled to avoid spending time fetching data that will be directly evicted.

Configuration options

Load the entire database into memory

It is also possible to configure `db.memory.pagecache.warmup.preload` to load the entire database data into memory. This is useful when the size of the database store is smaller than the available memory for the page cache. When enabled, it disables warmup by profile and prefetches data into the page cache as part of the startup.

Load specified files into memory

The files that you want to prefetched can be filtered using the `db.memory.pagecache.warmup.preload.allowlist` setting. It takes a regular expression as a value to match the files.

Example 86. Load only the nodes and relationships

For example, if you want to load only the nodes and relationships, you can use the regex `.*(node|relationship).*` to match the name of the store files. The *active page cache warmup* will prefetch the content of the following files:

```
neostore.nodestore.db
neostore.nodestore.db.id
neostore.nodestore.db.labels
neostore.nodestore.db.labels.id
neostore.relationshipgroupstore.db
neostore.relationshipgroupstore.db.id
neostore.relationshipstore.db
neostore.relationshipstore.db.id
neostore.relationshiptypestore.db
neostore.relationshiptypestore.db.id
neostore.relationshiptypestore.db.names
Neostore.relationshiptypestore.db.names.id
```

And can be verified using unix `grep`:

```
ls neo4j/ | grep -E '.*(node|relationship).*'
```

Configure the profile frequency for the page cache

The profile frequency is the rate at which the profiles are re-generated. More frequent means more accurate. A profile contains information about those parts of the files that are currently loaded into memory. By default, it is set to `db.memory.pagecache.warmup.profile.interval=1m`. It takes some time to generate these profiles, and therefore `1m` is a good interval. If the workload is very stable, then the profile will not change much. Accordingly, if the workload changes often, the profile will thus often become outdated.

Checkpoint IOPS limit 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 `db.checkpoint.iops.limit` restricts the IO bandwidth that the checkpoint process is allowed to use. Each IO is, in the case of the checkpoint process, an 8 KiB write. An IOPS limit of 600, for instance, would thus only allow the checkpoint process to write at a rate of roughly 5 MiB per second. This will, on the other hand, make checkpoints take longer to complete. A longer time between checkpoints can cause more transaction log data to accumulate, and can lengthen recovery times. See the 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.

Statistics and execution plans

When a Cypher query is issued, it gets compiled to an execution plan that can run and answer the query. The Cypher query engine uses the available information about the database, such as schema information about which indexes and constraints exist in the database. This page describes how to configure the Neo4j statistics collection and the query replanning in the Cypher query engine.

ℹ	Neo4j also uses statistical information about the database to optimize the execution plan. For more information, see Cypher Manual → Execution plans.

Configure statistics collection

The Cypher query planner depends on accurate statistics to create efficient plans. Therefore, these statistics are kept up-to-date as the database evolves.

For each database in the DBMS, Neo4j collects the following statistical information and keeps it up-to-date:

For graph entities

- The number of nodes with a certain label.
- The number of relationships by type.
- The number of relationships by type between nodes with a specific label.

These numbers are updated whenever you set or remove a label from a node.

For database schema

- Selectivity per index.

To produce a selectivity number, Neo4j runs a full index scan in the background. Because this could potentially be a very time-consuming operation, a full index scan is triggered only when the changed data reaches a specified threshold.

Automatic statistics collection

You can control whether and how often statistics are collected automatically by configuring the following settings:

Parameter name	Default value	Description
`db.index_sampling.background_enabled`	`true`	Enable the automatic (background) index sampling.
`db.index_sampling.update_percentage`	`5`	Percentage of index updates of total index size required before sampling of a given index is triggered.

Manual statistics collection

You can manually trigger index resampling by using the built-in procedures `db.resampleIndex()` and `db.resampleOutdatedIndexes()`.

`db.resampleIndex()`

Trigger resampling of a specified index.

```
CALL db.resampleIndex("indexName")
```

`db.resampleOutdatedIndexes()`

Trigger resampling of all outdated indexes.

```
CALL db.resampleOutdatedIndexes()
```

Configure the replanning of execution plans

Execution plans are cached and are not replanned until the statistical information used to produce the plan changes.

Automatic replanning

You can control how sensitive the replanning should be to database updates by configuring the following settings:

Parameter name	Default value	Description
`dbms.cypher.statistics_divergence_threshold`	`0.75`	The threshold for statistics above which a plan is considered stale. When the changes to the underlying statistics of an execution plan meet the specified threshold, the plan is considered stale and is replanned. Change is calculated as `abs(a-b)/max(a,b)`. This means that a value of `0.75` requires the database to approximately quadruple in size before replanning occurs. A value of `0` means that the query is replanned as soon as there is a change in the statistics and the replan interval elapses.
`dbms.cypher.min_replan_interval`	`10s`	The minimum amount of time between two query replanning executions. After this time, the graph statistics are evaluated, and if they have changed more than the value set in `dbms.cypher.statistics_divergence_threshold`, the query is replanned. Each time the statistics are evaluated, the divergence threshold is reduced until it reaches 10% after about 7h. This ensures that even moderately changing databases see query replanning after a sufficiently long time interval.

Manual replanning

You can manually force the database to replan the execution plans that are already in the cache by using the following built-in procedures:

`db.clearQueryCaches()`

Clear all query caches. Does not change the database statistics.

```
CALL db.clearQueryCaches()
```

`db.prepareForReplanning()`

Completely recalculates all database statistics to be used for any subsequent query planning.

The procedure triggers an index resampling, waits for it to complete, and clears all query caches. Afterwards, queries are planned based on the latest database statistics.

```
CALL db.prepareForReplanning()
```

You can use Cypher replanning to specify whether you want to force a replan, even if the plan is valid according to the planning rules, or skip replanning entirely should you wish to use a valid plan that already exists.

For more information, see:

- Cypher manual → Cypher replanning
- Cypher manual → Execution plans
- Procedures

Space reuse

Neo4j uses logical deletes to remove data from the database to achieve maximum performance and scalability. A logical delete means that all relevant records are marked as deleted, but the space they occupy is not immediately returned to the operating system. Instead, it is subsequently reused by the transactions *creating* data.

Marking a record as deleted requires writing a record update command to the transaction log, 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.

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.

Reclaim unused space

You can use the `neo4j-admin database copy` command to create a defragmented copy of your database. The `copy` command creates and entirely new and independent database. If you want to run that database in a cluster, you have to re-seed the existing cluster, or seed a new cluster from that copy.

Example 87. Example of database compaction using `neo4j-admin database copy`

The following is a detailed example on how to check your database store usage and how to reclaim space.

Let's use the Cypher Shell command-line tool to add 100k nodes and then see how much store they occupy.

1. In a running Neo4j standalone instance, log in to the Cypher Shell command-line tool with your credentials.

```
$neo4j-home/bin$> ./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 database copy command to create a defragmented copy of your database. Use the `system` database and stop the `neo4j` database before running the command.

1. Invoke the `neo4j-admin database copy` command to create a copy of your `neo4j` database.

```
$neo4j-home/bin$> ./neo4j-admin database copy neo4j neo4jcopy1 --compact-node-store --verbose
```

```
Starting to copy store, output will be saved to: $neo4j_home/logs/neo4j-admin-copy-2020-11-
04.11.30.57.log
2020-10-23 11:40:00.749+0000 INFO [StoreCopy] ### Copy Data ###
2020-10-23 11:40:00.750+0000 INFO [StoreCopy] Source: $neo4j_home/data/databases/neo4j (page
cache 8m) (page cache 8m)
2020-10-23 11:40:00.750+0000 INFO [StoreCopy] Target: $neo4j_home/data/databases/neo4jcopy1 (page
cache 8m)
2020-10-23 11:40:00.750+0000 INFO [StoreCopy] Empty database created, will start importing
readable data from the source.
2020-10-23 11:40:02.397+0000 INFO [o.n.i.b.ImportLogic] Import starting
Nodes, started 2020-11-04 11:31:00.088+0000
[*Nodes:?? 7.969MiB-----------------------------------------------------------------------------]
100K Δ 100K
Done in 632ms
Prepare node index, started 2020-11-04 11:31:00.735+0000
[*DETECT:7.969MiB-------------------------------------------------------------------------------]
0 Δ    0
Done in 79ms
Relationships, started 2020-11-04 11:31:00.819+0000
[*Relationships:?? 7.969MiB---------------------------------------------------------------------]
0 Δ    0
Done in 37ms
Node Degrees, started 2020-11-04 11:31:01.162+0000
[*>:??------------------------------------------------------------------------------------------]
0 Δ    0
Done in 12ms
Relationship --> Relationship 1/1, started 2020-11-04 11:31:01.207+0000
[*>:??------------------------------------------------------------------------------------------]
0 Δ    0
Done in 0ms
RelationshipGroup 1/1, started 2020-11-04 11:31:01.232+0000
[*>:??------------------------------------------------------------------------------------------]
0 Δ    0
Done in 10ms
Node --> Relationship, started 2020-11-04 11:31:01.245+0000
[*>:??------------------------------------------------------------------------------------------]
0 Δ    0
Done in 10ms
Relationship <-- Relationship 1/1, started 2020-11-04 11:31:01.287+0000
[*>:??------------------------------------------------------------------------------------------]
0 Δ    0
Done in 0ms
Count groups, started 2020-11-04 11:31:01.549+0000
[*>:??------------------------------------------------------------------------------------------]
0 Δ    0
Done in 0ms
Node --> Group, started 2020-11-04 11:31:01.579+0000
[*>:??------------------------------------------------------------------------------------------]
0 Δ    0
Done in 1ms
Node counts and label index build, started 2020-11-04 11:31:01.986+0000
[*>:??------------------------------------------------------------------------------------------]
0 Δ    0
Done in 11ms
Relationship counts, started 2020-11-04 11:31:02.034+0000
[*>:??------------------------------------------------------------------------------------------]
0 Δ    0
Done in 0ms

IMPORT DONE in 3s 345ms.
Imported:
  0 nodes
  0 relationships
  0 properties
Peak memory usage: 7.969MiB
2020-11-04 11:31:02.835+0000 INFO [o.n.i.b.ImportLogic] Import completed successfully, took 3s
345ms. Imported:
  0 nodes
  0 relationships
  0 properties
2020-11-04 11:31:03.330+0000 INFO [StoreCopy] Import summary: Copying of 100704 records took 5
seconds (20140 rec/s). Unused Records 100704 (100%) Removed Records 0 (0%)
2020-11-04 11:31:03.330+0000 INFO [StoreCopy] ### Extracting schema ###
2020-11-04 11:31:03.330+0000 INFO [StoreCopy] Trying to extract schema...
2020-11-04 11:31:03.338+0000 INFO [StoreCopy] ... found 0 schema definitions.
```

The example resulted in a compact and consistent store (any inconsistent nodes, properties, relationships are not copied over to the newly created store).

2. Use the `system` database and create the `neo4jcopy1` database.

```
neo4j@system> create database neo4jcopy1;
```

```
0 rows available after 60 ms, consumed after another 0 ms
```

3. Verify that the `neo4jcopy1` database is online.

```
neo4j@system> show databases;
```

```
+-----------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
| name         | type       | aliases | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage | default | home   | constituents |
+-----------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
| "neo4j"      | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   |
"offline"        | "offline"     | ""            | TRUE    | TRUE   | []           |
| "neo4jcopy1" | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   |
"online"         | "online"      | ""            | FALSE   | FALSE | []           |
| "system"     | "system"   | []      | "read-write" | "localhost:7687" | "primary" | TRUE   |
"online"         | "online"      | ""            | FALSE   | FALSE | []           |
+-----------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+

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

4. In Neo4j Browser, run `:sysinfo` to check the total store size of `neo4jcopy1`.

   The reported output for the store size after the compaction is 800.68 KiB, ID Allocation: Node ID 0, Property ID 0.

Monitoring

Neo4j provides mechanisms for continuous analysis through the output of metrics as well as the inspection and management of currently-executing queries.

Logs can be harvested for continuous analysis, or for specific investigations. Facilities are available for producing security event logs as well as query logs. The query management functionality is provided for specific investigations into query performance. Monitoring features are also provided for ad-hoc analysis of a Causal Cluster.

This chapter describes the following:

Logging

Logging framework

Neo4j provides logs for monitoring purposes. As of Neo4j 5.0, Neo4j uses Log4j 2 for logging. Log4j 2 can be configured using the two XML files *conf/server-logs.xml* and *conf/user-log.xml*. The XML files are passed to Log4j.

Neo4j supports all default Log4j components. You can also add plugins to Log4j by dropping them in the *plugin* directory. For more advanced usages, see the Log4j official documentation.

The following is a description of how a default Neo4j installation behaves and some typical examples of configuring it to use Log4j.

Log files

By default, the directory where the general log files are located is configured by `server.directories.logs`.

The following table describes the Neo4j general log files and the information they contain.

Table 46. Neo4j logs for monitoring

Filename	Name	Description
neo4j.log	*The user log*	Logs general information about Neo4j. For Debian and RPM packages run `journalctl --unit=neo4j`.
debug.log	*The debug log*	Logs information that is useful when debugging problems with Neo4j.
http.log	*The HTTP log*	Logs information about the HTTP API.
gc.log	*The garbage collection log*	Logs information provided by the JVM.
query.log	*The query log*	Enterprise Logs information about queries that run longer than a specified threshold.
security.log	*The security log*	Enterprise Logs information about security events.
service-out.log	*The windows service log*	Windows Log of the console output when installing or running the Windows service.
service-error.log	*The windows service log*	Windows Logs information about errors encountered when installing or running the Windows service.

Logging configuration file anatomy

The logging configuration is done using a Log4j 2 XML configuration file.

Neo4j has two different configuration files, one for the *neo4j.log*, which contains general information about Neo4j, and one configuration file for all other types of logging (except *gc.log*, see GC Logging).

Where the files are located and which logs are enabled or disabled is managed via the *neo4j.conf* file.

Table 47. Log configuration settings in neo4j.conf

Configuration setting	Default value	Description
`server.logs.config`	`conf/server-logs.xml`	Path to the XML configuration file for *debug.log*, *http.log*, *query.log*, and *security.log*.
`server.logs.user.config`	`conf/user-logs.xml`	Path to the XML configuration file for *neo4j.log* file.
`server.logs.debug.enabled`	`true`	Enable/disable the *debug.log*. It is highly recommended to keep it enabled.
`dbms.logs.http.enabled`	`false`	Enable/disable the *http.log*.
`db.logs.query.enabled`	`VERBOSE`	Must be one of `OFF`, `INFO`, or `VERBOSE`. `INFO` produces less output than `VERBOSE`, while `OFF` completely disables the *query.log*.

The Log4j 2 XML configuration file consists of 3 major components:

- *Appenders* — define output locations, for example, a file, the console, network socket, etc.
- *Layouts* — define how the output is formatted, for example, plain text, JSON, CSV, etc.
- *Loggers* — route log events to one or several appenders.

You can also configure filters to determine if and what log events to be published and how. For more information, see the Log4j official documentation.

An illustrative Log4j 2 configuration file

```
 1 <Configuration monitorInterval="30" packages="org.neo4j.logging.log4j">
 2     <Appenders>
 3         <RollingFile name="DebugLog" fileName="${config:server.directories.logs}/debug.log"
 4                      filePattern="${config:server.directories.logs}/debug.log.%02i">
 5             <Neo4jDebugLogLayout
 6                 pattern="%d{yyyy-MM-dd HH:mm:ss.SSSZ}{GMT+2} %-5p [%c{1.}] %m%n"/>
 7             <Policies>
 8                 <SizeBasedTriggeringPolicy size="20 MB"/>
 9             </Policies>
10             <DefaultRolloverStrategy fileIndex="min" max="7"/>
11         </RollingFile>
12
13         <RollingFile name="HttpLog" fileName="${config:server.directories.logs}/http.log"
14                      filePattern="${config:server.directories.logs}/http.log.%02i">
15             <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSSZ}{GMT+0} %-5p %m%n"/>
16             <Policies>
17                 <SizeBasedTriggeringPolicy size="20 MB"/>
18             </Policies>
19             <DefaultRolloverStrategy fileIndex="min" max="5"/>
20         </RollingFile>
21     </Appenders>
22
23     <Loggers>
24         <Root level="INFO">
25             <AppenderRef ref="DebugLog"/>
26         </Root>
27
28         <Logger name="HttpLogger" level="INFO" additivity="false">
29             <AppenderRef ref="HttpLog"/>
30         </Logger>
31     </Loggers>
32 </Configuration>
```

Table 48. Breakdown of the illustrative Log4j 2 configuration file line per line

Line(s)	Description
1	Configuration tag with a `monitorInterval` of 30 seconds and a package namespace of `org.neo4j.logging.log4j`. The monitor interval tells Log4j to periodically check the XML file for changes and reload the file if a change is detected. The package namespace gives access to the Neo4j configuration lookup with `${config:<setting>}`.
3, 13	Defines two `<RollingFile>` appenders. One writes to *debug.log* and one to *http.log*. The `name` must be unique and is used later when referencing the appender.
4, 14	Specifies a `filePattern` to be used when the file is rolled. The pattern renames the files to *debug.log.01* and *http.log.01* when they reach the defined trigger.
5	Defines the layout to use for the `DebugLog` appender, in this case with the `GMT+2` timezone.
8, 17	Defines a size-based trigger. When the size of the files reaches 20 MB, the files are renamed according to the `filePattern`, and the log files start over.
10	Defines a rollover strategy, keeping 7 files as history. `fileIndex=min` implies that the minimum/the lowest number is the most recent one.
24, 25	Defines a root logger with log level `INFO`. The root logger is a "catch-all" logger that captures everything that is not captured by the other loggers. Everything caught is sent to the appender with the name `DebugLog`.
28, 29	Defines a logger that matches log events with the `HttpLogger` target with a log level of `INFO` or above. The `additivity="false"` is set to fully consume the log event. If `additivity="true"` is set, which is the default, the log event is also sent to the root logger.

Appenders

An appender represents a destination for log events. All Log4j standard appenders are available in Neo4j. For more details, see the Log4j official documentation. A few of the most common appenders are `<Console>`, `<RollingFile>`, and `<RollingRandomAccessFile>`.

`<Console>` appender

The console appender outputs log events to *stdout* or *stderr*.

An example of a console appender

```
<Console name="console" target="SYSTEM_OUT"> <!-- or SYSTEM_ERR -->
  <PatternLayout pattern="%m%n"/>
</Console>
```

`<RollingFile>` appender

A rolling file appender writes log events to a file. It rolls when certain criteria are met. A standard scheme is to keep one log file daily or roll a log file once a specific size is reached.

An example of a rolling file appender with one new log file each day

```
<RollingFile name="myLog" fileName="${config:server.directories.logs}/my.log"
                       filePattern="${config:server.directories.logs}/my-%d{yyyy-MM-dd}.log">
  <!-- Layout -->
  <Policies>
      <TimeBasedTriggeringPolicy />
  </Policies>
</RollingFile>
```

The rolling also supports compression of rolled-out files. Adding one of `.gz`, `.zip`, `.bz2`, `.deflate`, or `.pack200` as a suffix to the `filePattern` attribute causes the file to be compressed with the appropriate compression scheme.

An example of a rolling file appender with zip compression

```
<RollingFile name="myLog" fileName="${config:server.directories.logs}/my.log"
                       filePattern="${config:server.directories.logs}/my.%i.log.zip">
  <!-- Layout -->
  <Policies>
      <SizeBasedTriggeringPolicy size="20 MB"/>
  </Policies>
</RollingFile>
```

`<RollingRandomAccessFile>` appender

The `<RollingRandomAccessFile>` is almost identical to the `<RollingFile`, with one major exception - all writes are buffered.

The drawback of using this appender is that log events might **not** be visible immediately and can be lost in case of a system crash. If these drawbacks are acceptable, switching to `<RollingRandomAccessFile>` can increase the performance of the logging.

Log layouts

The log files can be written in a lot of different ways, referred to as layouts. Neo4j comes bundled with all the default layouts of Log4j 2, as well as a few Neo4j-specific ones. For more details on the default Log4j 2 layouts, see the Log4j official documentation.

`<PatternLayout>`

The most common layout. The pattern layout is a flexible layout configurable with a pattern string. The pattern consists of different converters that are prefixed with `%`. An example pattern could be

```
<PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSSZ}{GMT+0} %-5p [%c{1.}] %m%n"/>
```

It contains the following converters:

Table 49. Example pattern layout converters

Converter	Description
`%d{date-pattern}{timezone}`	Date of the log event. The time zone is optional. If omitted, the system time is used.
`%p`	The log level of the event. Can be `DEBUG`, `INFO`, `WARN`, or `ERROR`. Adding `-5` between the `%` symbol and the `p` pads the level to be exactly 5 characters long.
`%c`	The class where the log event originated from. Adding `{1.}` after compacts the package names, e.g. `org.apache.commons.Foo` will become `o.a.c.Foo`.
`%m`	The log message of the log event.
`%n`	System-specific new line.

These are just a few examples. For all available converters, consult the Log4j 2 Pattern Layout documentation.

`<Neo4jDebugLogLayout>`

The Neo4j debug log layout is essentially the same as the `PatternLayout`. The main difference is that a header is injected at the start of the log file with diagnostic information useful for Neo4j developers. This layout should typically only be used for the *debug.log* file.

An example usage of the Neo4j debug log layout

```
<Neo4jDebugLogLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSSZ}{GMT+0} %-5p [%c{1.}] %m%n"/>
```

`<JsonTemplateLayout>`

The `<JsonTemplateLayout>` is equivalent to the pattern layout. For more information, see the Log4j official documentation.

The JSON template layout takes an event template, which can be a file residing on the file system, for example:

```
<JsonTemplateLayout eventTemplateUri="file://path/to/template.json"/>
```

Alternatively, you can embed it right into the XML:

```
<JsonTemplateLayout>
  <eventTemplate>
    <![CDATA[
      {
        "time": { "$resolver": "timestamp",
          "pattern": { "format": "yyyy-MM-dd HH:mm:ss.SSSZ", "timeZone": "UTC" }
        },
        "level": { "$resolver": "level", "field": "name" },
        "message": { "$resolver": "message" },
        "includeFullMap": { "$resolver": "map", "flatten": true },
        "stacktrace": { "$resolver": "exception", "field": "message" }
      }
    ]]>
  </eventTemplate>
</JsonTemplateLayout>
```

There are also a couple of built-in templates available from the classpath, for example:

```
<JsonTemplateLayout eventTemplateUri="classpath:org/neo4j/logging/StructuredJsonLayout.json"/>
```

Table 50. Available built-in templates

eventTemplateUri	Description
`classpath:org/neo4j/logging/StructuredJsonLayout.json`	Layout for structured log messages. Only applicable to the *query.log* and *security.log*.
`classpath:org/neo4j/logging/StructuredLayoutWithMessage.json`	Generic layout for logging JSON messages. Can be used for any log file.
`classpath:org/neo4j/logging/QueryLogJsonLayout.json`	Backward compatible JSON layout that will match the Neo4j 4.x query log.
`classpath:LogstashJsonEventLayoutV1.json`	Logstash `json_event` pattern for Log4j

eventTemplateUri	Description
`classpath:GelfLayout.json`	Graylog Extended Log Format (GELF) payload specification with additional `_thread` and `_logger` fields.
`classpath:GcpLayout.json`	Google Cloud Platform structured logging with additional `_thread`, `_logger`, and `_exception` fields.
`classpath:JsonLayout.json`	Same layout as the less flexible `<JsonLayout>`.

Loggers

Loggers forward log events to appenders. There can be an arbitrary number of `<Logger>` elements but only one `<Root>` logger element. Loggers have the possibility of being additive. An additive logger forwards a log event to its appender(s) and then passes the log event to the next matching logger. A non-additive logger forwards a log event to its appender(s) and then drops the event. The root logger is a special logger that matches everything, so if another logger does not pick up a log event, the root logger will. Therefore, it is best practice always to include a root logger so that no log events are missed.

Configuration of loggers

```xml
<Configuration>
    <!-- Appenders -->
    <Loggers>
        <Root level="WARN">
            <AppenderRef ref="DebugLog"/>
        </Root>

        <Logger name="HttpLogger" level="INFO" additivity="false">
            <AppenderRef ref="HttpLog"/>
        </Logger>
    </Loggers>
</Configuration>
```

A logger has a `level` that filters log events. A level can also include levels of different severity. For example, a logger with `level="INFO"` forwards log events with `INFO`, `WARN`, and `ERROR`. A logger with `level="WARN"` only logs `WARN` and `ERROR` events.

The following table lists all log levels raised by Neo4j and their severity level:

Table 51. Log levels

Message type	Severity level	Description
`DEBUG`	Low severity	Report details on the raised errors and possible solutions.
`INFO`	Low severity	Report status information and errors that are not severe.
`WARN`	Low severity	Report errors that need attention but are not severe.

Message type	Severity level	Description
ERROR	High severity	Reports errors that prevent the Neo4j server from running and must be addressed immediately.

For more details on loggers, see the Log4j official documentation → Configuring Loggers.

Garbage collection log

The garbage collection log, or GC log for short, is special and cannot be configured with Log4j 2. The GC log is handled by the Java Virtual Machine(JVM) and must be passed directly to the command line. To simplify this process, Neo4j exposes the following settings in *neo4j.conf*:

Table 52. Garbage collection log configurations

The *garbage collection log* configuration	Default value	Description
`server.logs.gc.enabled`	`false`	Enable garbage collection logging.
`server.logs.gc.options`	`-Xlog:gc*,safepoint,age*=trace`	Garbage collection logging options. For available options, consult the documentation of the JVM distribution used.
`server.logs.gc.rotation.keep_number`	`5`	The maximum number of history files for the garbage collection log.
`server.logs.gc.rotation.size`	`20MB`	The threshold size for rotation of the garbage collection log.

Security log Enterprise edition

Neo4j provides security event logging that records all security events. The security log is enabled automatically when the configuration `dbms.security.auth_enabled` is set to `true`.

For native user management, the following actions are recorded:

- Login attempts — by default, both successful and unsuccessful logins are recorded.
- All administration commands run against the `system` database.
- Authorization failures from role-based access control.

If using LDAP as the authentication method, some cases of LDAP misconfiguration will also be logged, as well as the LDAP server communication events and failures.

If many programmatic interactions are expected, it is advised to disable the logging of successful logins by setting the `dbms.security.log_successful_authentication` parameter in the *neo4j.conf* file:

```
dbms.security.log_successful_authentication=false
```

The following information is available in the JSON format:

Table 53. JSON format log entries

Name	Description
`time`	The timestamp of the log message.
`level`	The log level.
`type`	It is always `security`.
`source`	Connection details.
`database`	The database name the command is executed on.
`username`	The user connected to the security event. This field is deprecated by `executingUser`.
`executingUser`	The name of the user triggering the security event. Either same as `authenticatedUser` or an impersonated user.
`authenticatedUser`	The name of the user who authenticated and is connected to the security event.
`message`	The log message.
`stacktrace`	Included if there is a stacktrace associated with the log message.

An example of the security log in a plain format:

```
2019-12-09 13:45:00.796+0000 INFO  [johnsmith]: logged in
2019-12-09 13:47:53.443+0000 ERROR [johndoe]: failed to log in: invalid principal or credentials
2019-12-09 13:48:28.566+0000 INFO  [johnsmith]: CREATE USER janedoe SET PASSWORD '******' CHANGE REQUIRED
2019-12-09 13:48:32.753+0000 INFO  [johnsmith]: CREATE ROLE custom
2019-12-09 13:49:11.880+0000 INFO  [johnsmith]: GRANT ROLE custom TO janedoe
2019-12-09 13:49:34.979+0000 INFO  [johnsmith]: GRANT TRAVERSE ON GRAPH * NODES A, B (*) TO custom
2019-12-09 13:49:37.053+0000 INFO  [johnsmith]: DROP USER janedoe
2019-12-09 13:52:24.685+0000 INFO  [johnsmith:alice]: impersonating user alice logged in
```

Query log Enterprise edition

Query logging is enabled by default and is controlled by the setting `db.logs.query.enabled`. It helps you analyze long-running queries and does not impact system performance. The default is to log all queries, but it is recommended to log for queries exceeding a certain threshold.

The following configuration settings are available for the *query log*:

Table 54. Query log enabled setting

Option	Description
`OFF`	Completely disable logging.
`INFO`	Log at the end of queries that have either succeeded or failed. The `db.logs.query.threshold` parameter is used to determine the threshold for logging a query. If the execution of a query takes longer than this threshold, the query is logged. Setting the threshold to `0s` results in all queries being logged.
`VERBOSE`	Default Log all queries at both start and finish, regardless of `db.logs.query.threshold`.

The following configuration settings are available for the query log file:

Table 55. Query log configurations

The *query log* configuration	Default value	Description
`db.logs.query.early_raw_logging_enabled`	`false`	Log query text and parameters without obfuscating passwords. This allows queries to be logged earlier before parsing starts.
`db.logs.query.enabled`	`VERBOSE`	Log executed queries.
`db.logs.query.max_parameter_length`	`2147483647`	This configuration option allows you to set a maximum parameter length to include in the log. Parameters exceeding this length will be truncated and appended with `...`. This applies to each parameter in the query.

The *query log* configuration	Default value	Description
`db.logs.query.obfuscate_literals`	`false`	If `true`, obfuscates all query literals before writing to the log. This is useful when Cypher queries expose sensitive information. Node labels, relationship types, and map property keys are still shown. Changing the setting does not affect cached queries. Therefore, if you want the switch to have an immediate effect, you must also clear the query cache; `CALL db.clearQueryCaches()`. This does not obfuscate literals in parameters. If parameter values are not required in the log, set `db.logs.query.parameter_logging_enabled=false`.
`db.logs.query.parameter_logging_enabled`	`true`	Log parameters for the executed queries being logged. You can disable this configuration setting if you do not want to display sensitive information.

The *query log* configuration	Default value	Description
`db.logs.query.plan_description_enabled`	`false`	This configuration option allows you to log the query plan for each query. The query plan shows up as a description table and is useful for debugging purposes. Every time a Cypher query is run, it generates and uses a plan for the execution of the code. The plan generated can be affected by changes in the database, such as adding a new index. As a result, it is not possible to historically see what plan was used for the original query execution. Enabling this option has a performance impact on the database due to the cost of preparing and including the plan in the *query log*. It is not recommended for everyday use.
`db.logs.query.threshold`	`0s`	If the query execution takes longer than this threshold, the query is logged once completed (provided query logging is set to `INFO`). A threshold of `0` seconds logs all queries.
`db.logs.query.transaction.enabled`	`OFF`	Track the start and end of a transaction within the query log. Log entries are written to the *query log*. They include the transaction ID for a specific query and the start and end of a transaction. You can also choose a level of logging (`OFF`, `INFO`, or `VERBOSE`). If `INFO` is selected, you must exceed the time before the log is written (`db.logs.query.transaction.threshold`).

The *query log* configuration	Default value	Description
`db.logs.query.transaction.threshold`	`0s`	If the transaction is open for longer than this threshold (duration of time), the transaction is logged once completed, provided transaction logging is set to `INFO`. Defaults to `0` seconds, which means all transactions are logged. This can be useful when identifying where there is a significant time lapse after query execution and transaction commits, especially in performance analysis around locking.

Example 88. Configure for simple query logging

In this example, the query logging is set to `INFO`, and all other query log parameters are at their defaults.

```
db.logs.query.enabled=INFO
```

The following is an example of the query log with this basic configuration:

```
2017-11-22 14:31 ... INFO  9 ms: bolt-session   bolt    johndoe neo4j-javascript/1.4.1
client/127.0.0.1:59167  ...
2017-11-22 14:31 ... INFO  0 ms: bolt-session   bolt    johndoe neo4j-javascript/1.4.1
client/127.0.0.1:59167  ...
2017-11-22 14:32 ... INFO  3 ms: server-session http    127.0.0.1    /db/data/cypher neo4j - CALL
dbms.procedures() - {}
2017-11-22 14:32 ... INFO  1 ms: server-session http    127.0.0.1    /db/data/cypher neo4j - CALL
dbms.showCurrentUs...
2017-11-22 14:32 ... INFO  0 ms: bolt-session   bolt    johndoe neo4j-javascript/1.4.1
client/127.0.0.1:59167  ...
2017-11-22 14:32 ... INFO  0 ms: bolt-session   bolt    johndoe neo4j-javascript/1.4.1
client/127.0.0.1:59167  ...
2017-11-22 14:32 ... INFO  2 ms: bolt-session   bolt    johndoe neo4j-javascript/1.4.1
client/127.0.0.1:59261  ...
```

Example 89. Configure for query logging with more details

In this example, the query log is enabled, as well as some additional logging:

```
db.logs.query.enabled=INFO
db.logs.query.parameter_logging_enabled=true
db.logs.query.threshold=<appropriate value>
```

The following sample query is run on the Movies database:

```
MATCH (n:Person {name:'Tom Hanks'})-[:ACTED_IN]->(n1:Movie)<-[:DIRECTED]-(n2:Person {name:"Tom
Hanks"}) RETURN n1.title
```

The corresponding query log in <.file>*query.log* is:

```
2017-11-23 12:44:56.973+0000 INFO  1550 ms: (planning: 20, cpu: 920, waiting: 10) - 13792 B - 15 page
hits, 0 page faults - bolt-session    bolt    neo4j    neo4j-javascript/1.4.1
client/127.0.0.1:58189  server/127.0.0.1:7687>  neo4j - match (n:Person {name:'Tom Hanks'})-
[:ACTED_IN]->(n1:Movie)<-[:DIRECTED]-(n2:Person {name:"Tom Hanks"}) return n1.title; - {} - {}
```

An obvious but essential point of note when examining parameters of a particular query is to ensure you analyze only the entries relevant to that specific query plan, as opposed to, e.g., CPU, time, bytes, and so on for each log entry in sequence.

Following is a breakdown of resource usage parameters with descriptions corresponding to the above query:

`2017-11-23 12:44:56.973+0000`
: Log timestamp.

`INFO`
: Log category.

`1550 ms`
: Total elapsed cumulative wall time spent in query execution. It is the total of planning time + CPU + waiting + any other processing time, e.g., taken to acquire execution threads. This figure is cumulative for every time a CPU thread works on executing the query.

`Planning`
: Refers to the time the Cypher engine takes to create a query plan. Plans may be cached for repetitive queries, and therefore, planning times for such queries will be shorter than those for previously unplanned ones. In the example, this contributed 20ms to the total execution time of 1550ms.

`CPU time`
: Refers to the time taken by the individual threads executing the query, e.g., a query is submitted at 08:00. It uses CPU for 720ms, but then the CPU swaps out to another query, so the first query is no longer using the CPU. Then, after 100ms, it gets/uses the CPU again for 200ms (more results to be loaded, requested by the client via the Driver), then the query completes at 08:01:30, so the total duration is 1550ms (includes some round-trip time for 2 round-trips), and CPU is

720+200=920ms.

`Waiting`

Time a query spent waiting before execution (in ms), for example, if an existing query has a lock which the new query must wait to release. In the example, this contributed 10ms to the total execution time of 1550ms.
It is important to note that the client requests data from the server only when its record buffer is empty (one round-trip from the server may end up with several records), and the server stops pushing data into outgoing buffers if the client does not read them in a timely fashion. Therefore, it depends on the size of the result set. If it is relatively small and fits in a single round-trip, the client receives all the results at once, and the server finishes processing without any client-side effect. Meanwhile, if the result set is large, the client-side processing time will affect the overall time, as it is directly connected to when new data is requested from the server.

`13792 B`

The logged allocated bytes for the executed queries. This is the amount of HEAP memory used during the life of the query. The logged number is cumulative over the duration of the query, i.e., for memory-intense or long-running queries, the value may be larger than the current memory allocation.

`15 page hits`

Page hit means the result was returned from page cache as opposed to disk. In this case, the page cache was hit 15 times.

`0 page faults`

Page fault means that the query result data was not in the `dbms.memory.pagecache` and, therefore, had to be fetched from the file system. In this case, query results were returned entirely from the 8 page cache hits mentioned above, so there were 0 hits on the disk required.

`bolt-session`

The session type.

`bolt`

The Browser ↔ database communication protocol used by the query.

`neo4j`

The process ID.

`neo4j-javascript/1.4.1`

The Driver version.

`client/127.0.0.1:52935`

The query client outbound `IP:port` used.

`server/127.0.0.1:7687>`

The server listening `IP:port` used.

`neo4j`

username of the query executioner

`match (n:Person {name:'Tom Hanks'})-[:ACTED_IN]→(n1:Movie)←[:DIRECTED]-(n2:Person {name:"Tom Hanks"}) return n1.title`

The executed query.

The last two parenthesis `{}` `{}` are for the query parameters and `txMetaData`.

Attach metadata to a transaction

You can attach metadata to a transaction and have it printed in the query log using the built-in procedure `tx.setMetaData`.

Neo4j Drivers also support attaching metadata to a transaction. For more information, see the respective Driver's manual.

Every graph app should follow a convention for passing metadata with the queries that it sends to Neo4j:

```
{
  app: "neo4j-browser_v4.4.0", ①
  type: "system" ②
}
```

① `app` can be a user-agent styled-name plus version.

② `type` can be one of:

- `system` — a query automatically run by the app.
- `user-direct` — a query the user directly submitted to/through the app.
- `user-action` — a query resulting from an action the user performed.
- `user-transpiled` — a query that has been derived from the user input.

This is typically done programmatically but can also be used with the Neo4j dev tools.
In general, you start a transaction on a user database and attach a list of metadata to it by calling `tx.setMetaData`. You can also use the procedure `CALL tx.getMetaData()` to show the metadata of the current transaction. These examples use the MovieGraph dataset from the Neo4j Browser guide.

Example 90. Using `cypher-shell`, attach metadata to a transaction

Cypher Shell always adds metadata that follows the convention by default. In this example, the defaults are overridden.

```
neo4j@neo4j> :begin
neo4j@neo4j# CALL tx.setMetaData({app: 'neo4j-cypher-shell_v.4.4.0', type: 'user-direct', user:
'jsmith'});
0 rows
ready to start consuming query after 2 ms, results consumed after another 0 ms
neo4j@neo4j# CALL tx.getMetaData();
+--------------------------------------------------------------------------+
| metadata                                                                 |
+--------------------------------------------------------------------------+
| {app: "neo4j-cypher-shell_v.4.4.0", type: "user-direct", user: "jsmith"} |
+--------------------------------------------------------------------------+

1 row
ready to start consuming query after 37 ms, results consumed after another 2 ms
neo4j@neo4j# MATCH (n:Person) RETURN n  LIMIT 5;
+----------------------------------------------------+
| n                                                  |
+----------------------------------------------------+
| (:Person {name: "Keanu Reeves", born: 1964})       |
| (:Person {name: "Carrie-Anne Moss", born: 1967})   |
| (:Person {name: "Laurence Fishburne", born: 1961}) |
| (:Person {name: "Hugo Weaving", born: 1960})       |
| (:Person {name: "Lilly Wachowski", born: 1967})    |
+----------------------------------------------------+

5 rows
ready to start consuming query after 2 ms, results consumed after another 1 ms
neo4j@neo4j# :commit
```

Example result in the query.log file

```
2021-07-30 14:43:17.176+0000 INFO  id:225 - 2 ms: 136 B - bolt-session  bolt    neo4j-cypher-
shell/v4.4.0       client/127.0.0.1:54026  server/127.0.0.1:7687>  neo4j - neo4j -
MATCH (n:Person) RETURN n  LIMIT 5; - {} - runtime=pipelined - {app: 'neo4j-cypher-shell_v.4.4.0',
type: 'user-direct', user: 'jsmith'}
```

Example 91. Using Neo4j Browser, attach metadata to a transaction

```
CALL tx.setMetaData({app: 'neo4j-browser_v.4.4.0', type: 'user-direct', user: 'jsmith'});
MATCH (n:Person) RETURN n LIMIT 5
```

Example result in the query.log file

```
2021-07-30 14:51:39.457+0000 INFO  Query started: id:328 - 0 ms: 0 B - bolt-session bolt    neo4j-
browser/v4.4.0        client/127.0.0.1:53666  server/127.0.0.1:7687>  neo4j - neo4j - MATCH
(n:Person) RETURN n  LIMIT 5 - {} - runtime=null - {type: 'system', app: 'neo4j-browser_v4.4.0'}
```

Example 92. 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.

JSON format

The query log can use a JSON layout. In order to change the format, the layout for the `QueryLogger` must be changed from using the `PatternLayout`:

```
<RollingRandomAccessFile name="QueryLog" fileName="${config:server.directories.logs}/query.log"
                         filePattern="${config:server.directories.logs}/query.log.%02i"
                 createOnDemand="true">
    <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSSZ}{GMT+0} %-5p %m%n"/>
    <Policies>
        <SizeBasedTriggeringPolicy size="20 MB"/>
    </Policies>
    <DefaultRolloverStrategy fileIndex="min" max="7"/>
</RollingRandomAccessFile>
```

to using the `JsonTemplateLayout`:

```
<RollingRandomAccessFile name="QueryLog" fileName="${config:server.directories.logs}/query.log"
                         filePattern="${config:server.directories.logs}/query.log.%02i"
                 createOnDemand="true">
    <JsonTemplateLayout eventTemplateUri="classpath:org/neo4j/logging/QueryLogJsonLayout.json"/>
    <Policies>
        <SizeBasedTriggeringPolicy size="20 MB"/>
    </Policies>
    <DefaultRolloverStrategy fileIndex="min" max="7"/>
</RollingRandomAccessFile>
```

The `QueryLogJsonLayout.json` template mimics the 4.x layout and contains the following information:

Table 56. JSON format log entries

Name	Description
`time`	The timestamp of the log message.
`level`	The log level.
`type`	Valid options are `query` and `transaction`.
`stacktrace`	Included when there is a stacktrace associated with the log message.

If the type of the log entry is `query`, these additional fields are available:

Table 57. JSON format log entries

Name	Description
`event`	Valid options are `start`, `fail`, and `success`.

Name	Description
`id`	The query ID. Included when `db.logs.query.enabled` is `VERBOSE`.
`elapsedTimeMs`	The elapsed time in milliseconds.
`planning`	Milliseconds spent on planning.
`cpu`	Milliseconds spent actively executing on the CPU.
`waiting`	Milliseconds spent waiting on locks or other queries, as opposed to actively running this query.
`allocatedBytes`	Number of bytes allocated by the query.
`pageHits`	Number of page hits.
`pageFaults`	Number of page faults.
`source`	Connection details.
`database`	The database name on which the query is run.
`executingUser`	The name of the user executing the query. Either same as `authenticatedUser` or an impersonated user.
`authenticatedUser`	The name of the user who authenticated and is executing the query.
`query`	The query text.
`queryParameters`	The query parameters. Included when `db.logs.query.parameter_logging_enabled` is `true`.
`runtime`	The runtime used to run the query.
`annotationData`	Metadata attached to the transaction.
`failureReason`	Reason for failure. Included when applicable.
`transactionId`	The transaction ID of the running query.

Name	Description
`queryPlan`	The query plan. Included when `db.logs.query.plan_description_enabled` is `true`.

If the type of the log entry is `transaction`, the following additional fields are available:

Table 58. JSON format log entries

Name	Description
`event`	Valid options are `start`, `rollback`, and `commit`.
`database`	The database name on which the transaction is run.
`executingUser`	The name of the user connected to the transaction. Either same as `authenticatedUser` or an impersonated user.
`authenticatedUser`	The name of the user who authenticated and is connected to the transaction.
`transactionId`	ID of the transaction.

Metrics

You can configure Neo4j to log many different metrics to help you keep your applications running smoothly. By default, you can view this data via Neo4j Ops Manager (NOM), retrieved from CSV files, or exposed over JMX MBeans, but you can also export it to third-party monitoring tools, such as Graphite and Prometheus.

Reading the Performance section is recommended to better understand the metrics.

This section describes the following:

- Essential metrics — Some of the important metrics most Neo4j administrators need to monitor.
- Enable metrics logging — How to configure Neo4j to log metrics data.
- Expose metrics — How to view data using NOM, JMX, CSV files, or other third-party monitoring tools.
- Metrics reference — Neo4j available metrics.

Essential metrics

To ensure your applications are running smoothly, you should monitor:

- The **server load** — the strain on the machine hosting Neo4j.
- The **Neo4j load** — the strain on Neo4j.

- The **cluster health** — to ensure the cluster is working as expected.
- The **workload** of a Neo4j instance.

Reading the Performance section is recommended to better understand the metrics.

Server load metrics

Monitoring the hardware resources shows the strain on the server running Neo4j.

You can use utilities, such as the collectd daemon or `systemd` on Linux, to gather information about the system. These metrics can help with capacity planning as your workload grows.

Metric name	Description
CPU usage	If this is reaching 100%, you may need additional CPU capacity.
Used memory	This metric tells you if you are close to using all available memory on the server. Make sure your peaks are at 95% or below to reduce the risk of running out of memory. For more information, see Memory configuration.
Free disk space	Observe the rate of your data growth so you can plan for additional storage before you run out. This applies to all disks that Neo4j is writing to. You might also choose to write the log files to a different disk.

Neo4j load metrics

The Neo4j load metrics monitor the strain that Neo4j is being put under. They can help with capacity planning.

Metric name	Metric	Description
Heap usage	`<prefix>.dbms.vm.heap.used` and `<prefix>.dbms.vm.heap.total`	If Neo4j consistently uses 100% of the heap, increase the initial and max heap size. For more information, see Memory configuration.
Page cache	`<prefix>.dbms.page_cache.hit_ratio` and `<prefix>.dbms.page_cache.usage_ratio`	When a request misses the page cache, the data must be fetched from a much slower disk. Ideally, the hit_ratio should be above 98% most of the time. This shows how much of the allocated memory to the page cache is used. If this is at 100%, consider increasing the page cache size.

Metric name	Metric	Description
JVM garbage collection	`<prefix>.dbms.vm.gc.time.%s`	The proportion of time the JVM spends reclaiming the heap instead of doing other work. This metric can spike when the database is running low on memory. If this happens, it can halt processing and cause query execution errors. Consider increasing the size of your database if this appears to be the case.
Checkpoint time	`<prefix>.database.<db>.check_point.duration`	You should monitor the checkpoint duration to ensure it does not start to approach the interval between checkpoints. If this happens, consider the following steps to improve checkpointing performance: • Raise the `db.checkpoint.iops.limit` to make checkpoints faster, but only if there is enough IOPS budget available to avoid slowing the commit process. • `server.memory.pagecache.flush.buffer.enabled` / `server.memory.pagecache.flush.buffer.size_in_pages` make checkpoints faster by writing batches of data in a way that plays well with the underlying disk (with a nice multiple to the block size). • Change the checkpointing policy (`db.checkpoint.*`, `db.checkpoint.interval.time`) to more frequent/smaller checkpoints, continuous checkpointing, or checkpoint by volume or `tx` count. For more information, see Checkpoint IOPS limit and Log pruning.

Neo4j cluster health metrics

The cluster health metrics indicate the health of a cluster member at a glance. It is essential to know which instance is the leader. The leader's load pattern differs from the followers, which should exhibit similar load patterns.

Metric name	Metric	Description
Leader	`<prefix>.database.<db>.cluster.raft.is_leader`	Track this for each database primary. It reports `0` if it is not the leader and `1` if it is the leader. The sum of all of these should always be `1`. However, there are transient periods in which the sum can be more than `1` because more than one member thinks it is the leader.
Transaction workload	`<prefix>.database.<db>.transaction.last_committed_tx_id`	The ID of the last committed transaction. Track this for each Neo4j instance. It might break into separate charts. It should show one line, ever-increasing, and if one of the lines levels off or falls behind, it is clear that this instance is no longer replicating data, and action is needed to rectify the situation.

Workload metrics

These metrics help monitor the workload of a Neo4j instance. The absolute values of these depend on the sort of workload you expect.

Metric name	Metric	Description
Bolt connections	`<prefix>.dbms.bolt.connections_running`	The number of connections that are currently executing Cypher and returning results.
Total nodes/relationships	`<prefix>.database.<db>.count.node` and `<prefix>.database.<db>.count.relationship`	(Not enabled by default) Total number of distinct relationship types. Total number of distinct property names. Total number of relationships. Total number of nodes.
Throughput	`<prefix>.database.<db>.db.query.execution.latency.millis`	This metric produces a histogram of 99th and 95th percentile transaction latencies. Useful for identifying spikes or increases in the data load.

For the complete list of all available metrics in Neo4j, see Metrics reference.

Enable metrics logging

A subset of all available metrics is enabled by default. See server.metrics.filter. The list was last updated in version 4.2.

You can enable/disable metrics using the configuration setting server.metrics.enabled. You can also use the setting server.metrics.filter to enable only the metrics you want. The metrics must be specified as a comma-separated list of globbing patterns. The following example enables all checkpoint metrics and the pagecache eviction metric:

```
# Setting for enabling all supported metrics. (Default is true) Setting this to false disables all
metrics.
server.metrics.enabled=true

# Setting for enabling only the metrics matching the filter.
server.metrics.filter=*check_point*,neo4j.page_cache.evictions
```

For more information on the available metrics configuration settings, see Configuration settings.

Expose metrics

Neo4j supports the following ways of exposing data for monitoring purposes:

- Neo4j Ops Manager — a UI-based tool that enables a DBA (or any administrator) to monitor, administer, and operate all of the Neo4j DBMSs in an Enterprise.
- CSV files — retrieve metrics from CSV files (true by default).
- JMX MBeans — expose metrics over JMX MBeans (true by default).
- Graphite — send metrics to Graphite or any monitoring tool based on the Graphite protocol (false by default).
- Prometheus — publish metrics for polling as Prometheus endpoint (false by default).

Neo4j Ops Manager

The Neo4j metrics data can be viewed via Neo4j Ops Manager (NOM). For more information on how to install and set up NOM to be used with your Neo4j DBMS, see Neo4j Ops Manager documentation.

CSV files

Export metrics to CSV files.

Add the following settings to *neo4j.conf* in order to enable export of metrics into local .CSV files:

```
# Enable the CSV exporter. Default is true.
server.metrics.csv.enabled=true
# Directory path for output files.
# Default is the /metrics directory under NEO4J_HOME.
server.directories.metrics=/local/file/system/path
# How often to store data. Default is 30 seconds.
server.metrics.csv.interval=30s
# The maximum number of CSV files that will be saved. Default is 7.
server.metrics.csv.rotation.keep_number=7
# The file size at which the CSV files will auto-rotate. Default is 10.00MiB.
server.metrics.csv.rotation.size=10.00MiB
# Compresses the metric archive files. Default is NONE. Possible values are NONE, ZIP, and GZ.
server.metrics.csv.rotation.compression=ZIP
```

`server.metrics.csv.rotation.compression` selects the compression scheme to use on the files after rotation. Since CSV files are highly compressible, it is recommended to enable compression of the files to save disk space.

JMX MBeans

From version 4.2.2 onwards, the JMX metrics are exposed by default over JMX MBeans.

```
# Enable the JMX MBeans integration. Default is true.
server.metrics.jmx.enabled=true
```

For more information about accessing and adjusting the metrics, see The Java Reference Guide → JMX metrics.

Graphite

Send metrics to Graphite or any monitoring tool based on the Graphite protocol.

Add the following settings to neo4j.conf to enable integration with Graphite:

```
# Enable the Graphite integration. Default is false.
server.metrics.graphite.enabled=true
# The hostname or IP address of the Graphite server.
# A socket address in the format <hostname>, <hostname>:<port>, or :<port>.
# If missing, the port or hostname is acquired from server.default_listen_address.
# The default port number for Graphite is 2003.
server.metrics.graphite.server=localhost:2003
# How often to send data. Default is 30 seconds.
server.metrics.graphite.interval=30s
# Prefix for Neo4j metrics on Graphite server.
server.metrics.prefix=neo4j
```

Start Neo4j and connect to Graphite via a web browser to monitor your Neo4j metrics.

If you configure the Graphite server to be a hostname or DNS entry, you should be aware that the JVM resolves hostnames to IP addresses and, by default, caches the result indefinitely for security reasons. This is controlled by the value of `networkaddress.cache.ttl` in the JVM Security properties. See https://docs.oracle.com/javase/8/docs/technotes/guides/net/properties.html for more information.

Prometheus

Publish metrics for polling as Prometheus endpoint.

Add the following settings to *neo4j.conf* to enable the Prometheus endpoint.

```
# Enable the Prometheus endpoint. Default is false.
server.metrics.prometheus.enabled=true
# The hostname and port to use as Prometheus endpoint.
# A socket address is in the format <hostname>, <hostname>:<port>, or :<port>.
# If missing, the port or hostname is acquired from server.default_listen_address.
# The default is localhost:2004.
server.metrics.prometheus.endpoint=localhost:2004
```

When Neo4j is fully started, a Prometheus endpoint will be available at the configured address.

Metrics reference

You should use caution when interpreting unfamiliar metrics. Reading the Performance section is recommended to better understand the metrics.

Types of metrics

Neo4j has the following types of metrics:

- Global — covers the whole Neo4j DBMS.
- Per database — covers an individual database.

The metrics fall into one of the following categories:

- Gauge — shows an instantaneous reading of a particular value.
- Counter — shows an accumulated value.
- Histogram — shows the distribution of values.

Global metrics

Global metrics cover the whole database management system and represent the system's status as a whole.

Global metrics have the following name format:

- `<metrics-prefix>.dbms.<metric-name>`, where the <user-configured-prefix> can be configured with the `server.metrics.prefix` configuration setting.

Metrics of this type are reported as soon as the database management system is available. For example, all JVM-related metrics are global. In particular, the `neo4j.dbms.vm.thread.count` metric has a default user-configured-prefix `neo4j` and the global metric name is `vm.thread.count`.

By default, global metrics include:

* Bolt metrics
* Page cache metrics
* GC metrics
* Thread metrics
* Database operation metrics
* Web Server metrics
* JVM metrics

Database metrics

Each database metric is reported for a particular database only. Database metrics are only available during the lifetime of the database. When a database becomes unavailable, all of its metrics become unavailable also.

Database metrics have the following name format:

* `<user-configured-prefix>.database.<database-name>.<metric-name>`, where the <user-configured-prefix> can be configured with the `server.metrics.prefix` configuration setting.

For example, any transaction metric is a database metric. In particular, the `neo4j.database.mydb.transaction.started` metric has a default user-configured-prefix `neo4j` and is a metric for the `mydb` database.

By default, database metrics include:

* Transaction metrics
* Checkpoint metrics
* Log rotation metrics
* Database data metrics
* Cypher metrics
* Clustering metrics

General-purpose metrics

Table 59. Bolt metrics

Name	Description
`<prefix>.bolt.connections_opened`	The total number of Bolt connections opened since startup. This includes both succeeded and failed connections. Useful for monitoring load via the Bolt drivers in combination with other metrics. (counter)
`<prefix>.bolt.connections_closed`	The total number of Bolt connections closed since startup. This includes both properly and abnormally ended connections. Useful for monitoring load via Bolt drivers in combination with other metrics. (counter)

Name	Description
`<prefix>.bolt.connections_running`	The total number of Bolt connections that are currently executing Cypher and returning results. Useful to track the overall load on Bolt connections. This is limited to the number of Bolt worker threads that have been configured via `dbms.connector.bolt.thread_pool_max_size`. Reaching this maximum indicated the server is running at capacity. (gauge)
`<prefix>.bolt.connections_idle`	The total number of Bolt connections that are not currently executing Cypher or returning results. (gauge)
`<prefix>.bolt.messages_received`	The total number of messages received via Bolt since startup. Useful to track general message activity in combination with other metrics. (counter)
`<prefix>.bolt.messages_started`	The total number of messages that have started processing since being received. A received message may have begun processing until a Bolt worker thread becomes available. A large gap observed between `bolt.messages_received` and `bolt.messages_started` could indicate the server is running at capacity. (counter)
`<prefix>.bolt.messages_done`	The total number of Bolt messages that have completed processing whether successfully or unsuccessfully. Useful for tracking overall load. (counter)
`<prefix>.bolt.messages_failed`	The total number of messages that have failed while processing. A high number of failures may indicate an issue with the server and further investigation of the logs is recommended. (counter)
`<prefix>.bolt.accumulated_queue_time`	(unsupported feature) When `internal.server.bolt.thread_pool_queue_size` is enabled, the total time in milliseconds that a Bolt message waits in the processing queue before a Bolt worker thread becomes available to process it. Sharp increases in this value indicate that the server is running at capacity. If `internal.server.bolt.thread_pool_queue_size` is disabled, the value should be `0`, meaning that messages are directly handed off to worker threads. (counter)
`<prefix>.bolt.accumulated_processing_time`	The total amount of time in milliseconds that worker threads have been processing messages. Useful for monitoring load via Bolt drivers in combination with other metrics. (counter)

Table 60. Database checkpointing metrics

Name	Description
`<prefix>.check_point.events`	The total number of checkpoint events executed so far. (counter)
`<prefix>.check_point.total_time`	The total time, in milliseconds, spent in checkpointing so far. (counter)
`<prefix>.check_point.duration`	The duration, in milliseconds, of the last checkpoint event. Checkpoints should generally take several seconds to several minutes. Long checkpoints can be an issue, as these are invoked when the database stops, when a hot backup is taken, and periodically as well. Values over `30` minutes or so should be cause for some investigation. (gauge)

Name	Description
`<prefix>.check_point.pages_flushed`	The number of pages that were flushed during the last checkpoint event. (gauge)
`<prefix>.check_point.io_performed`	The number of IOs from Neo4j perspective performed during the last check point event. (gauge)
`<prefix>.check_point.io_limit`	The IO limit used during the last checkpoint event. (gauge)

Table 61. Cypher metrics

Name	Description
`<prefix>.cypher.replan_events`	The total number of times Cypher has decided to re-plan a query. Neo4j caches 1000 plans by default. Seeing sustained replanning events or large spikes could indicate an issue that needs to be investigated. (counter)
`<prefix>.cypher.replan_wait_time`	The total number of seconds waited between query replans. (counter)

Table 62. Database data count metrics

Name	Description
`<prefix>.neo4j.count.relationship`	The total number of relationships in the database. (gauge)
`<prefix>.neo4j.count.node`	The total number of nodes in the database. A rough metric of how big your graph is. And if you are running a bulk insert operation you can see this tick up. (gauge)

Table 63. Database neo4j pools metrics

Name	Description
`<prefix>.pool.<pool>.<database>.used_heap`	Used or reserved heap memory in bytes. (gauge)
`<prefix>.pool.<pool>.<database>.used_native`	Used or reserved native memory in bytes. (gauge)
`<prefix>.pool.<pool>.<database>.total_used`	Sum total used heap and native memory in bytes. (gauge)
`<prefix>.pool.<pool>.<database>.total_size`	Sum total size of capacity of the heap and/or native memory pool. (gauge)
`<prefix>.pool.<pool>.<database>.free`	Available unused memory in the pool, in bytes. (gauge)

Table 64. 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)

Name	Description
`<prefix>.db.operation.count.recovered`	Count of database operations that failed previously but have recovered. (counter)

Table 65. Database data metrics

Name	Description
`<prefix>.ids_in_use.relationship_type`	The total number of different relationship types stored in the database. Informational, not an indication of any issue. Spikes or large increases indicate large data loads, which could correspond with some behavior you are investigating. (gauge)
`<prefix>.ids_in_use.property`	The total number of different property names used in the database. Informational, not an indication of any issue. Spikes or large increases indicate large data loads, which could correspond with some behavior you are investigating. (gauge)
`<prefix>.ids_in_use.relationship`	The total number of relationships stored in the database. Informational, not an indication of any issue. Spikes or large increases indicate large data loads, which could correspond with some behavior you are investigating. (gauge)
`<prefix>.ids_in_use.node`	The total number of nodes stored in the database. Informational, not an indication of any issue. Spikes or large increases indicate large data loads, which could correspond with some behavior you are investigating. (gauge)

Table 66. Global neo4j pools metrics

Name	Description
`<prefix>.dbms.pool.<pool>.used_heap`	Used or reserved heap memory in bytes. (gauge)
`<prefix>.dbms.pool.<pool>.used_native`	Used or reserved native memory in bytes. (gauge)
`<prefix>.dbms.pool.<pool>.total_used`	Sum total used heap and native memory in bytes. (gauge)
`<prefix>.dbms.pool.<pool>.total_size`	Sum total size of the capacity of the heap and/or native memory pool. (gauge)
`<prefix>.dbms.pool.<pool>.free`	Available unused memory in the pool, in bytes. (gauge)

Table 67. 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)

Name	Description
`<prefix>.page_cache.pins`	The total number of page pins executed by the page cache. (counter)
`<prefix>.page_cache.evictions`	The total number of page evictions executed by the page cache. (counter)
`<prefix>.page_cache.evictions.cooperative`	The total number of cooperative page evictions executed by the page cache due to low available pages. (counter)
`<prefix>.page_cache.page_faults`	The total number of page faults happened in the page cache. If this continues to rise over time, it could be an indication that more page cache is needed. (counter)
`<prefix>.page_cache.page_fault_failures`	The total number of failed page faults happened in the page cache. (counter)
`<prefix>.page_cache.page_cancelled_faults`	The total number of canceled page faults happened in the page cache. (counter)
`<prefix>.page_cache.hits`	The total number of page hits happened in the page cache. (counter)
`<prefix>.page_cache.hit_ratio`	The ratio of hits to the total number of lookups in the page cache. Performance relies on efficiently using the page cache, so this metric should be in the 98-100% range consistently. If it is much lower than that, then the database is going to disk too often. (gauge)
`<prefix>.page_cache.usage_ratio`	The ratio of number of used pages to total number of available pages. This metric shows what percentage of the allocated page cache is actually being used. If it is 100%, then it is likely that the hit ratio will start dropping, and you should consider allocating more RAM to page cache. (gauge)
`<prefix>.page_cache.bytes_read`	The total number of bytes read by the page cache. (counter)
`<prefix>.page_cache.bytes_written`	The total number of bytes written by the page cache. (counter)
`<prefix>.page_cache.iops`	The total number of IO operations performed by page cache.
`<prefix>.page_cache.throttled.times`	The total number of times page cache flush IO limiter was throttled during ongoing IO operations.
`<prefix>.page_cache.throttled.millis`	The total number of millis page cache flush IO limiter was throttled during ongoing IO operations.
`<prefix>.page_cache.pages_copied`	The total number of page copies happened in the page cache. (counter)

Table 68. Query execution metrics

Name	Description
`<prefix>.db.query.execution.success`	Count of successful queries executed. (counter)
`<prefix>.db.query.execution.failure`	Count of failed queries executed. (counter)
`<prefix>.db.query.execution.latency.millis`	Execution time in milliseconds of queries executed successfully. (histogram)

Table 69. Database store size metrics

Name	Description
`<prefix>.store.size.total`	The total size of the database and transaction logs, in bytes. The total size of the database helps determine how much cache page is required. It also helps compare the total disk space used by the data store and how much is available. (gauge)
`<prefix>.store.size.database`	The size of the database, in bytes. The total size of the database helps determine how much cache page is required. It also helps compare the total disk space used by the data store and how much is available. (gauge)

Table 70. Database transaction log metrics

Name	Description
`<prefix>.log.rotation_events`	The total number of transaction log rotations executed so far. (counter)
`<prefix>.log.rotation_total_time`	The total time, in milliseconds, spent in rotating transaction logs so far. (counter)
`<prefix>.log.rotation_duration`	The duration, in milliseconds, of the last log rotation event. (gauge)
`<prefix>.log.appended_bytes`	The total number of bytes appended to the transaction log. (counter)
`<prefix>.log.flushes`	The total number of transaction log flushes. (counter)
`<prefix>.log.append_batch_size`	The size of the last transaction append batch. (gauge)

Table 71. Database transaction metrics

Name	Description
`<prefix>.transaction.started`	The total number of started transactions. (counter)
`<prefix>.transaction.peak_concurrent`	The highest peak of concurrent transactions. This is a useful value to understand. It can help you with the design for the highest load scenarios and whether the Bolt thread settings should be altered. (counter)
`<prefix>.transaction.active`	The number of currently active transactions. Informational, not an indication of any issue. Spikes or large increases could indicate large data loads or just high read load. (gauge)
`<prefix>.transaction.active_read`	The number of currently active read transactions. (gauge)
`<prefix>.transaction.active_write`	The number of currently active write transactions. (gauge)
`<prefix>.transaction.committed`	The total number of committed transactions. Informational, not an indication of any issue. Spikes or large increases indicate large data loads or just high read load. (counter)
`<prefix>.transaction.committed_read`	The total number of committed read transactions. Informational, not an indication of any issue. Spikes or large increases indicate high read load. (counter)

Name	Description
`<prefix>.transaction.committed_write`	The total number of committed write transactions. Informational, not an indication of any issue. Spikes or large increases indicate large data loads, which could correspond with some behavior you are investigating. (counter)
`<prefix>.transaction.rollbacks`	The total number of rolled back transactions. (counter)
`<prefix>.transaction.rollbacks_read`	The total number of rolled back read transactions. (counter)
`<prefix>.transaction.rollbacks_write`	The total number of rolled back write transactions. Seeing a lot of writes rolled back may indicate various issues with locking, transaction timeouts, etc. (counter)
`<prefix>.transaction.terminated`	The total number of terminated transactions. (counter)
`<prefix>.transaction.terminated_read`	The total number of terminated read transactions. (counter)
`<prefix>.transaction.terminated_write`	The total number of terminated write transactions. (counter)
`<prefix>.transaction.last_committed_tx_id`	The ID of the last committed transaction. Track this for each instance. (Cluster) Track this for each primary, and each secondary. Might break into separate charts. It should show one line, ever increasing, and if one of the lines levels off or falls behind, it is clear that this member is no longer replicating data, and action is needed to rectify the situation. (counter)
`<prefix>.transaction.last_closed_tx_id`	The ID of the last closed transaction. (counter)
`<prefix>.transaction.tx_size_heap`	The transactions' size on heap in bytes. (histogram)
`<prefix>.transaction.tx_size_native`	The transactions' size in native memory in bytes. (histogram)

Table 72. Server metrics

Name	Description
`<prefix>.server.threads.jetty.idle`	The total number of idle threads in the jetty pool. (gauge)
`<prefix>.server.threads.jetty.all`	The total number of threads (both idle and busy) in the jetty pool. (gauge)

Metrics specific to clustering

Table 73. CatchUp Metrics

Name	Description
`<prefix>.cluster.catchup.tx_pull_requests_received`	TX pull requests received from secondaries. (counter)

Table 74. Discovery database primary metrics

Name	Description
`<prefix>.cluster.discovery.replicated_data`	Size of replicated data structures. (gauge)
`<prefix>.cluster.discovery.cluster.members`	Discovery cluster member size. (gauge)
`<prefix>.cluster.discovery.cluster.unreachable`	Discovery cluster unreachable size. (gauge)

Name	Description
`<prefix>.cluster.discovery.cluster.converged`	Discovery cluster convergence. (gauge)

Table 75. Raft database primary metrics

Name	Description
`<prefix>.cluster.raft.append_index`	The append index of the Raft log. Each index represents a write transaction (possibly internal) proposed for commitment. The values mostly increase, but sometimes they can decrease as a consequence of leader changes. The append index should always be less than or equal to the commit index. (gauge)
`<prefix>.cluster.raft.commit_index`	The commit index of the Raft log. Represents the commitment of previously appended entries. Its value increases monotonically if you do not unbind the cluster state. The commit index should always be bigger than or equal to the appended index. (gauge)
`<prefix>.cluster.raft.applied_index`	The applied index of the Raft log. Represents the application of the committed Raft log entries to the database and internal state. The applied index should always be bigger than or equal to the commit index. The difference between this and the commit index can be used to monitor how up-to-date the follower database is. (gauge)
`<prefix>.cluster.raft.term`	The Raft Term of this server. It increases monotonically if you do not unbind the cluster state. (gauge)
`<prefix>.cluster.raft.tx_retries`	Transaction retries. (counter)
`<prefix>.cluster.raft.is_leader`	Is this server the leader? Track this for each database primary in the cluster. It reports `0` if it is not the leader and `1` if it is the leader. The sum of all of these should always be `1`. However, there are transient periods in which the sum can be more than `1` because more than one member thinks it is the leader. Action may be needed if the metric shows `0` for more than 30 seconds. (gauge)
`<prefix>.cluster.raft.in_flight_cache.total_bytes`	In-flight cache total bytes. (gauge)
`<prefix>.cluster.raft.in_flight_cache.max_bytes`	In-flight cache max bytes. (gauge)
`<prefix>.cluster.raft.in_flight_cache.element_count`	In-flight cache element count. (gauge)
`<prefix>.cluster.raft.in_flight_cache.max_elements`	In-flight cache maximum elements. (gauge)
`<prefix>.cluster.raft.in_flight_cache.hits`	In-flight cache hits. (counter)
`<prefix>.cluster.raft.in_flight_cache.misses`	In-flight cache misses. (counter)
`<prefix>.cluster.raft.raft_log_entry_prefetch_buffer.lag`	Raft Log Entry Prefetch Lag. (gauge)
`<prefix>.cluster.raft.raft_log_entry_prefetch_buffer.bytes`	Raft Log Entry Prefetch total bytes. (gauge)
`<prefix>.cluster.raft.raft_log_entry_prefetch_buffer.size`	Raft Log Entry Prefetch buffer size. (gauge)
`<prefix>.cluster.raft.raft_log_entry_prefetch_buffer.async_put`	Raft Log Entry Prefetch buffer async puts. (gauge)

Name	Description
`<prefix>.cluster.raft.raft_log_entry_prefetch_buffer.sync_put`	Raft Log Entry Prefetch buffer sync puts. (gauge)
`<prefix>.cluster.raft.message_processing_delay`	Delay between Raft message receive and process. (gauge)
`<prefix>.cluster.raft.message_processing_timer`	Timer for Raft message processing. (counter, histogram)
`<prefix>.cluster.raft.replication_new`	The total number of Raft replication requests. It increases with write transactions (possibly internal) activity. (counter)
`<prefix>.cluster.raft.replication_attempt`	The total number of Raft replication requests attempts. It is bigger or equal to the replication requests. (counter)
`<prefix>.cluster.raft.replication_fail`	The total number of Raft replication attempts that have failed. (counter)
`<prefix>.cluster.raft.replication_maybe`	Raft Replication maybe count. (counter)
`<prefix>.cluster.raft.replication_success`	The total number of Raft replication requests that have succeeded. (counter)
`<prefix>.cluster.raft.last_leader_message`	The time elapsed since the last message from a leader in milliseconds. Should reset periodically. (gauge)

Table 76. Database secondary Metrics

Name	Description
`<prefix>.cluster.store_copy.pull_updates`	The total number of pull requests made by this instance. (counter)
`<prefix>.cluster.store_copy.pull_update_highest_tx_id_requested`	The highest transaction id requested in a pull update by this instance. (counter)
`<prefix>.cluster.store_copy.pull_update_highest_tx_id_received`	The highest transaction id that has been pulled in the last pull updates by this instance. (counter)

Java Virtual Machine Metrics

The JVM metrics show information about garbage collections (for example, the number of events and time spent collecting), memory pools and buffers, and the number of active threads running. They are environment dependent and therefore, may vary on different hardware and with different JVM configurations. The metrics about the JVM's memory usage expose values that are provided by the MemoryPoolMXBeans and BufferPoolMXBeans. The memory pools are memory managed by the JVM, for example, `neo4j.dbms.vm.memory.pool.g1_survivor_space`. Therefore, if necessary, you can tune them using the JVM settings. The buffer pools are space outside of the memory managed by the garbage collector. Neo4j allocates buffers in those pools as it needs them. You can limit this memory using JVM settings, but there is never any good reason for you to set them.

Table 77. JVM file descriptor metrics.

Name	Description
`<prefix>.vm.file.descriptors.count`	The current number of open file descriptors. (gauge)

Name	Description
`<prefix>.vm.file.descriptors.maximum`	(OS setting) The maximum number of open file descriptors. It is recommended to be set to 40K file handles, because of the native and Lucene indexing Neo4j uses. If this metric gets close to the limit, you should consider raising it. (gauge)

Table 78. GC metrics.

Name	Description
`<prefix>.vm.gc.time.<gc>`	Accumulated garbage collection time in milliseconds. Long GCs can be an indication of performance issues or potential instability. If this approaches the heartbeat timeout in a cluster, it may cause unwanted leader switches. (counter)
`<prefix>.vm.gc.count.<gc>`	Total number of garbage collections. (counter)

Table 79. JVM Heap metrics.

Name	Description
`<prefix>.vm.heap.committed`	Amount of memory (in bytes) guaranteed to be available for use by the JVM. (gauge)
`<prefix>.vm.heap.used`	Amount of memory (in bytes) currently used. This is the amount of heap space currently used at a given point in time. Monitor this to identify if you are maxing out consistently, in which case, you should increase the initial and max heap size, or if you are underutilizing, you should decrease the initial and max heap sizes. (gauge)
`<prefix>.vm.heap.max`	Maximum amount of heap memory (in bytes) that can be used. This is the amount of heap space currently used at a given point in time. Monitor this to identify if you are maxing out consistently, in which case, you should increase the initial and max heap size, or if you are underutilizing, you should decrease the initial and max heap sizes. (gauge)

Table 80. JVM memory buffers metrics.

Name	Description
`<prefix>.vm.memory.buffer.<bufferpool>.count`	Estimated number of buffers in the pool. (gauge)
`<prefix>.vm.memory.buffer.<bufferpool>.used`	Estimated amount of memory used by the pool. (gauge)
`<prefix>.vm.memory.buffer.<bufferpool>.capacity`	Estimated total capacity of buffers in the pool. (gauge)

Table 81. JVM memory pools metrics.

Name	Description
`<prefix>.vm.memory.pool.<pool>`	Estimated amount of memory in bytes used by the pool. (gauge)

Table 82. JVM pause time metrics.

Name	Description
`<prefix>.vm.pause_time`	Accumulated detected VM pause time. (counter)

Table 83. JVM threads metrics.

Name	Description
`<prefix>.vm.threads`	The total number of live threads including daemon and non-daemon threads. (gauge)

Manage queries

List all running queries

The procedure for listing queries, `dbms.listQueries()`, is replaced by the command for listing transactions, `SHOW TRANSACTIONS`. This command returns information about the currently executing query in the transaction. For more information on the command, see the Cypher manual → `SHOW TRANSACTIONS` command.

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 93. Viewing active locks for a query

The following example demonstrates how to show the active locks held by the transaction executing a given query.

Firstly, to get the IDs of the currently executing queries, yield the `currentQueryId` from the `SHOW TRANSACTIONS` command:

```
SHOW TRANSACTIONS YIELD currentQueryId, currentQuery
```

Then, call `dbms.listActiveLocks` passing the `currentQueryId` of interest (`query-614` in this example):

```
CALL dbms.listActiveLocks( "query-614" )
```

```
╒════════╤══════════════╤════════════╕
│"mode"  │"resourceType"│"resourceId"│
╞════════╪══════════════╪════════════╡
│"SHARED"│"SCHEMA"      │0           │
└────────┴──────────────┴────────────┘
1 row
```

Terminate queries

Queries are terminated by terminating the transaction on which they are running. This is done using the `TERMINATE TRANSACTIONS transactionIds` command. The `transactionIds` can be found using the `SHOW TRANSACTIONS` command.

The `TERMINATE TRANSACTION` privilege determines what transactions can be terminated. However, the current user can always terminate all of their own transactions.

Syntax:

`TERMINATE TRANSACTIONS transactionIds`

Argument:

Name	Type	Description
`transactionIds`	Comma-separated strings	The IDs of all the transactions to be terminated.
`transactionIds`	Single string parameter	The ID of the transaction to be terminated.
`transactionIds`	List parameter	The IDs of all the transactions to be terminated.

For more information on the command, see the Cypher manual → `TERMINATE TRANSACTIONS` command.

Manage transactions

Configure transaction timeout

It is recommended to configure Neo4j to terminate transactions whose execution time has exceeded the configured timeout.

- Set `db.transaction.timeout` to some positive time interval value (e.g.,`10s`) denoting the default transaction timeout. Setting `db.transaction.timeout` to `0` — which is the default value — disables the feature.
- You can also set this dynamically on each instance (Read Replicas only if required) using the procedure `dbms.setConfigValue('db.transaction.timeout','10s')`.

Example 94. Configure transaction timeout

Set the timeout to ten seconds.

```
db.transaction.timeout=10s
```

Configuring transaction timeout has no effect on transactions executed with custom timeouts (e.g., via the Java API or Neo4j Drivers), as the custom timeout overrides the value set for `db.transaction.timeout`. Note that the timeout value can only be overridden to a value that is smaller than that configured by `db.transaction.timeout`.

The *transaction timeout* feature is also known as the *transaction guard*.

Configure lock acquisition timeout

An executing transaction may get stuck while waiting for some lock to be released by another transaction. To kill that transaction and remove the lock, set set `db.lock.acquisition.timeout` to some positive time interval value (e.g., `10s`) denoting the maximum time interval within which any particular lock should be acquired, before failing the transaction. Setting `db.lock.acquisition.timeout` to `0` — which is the default value — disables the lock acquisition timeout.

This feature cannot be set dynamically.

Example 95. Configure lock acquisition timeout

Set the timeout to ten seconds.

```
db.lock.acquisition.timeout=10s
```

List all running transactions

To list the currently running transactions within an instance, use the `SHOW TRANSACTIONS` command.

The `SHOW TRANSACTION` privilege determines what transactions are returned by the command. However, the current user can always view all of their own currently executing transactions.

Syntax:

```
SHOW TRANSACTIONS
```

For more information on this command, see the Cypher manual → `SHOW TRANSACTIONS` command.

Manage connections

List all network connections

An administrator is able to view all network connections within the database instance. Alternatively, the current user may view all of their own network connections.

The procedure `dbms.listConnections` lists all accepted network connections for all configured connectors, including Bolt, HTTP, and HTTPS. Some listed connections might never perform authentication. For example, HTTP GET requests to the Neo4j Browser endpoint fetches static resources and does not need to authenticate. However, connections made using Neo4j Browser require the user to provide credentials and perform authentication. For more information on Neo4j Browser connections, see the Neo4j Browser documentation.

Syntax:

```
CALL dbms.listConnections()
```

Table 84. Data retrieved from a database

Name	Type	Description
`connectionId`	String	This is the ID of the network connection.
`connectTime`	String	This is the time at which the connection was started.
`connector`	String	Name of the connector that accepted the connection.
`username`	String	This is the username of the user who initiated the connection. This field will be null if the transaction was issued using embedded API. It can also be null if connection did not perform authentication.
`userAgent`	String	Name of the software that is connected. For HTTP and HTTPS connections, this information is extracted from the `User-Agent` request header. For Bolt connections, the user agent is available natively and is supplied in an initialization message.

Name	Type	Description
`serverAddress`	String	The server address this connection is connected to.
`clientAddress`	String	The client address of the connection.

Table 85. Default `userAgent` string formats

Neo4j client agent	`userAgent` default string format	Example
Cypher Shell	`"neo4j-cypher-shell/v${version}"`	`"neo4j-cypher-shell/v4.3.0"`
Neo4j Browser	`"neo4j-browser/v${version}"`	`"neo4j-browser/v4.3.0"`
Neo4j Bloom	`"neo4j-bloom/v${version}"`	`"neo4j-bloom/v1.7.0"`
Neo4j Java Driver	`"neo4j-java/x.y.z"`	`"neo4j-java/1.6.3"`
Neo4j .Net Driver	`"neo4j-dotnet/x.y"`	`"neo4j-dotnet/4.3"`
Neo4j Go Driver	`"Go Driver/x.y"`	`"Go Driver/4.3"`
Neo4j Python Driver	`"neo4j-python/x.y Python/x.y.z-a-b (<operating-system>)"`	`"neo4j-python/4.3 Python/3.7.6 (Linux)"`
Neo4j JavaScript Driver	`"neo4j-javascript/x.y.z"`	`"neo4j-javascript/4.3.0"`

Example 96. List all network connections

The following example shows that the user '**alwood**' is connected using Java driver and a Firefox web browser. The procedure call yields specific information about the connection, namely `connectionId`, `connectTime`, `connector`, `username`, `userAgent`, and `clientAddress`.

```
CALL dbms.listConnections() YIELD connectionId, connectTime, connector, username, userAgent,
clientAddress
```

```
╒══════════════════════════════════════════════════════════════════════════════════════════════
╤══════════════╤═════════════════════════╤══════════╤═══════════╤══════════
══════════════════════════════════════════════════════════════════════════╕
│"connectionId"│"connectTime"              │"connector"│"username"│"userAgent"
"clientAddress" │"status" │
╞══════════════╪═════════════════════════╪══════════╪═══════════╪══════════════════════════════
══════════════════════════════════════════════════════════════════════════╡
│"bolt-21"      │"2018-10-10T12:11:42.276Z"│"bolt"      │"alwood"   │"neo4j-java/1.6.3"
"127.0.0.1:53929"│"Running"│
├──────────────┼─────────────────────────┼──────────┼───────────┼──────────────────────────────
──────────────────────────────────────────────────────────────────────────┤
│"http-11"      │"2018-10-10T12:37:19.014Z"│"http"      │null       │"Mozilla/5.0 (Macintosh; Intel
macOS 10.13; rv:62.0) Gecko/20100101 Firefox/62.0"│"127.0.0.1:54118"│"Running"│
└──────────────┴─────────────────────────┴──────────┴───────────┴──────────────────────────────
──────────────────────────────────────────────────────────────────────────┘

2 rows
```

Terminate multiple network connections

An administrator is able to terminate within the instance all network connections with any of the given IDs. Alternatively, the current user may terminate all of their own network connections with any of the given IDs.

Syntax:

```
CALL dbms.killConnections(connectionIds)
```

Arguments:

Name	Type	Description
`ids`	List<String>	This is a list of the IDs of all the connections to be terminated.

Returns:

Name	Type	Description
`connectionId`	String	This is the ID of the terminated connection.
`username`	String	This is the username of the user who initiated the (now terminated) connection.
`message`	String	A message stating whether the connection was successfully found.

Considerations:

Bolt connections are stateful. Termination of a Bolt connection results in termination of the ongoing query/transaction.
Termination of an HTTP/HTTPS connection can terminate the ongoing HTTP/HTTPS request.

Example 97. Terminate multiple network connections

The following example shows that the administrator has terminated the connections with IDs '**bolt-37**' and '**https-11**', started by the users '**joesmith**' and '**annebrown**', respectively. The administrator also attempted to terminate the connection with ID '**http-42**' which did not exist.

```
CALL dbms.killConnections(['bolt-37', 'https-11', 'http-42'])
```

```
╔══════════════╤═══════════╤═══════════════════════════════╗
║"connectionId"│"username" │"message"                      ║
╠══════════════╪═══════════╪═══════════════════════════════╣
║"bolt-37"     │"joesmith" │"Connection found"             ║
╟──────────────┼───────────┼───────────────────────────────╢
║"https-11"    │"annebrown"│"Connection found"             ║
╟──────────────┼───────────┼───────────────────────────────╢
║"http-42"     │"n/a"      │"No connection found with this id"║
╚══════════════╧═══════════╧═══════════════════════════════╝
3 rows
```

Terminate a single network connection

An administrator is able to terminate within the instance any network connection with the given ID. Alternatively, the current user may terminate their own network connection with the given ID.

Syntax:

```
CALL dbms.killConnection(connectionId)
```

Arguments:

Name	Type	Description
`id`	String	This is the ID of the connection to be terminated.

Returns:

Name	Type	Description
`connectionId`	String	This is the ID of the terminated connection.
`username`	String	This is the username of the user who initiated the (now terminated) connection.
`message`	String	A message stating whether the connection was successfully found.

Considerations:

Bolt connections are stateful. Termination of a Bolt connection results in termination of the ongoing query/transaction.
Termination of an HTTP/HTTPS connection can terminate the ongoing HTTP/HTTPS request.

Example 98. Terminate a single network connection

The following example shows that the user '**joesmith**' has terminated his connection with the ID '**bolt-4321**'.

```
CALL dbms.killConnection('bolt-4321')
```

```
╒══════════════╤════════════╤════════════════════╕
│"connectionId"│"username"  │"message"           │
╞══════════════╪════════════╪════════════════════╡
│"bolt-4321"   │"joesmith"  │"Connection found"  │
└──────────────┴────────────┴────────────────────┘
1 row
```

The following example shows the output when trying to kill a connection with an ID that does not exist.

```
CALL dbms.killConnection('bolt-987')
```

```
╒══════════════╤════════════╤══════════════════════════════╕
│"connectionId"│"username"  │"message"                     │
╞══════════════╪════════════╪══════════════════════════════╡
│"bolt-987"    │"n/a"       │"No connection found with this id"│
└──────────────┴────────────┴──────────────────────────────┘
1 row
```

Manage background jobs

There are many types of background jobs performed in the DBMS, many of which are triggered as system jobs by the DBMS itself without any user action. For example, important background jobs include checkpoint or index population. The former is triggered by the DBMS, and the latter can be a result of a user creating or modifying an index definition.

Background jobs are of the following types:

- `IMMEDIATE` - a one-time action, triggered and run in the background.
- `DELAYED` - a one-time action, run in the background at a given point in the future.
- `PERIODIC` - a recurring action, run in the background at a given time interval.

The DBMS provides a way to show active and failed background jobs. Active jobs are those that are currently running, or are scheduled to be delayed or periodic jobs. If a background job fails, or fails to start, the details of the failure are stored in the failed jobs list. Please note that only the last 100 jobs are stored in the failed jobs list, and that this list is not persistent, so it is cleared with a DBMS restart.

Additionally, it should be noted that a single periodic job can contribute multiple times to the failed jobs list.

Listing active background jobs

An administrator can list background jobs active on an instance:

Syntax:

```
CALL dbms.scheduler.jobs()
```

Returns:

Name	Type
`jobId` ID of the job. Can be used to keep track of an active job, and to link a job to a failed job run.	String
`group` A job is a member of a job group. For example, `INDEX_POPULATION`, `LOG_ROTATION` or `RAFT_SERVER`.	String
`submitted` A timestamp for when the job was submitted, in ISO-8601 format.	String
`database` Jobs can have either a database or a DBMS scope: • For database, this column will display the name of the database. • For DBMS, this column will be blank.	String
`submitter` Jobs are either triggered as a result of user action, or as a system job by the DBMS itself. This column will contain a username for jobs triggered by users, or is otherwise blank.	String
`description` A short description of a job that, unlike `currentStateDescription`, does not change during the running of the job.	String
`type` Type of the job. The values can be `IMMEDIATE`, `DELAYED` or `PERIODIC`.	String

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

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

Tools

This chapter covers the following topics:

- Neo4j Admin and Neo4j CLI — A description of the *Neo4j Admin*, *Neo4j CLI* tools.
- Neo4j Admin commands
  - Consistency checker — How to check the consistency of a Neo4j database using Neo4j Admin.
  - Neo4j Admin report — How to collect the most common information needed for remote assessments.
  - Display store information — How to display information about a database store.
  - Memory recommendations — How to get an initial recommendation for Neo4j memory settings.
  - Import — How to import data into Neo4j using the command `neo4j-admin import`.
  - Unbind a Neo4j cluster server — How to remove cluster state data from a Neo4j server.
  - Upload to Neo4j Aura — How to upload an existing local database to a Neo4j Aura instance.
  - Migrate a database — How to migrate a Neo4j database from one store format to another or to a later `MAJOR` version of the same format.
  - Migrate the Neo4j configuration file — How to migrate a Neo4j configuration file.
- Cypher Shell — How to use the Cypher Shell.

Neo4j Admin and Neo4j CLI

Introduction

`neo4j-admin` and `neo4j` are command-line tools for managing and administering a Neo4j DBMS. Both are installed as part of the product and can be executed with a number of commands. The `neo4j` commands are equivalent to the most important commands in the `neo4j-admin` server category.

`help` and `version` are special commands and can be invoked with both `neo4j-admin` and `neo4j`. They are described in more detail in the Version command and Help command sections.

All admin commands must be invoked with the same user as Neo4j runs as. This guarantees that Neo4j will have full rights to start and work with the database files you use.

The `neo4j-admin` tool

The `neo4j-admin` command-line tool is located in the *bin* directory.

General synopsis

`neo4j-admin` has the following general synopsis:

```
neo4j-admin [category] [command] [subcommand]
```

`neo4j-admin` commands per category

All administration commands, except for `help` and `version`, are organized into the following three categories:

- `dbms` - DBMS-wide (for single and clustered environments) administration tasks
- `server` - server-wide administration tasks
- `database` - database-specific administration tasks

Table 86. Available commands per category

Category	Command	Description
`dbms`	`set-default-admin`	Sets the default admin user when no roles are present.
	`set-initial-password`	Sets the initial password of the initial admin user (`neo4j`). For details, see Set an initial password.
	`unbind-system-db`	Removes and archives the cluster state of the `system` database so the instance can rebind to a new cluster state of the `system` database.

Category	Command	Description
`server`	`console`	Starts DBMS server in the console.
	`get-id`	Displays the server ID of an instance. The server ID can be used to specify a server in Cypher commands.
	`memory-recommendation`	Prints recommendations for Neo4j heap and page cache memory usage. For details, see Memory recommendations.
	`migrate-configuration`	Migrates server configuration from the previous major version.
	`report`	Produces a ZIP/TAR of the most common information needed for remote assessments. For details, see Neo4j Admin report.
	`restart`	Restarts the server daemon.
	`start`	Starts the server as a daemon.
	`status`	Gets the status of the server.
	`stop`	Stops the server daemon.
	`unbind`	Removes cluster state data from a stopped Neo4j server. For details, see Unbind a Neo4j cluster server.
	`windows-service`	A command whose subcommands can be used to install, uninstall, and update Neo4j as a Windows service.

Category	Command	Description
database	aggregate-backup	Aggregates a chain of backup artifacts into a single artifact.
	backup	Performs an online backup from a running Neo4j enterprise server.
	check	Checks the consistency of a database. For details, see Consistency checker.
	copy	Copies a database and optionally applies filters. For details, see Copy a database store.
	dump	Dumps a database into a single-file archive.
	import	Imports a collection of CSV files. For details, see Import.
	info	Prints information about a Neo4j database store. For details, see Display store information.
	load	Loads a database from an archive created with the dump command.
	migrate	Migrates a database from one store format to another or between versions of the same format.
	restore	Restores a backed up database.
	upload	Pushes a local database to a Neo4j Aura instance. For details, see Upload to Neo4j AuraDB.

The neo4j tool

The neo4j command-line tool is located in the bin directory.

General synopsis

`neo4j` has the following general synopsis:

`neo4j [command]`

`neo4j` commands

The command is an alias for the most important commands in the `neo4j-admin server` category.

Table 87. Equivalence between `neo4j` and `neo4j-admin` commands

`neo4j` command	Equivalent `neo4j-admin` command
`neo4j console`	`neo4j-admin server console`
`neo4j restart`	`neo4j-admin server restart`
`neo4j start`	`neo4j-admin server start`
`neo4j status`	`neo4j-admin server status`
`neo4j stop`	`neo4j-admin server stop`
`neo4j windows-service`	`neo4j-admin server windows-service`

Version command

Version can be obtained by invoking the `version` command, `--version` command option, or its short alternative `-V`, on the root level of both `neo4j` and `neo4j-admin` commands. For example, `neo4j --version`, `neo4j-admin -V`, `neo4j-admin version`, or `neo4j version`.

Help command

Help can be obtained by invoking the `help` command, `--help` command option, or its short alternative `-h`, with both `neo4j` and `neo4j-admin` commands. `--help` and `-h` options can be invoked on any level, namely root, category, command, and subcommand. For example, `neo4j --help`, `neo4j [command] -h`, `neo4j-admin -h`, `neo4j-admin [category] --help`, or `neo4j-admin [category] [command] [subcommand] -h`.

The help command can be invoked on any level except the last one, which means command-level for commands that do not have subcommands or subcommand level for commands with subcommands. The help command also accepts a parameter. For example, `neo4j help`, `neo4j-admin help`, `neo4j-admin [category] help`, `neo4j-admin help [category]`, `neo4j help [command]`, or `neo4j-admin [category] [command ] help [subcommand]`.

Configuration

Administration operations use the configuration specified in the *neo4j.conf* file. Sharing configuration between the DBMS and its administration tasks makes sense as most settings are the same. In some cases, however, it is better to override some settings specified in *neo4j.conf* by configuring the tasks (instead of updating the config settings in the *neo4j.conf* file) because administration tasks generally use fewer resources than the DBMS. For instance, if the page cache of your DBMS is configured to a very high value in *neo4j.conf*, and you want to override this because the admin tasks like backup do not need so much memory, you provide configuration for the admin tasks instead of updating the page cache setting in the *neo4j.conf* file.

There are several options for overriding settings specified in the *neo4j.conf* file using administration tasks:

- `--additional-config` option — almost all administration commands support the `--additional-config` option, which you can use to provide a path to a file with additional configuration.
- *neo4j-admin.conf* — a configuration file located in the same directory as the `neo4j.conf` file, which you can use to provide administration-task-specific settings.
- Some commands also support a command-specific configuration file. Such files are also looked for in the same directory as the *neo4j.conf* file. The following table lists command-specific configuration files:

Table 88. Command-specific configuration files

Command	Configuration file
`neo4j-admin database backup`	`neo4j-admin-database-backup.conf`
`neo4j-admin database check`	`neo4j-admin-database-check.conf`
`neo4j-admin database copy`	`neo4j-admin-database-copy.conf`
`neo4j-admin database dump`	`neo4j-admin-database-dump.conf`
`neo4j-admin database import`	`neo4j-admin-database-import.conf`
`neo4j-admin database load`	`neo4j-admin-database-load.conf`
`neo4j-admin database migrate`	`neo4j-admin-database-migrate.conf`
`neo4j-admin database restore`	`neo4j-admin-database-resore.conf`

All four configuration sources are optional and settings for administration commands are resolved from them with the following descending priority:

1. `--additional-config` option
2. command-specific configuration file
3. `neo4j-admin.conf`

4. `neo4j.conf`

> The commands for launching the DBMS, `neo4j start` and `neo4j console`, must be configured only in the *neo4j.conf* file.

Environment variables

Neo4j Admin can also use the following environment variables:

Environment variable	Description
`NEO4J_DEBUG`	Set to anything to enable debug output.
`NEO4J_HOME`	Neo4j home directory.
`NEO4J_CONF`	Path to the directory that contains *neo4j.conf*.
`HEAP_SIZE`	Set JVM maximum heap size during command execution. Takes a number and a unit, for example, 512m.
`JAVA_OPTS`	Additional JVM arguments.

If set, `HEAP_SIZE` and `JAVA_OPTS` override all relevant settings specified in the configuration file.

Exit codes

When `neo4j-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.

Consistency checker

You can use the `neo4j-admin database check` command to check the consistency of a database, a dump, or a backup. 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, a dump, or a backup as this slows the process down significantly.

Syntax

The `neo4j-admin database check` command has the following syntax:

```
neo4j-admin database check [-h]
                           [--expand-commands]
                           [--force]
                           [--verbose]
                           [--check-counts[=true|false]]
                           [--check-graph[=true|false]]
                           [--check-indexes[=true|false]]
                           [--check-property-owners[=true|false]]
                           [--additional-config=<file>]
                           [--max-off-heap-memory=<size>]
                           [--report-path=<path>]
                           [--threads=<number of threads>]
                           [[--from-path-data=<path> --from-path-txn=<path>] | [--from-path=<path> [--
temp-path=<path>]]]
                           <database>
```

`<database>` — Name of the database to check.

Options

The `neo4j-admin database check` command has the following options:

Option	Possible values	Description
`-h, --help`		Show this help message and exit.
`--expand-commands`		Allow command expansion in config value evaluation.
`--force`	`true` or `false` (Default: `false`)	Force a consistency check to be run, despite resources, and may run a more thorough check.
`--verbose`	`true` or `false` (Default: `false`)	Enable verbose output.
`--check-graph`	`true` or `false` (Default: `true`)	Perform checks between nodes, relationships, properties, types, and tokens.
`--check-counts`	`true` or `false`	Perform consistency checks on the counts. Requires the `check-graph` option to be `true` (which is by default) and will implicitly enable `<check-graph>` if it is not explicitly disabled. Default: `<check-graph>`.
`--check-indexes`	`true` or `false` (Default: `true`)	Perform checks on indexes.
`--check-property-owners`	`true` or `false` (Default: `false`)	Perform consistency checks on the ownership of properties. Requires the `check-graph` option to be `true` (which is by default) and will implicitly enable `<check-graph>` if it is not explicitly disabled.

Option	Possible values	Description
`--additional-config`	`<file>`	Path to a configuration file that contains additional configuration options.
`--max-off-heap-memory`	`<size>` (Default: `90%`)	Maximum memory that `neo4j-admin`` can use for page cache and various caching data structures to improve performance. Value can be plain numbers, like `10000000` or e.g. `20G` for 20 gigabytes, or even e.g. `70%`, which will amount to 70% of currently free memory on the machine.
`--report-path`	`<path>` (Default: `.`)	Path to where a consistency report will be written. Interpreted as a directory, unless it has an extension of .report.
`--threads`	`<number of threads>` (Default: the number of CPUs on the machine)	Number of threads used to check the consistency.
`--from-path-data`	`<path>` (Default: `,` `server.directories.data`/databases)	Path to the databases directory, containing the database directory to source from.
`--from-path-txn`	`<path>` (Default: `server.directories.transaction.logs.root`)	Path to the transactions_ directory, containing the transaction directory for the database to source from.
`--from-path`	`<path>`	Path to the directory containing dump/backup artifacts whose consistency will be checked.
`--temp-path`	`<path>`	Path to a directory to be used as a staging area to extract dump/backup artifacts if needed.
`<database>`		Name of the database to check.

Output

If the consistency checker does not find errors, it exits cleanly and does not produce a report. If the consistency checker finds errors, it exits with an exit code other than `0` and writes a report file with a name in the format `inconsistencies-YYYY-MM-DD.HH24.MI.SS.report`. The location of the report file is the current working directory, or as specified by the parameter `report-path`.

Example

Example 99. Run the consistency checker

Note that the database must be stopped first.

```
$neo4j-home> bin/neo4j-admin database check neo4j

Running consistency check with max off-heap:618.6MiB
  Store size:160.0KiB
  Allocated page cache:160.0KiB
  Off-heap memory for caching:618.5MiB
ID Generator consistency check
....................  10%
....................  20%
....................  30%
....................  40%
....................  50%
....................  60%
....................  70%
....................  80%
....................  90%
.................... 100%
Index structure consistency check
....................  10%
....................  20%
....................  30%
....................  40%
....................  50%
....................  60%
....................  70%
....................  80%
....................  90%
.................... 100%
Consistency check
....................  10%
....................  20%
....................  30%
....................  40%
....................  50%
....................  60%
....................  70%
....................  80%
....................  90%
.................... 100%
```

Run with the `--from-path` option to check the consistency of a backup or a dump.

```
bin/neo4j-admin database check --from-path=<directory-with-backup-or-dump> neo4j
```

`neo4j-admin database check` cannot be applied to composite databases. It must be run directly on the databases that are part of a composite database.

Neo4j Admin report

You can use the `neo4j-admin server report` command to collect information about a Neo4j installation and save it to an archive.

Syntax

The `neo4j-admin server report` command has the following syntax:

```
neo4j-admin server report [-h]
                          [--expand-commands]
                          [--list]
                          [--verbose]
                          [--ignore-disk-space-check[=true|false]]
                          [--additional-config=<file>]
                          [--to-path=<path>]
                          [<classifier>...]
```

The intended usage of the report tool is to simplify the support process by collecting the relevant information in a standard way. This tool does not send any information automatically. To share this information with the Neo4j Support organization, you have to send it manually.

Options and classifiers

The `neo4j-admin server report` command has the following options and classifiers:

Table 89. Options

Option	Possible values	Description
`-h, --help`		Show this help message and exit.
`--expand-commands`		Allow command expansion in config value evaluation.
`--list`		List all available classifiers.
`--verbose`		Enable verbose output.
`--ignore-disk-space-check`	`true` or `false` (Default: `false`)	Ignore disk full warning.
`--to-path`	`<path>` (Default: /tmp)	Destination directory for reports.

By default, the tool tries to estimate the final size of the report and uses that to assert that there is enough disk space available for it. If there is not enough available space, the tool aborts. However, this estimation is pessimistic and does not consider the compression. Therefore, if you are confident that you do have enough disk space, you can disable this check with the option `--ignore-disk-space-check`.

Table 90. Classifiers

Classifier	Online	Description
`all`		Include all of the available classifiers.
`ccstate`		Include the current cluster state.
`config`		Include Neo4j configuration files.
`heap`	✓	Include a heap dump.
`logs`		Include log files, e.g., *debug.log*, *neo4j.log*, etc.
`metrics`		Include the collected metrics.
`plugins`		Include a text view of the plugin directory (no files are collected).
`ps`		Include a list of running processes.
`raft`		Include the raft log.
`sysprop`	✓	Include a list of Java system properties.
`threads`	✓	Include a thread dump of the running instance.
`tree`		Include a text view of the folder structure of the data directory (no files are collected).
`tx`		Include transaction logs.

The classifiers marked as *Online* work only when you have a running Neo4j instance that the tool can find.

If no classifiers are specified, the following classifiers are used: `logs`, `config`, `plugins`, `tree`, `metrics`, `threads`, `sysprop`, `ps`, and `version`.

The reporting tool does not read any data from your database. However, the heap, the raft logs, and the transaction logs may contain data. Additionally, even though the standard *neo4j.conf* file does not contain password information, for specific configurations, it may have this type of information. Therefore, be aware of your organization's data security rules before using the classifiers `heap`, `tx`, `raft`, and `config`.

Examples

This tool uses the Java Attach API to gather data from a running Neo4j instance. Therefore, it requires the Java JDK to run properly.

Example 100. Invoke `neo4j-admin server report` using the default classifiers

The following command gathers information about the Neo4j instance using the default classifiers and saves it to the default location:

```
$neo4j-home> bin/neo4j-admin server report
```

Example 101. Invoke `neo4j-admin server report` using all classifiers

The following command gathers information about the Neo4j instance using all classifiers and saves it to a specified location:

```
$neo4j-home> bin/neo4j-admin server report --to-path=./report all
```

Example 102. Invoke `neo4j-admin server report` to gather only logs and thread dumps

The following command gathers only logs and thread dumps from the running Neo4j instance and saves it to a specified location:

```
$neo4j-home> bin/neo4j-admin server report --to-path=./report threads logs
```

Display store information

You can use the `neo4j-admin database info` command to get the following information about the store format of a given database store:

- The store format version.
- When the store format version was introduced.
- Whether the store format needs to be migrated to a newer version.

The `neo4j-admin database info` command is located in the bin directory.

Syntax

The `neo4j-admin database info` command should be invoked against an **offline** database store or a backup and has the following syntax:

```
neo4j-admin database info [-h] [--expand-commands]
                               [--verbose]
                               [--additional-config=<file>]
                               [--format=text|json]
                               [--from-path=<path>]
                               [<database>]
```

Options

The `neo4j-admin database info` command has the following options:

Table 91. Options

Name	Possible values	Description
`-h, --help`		Show this help message and exit.
`--expand-commands`		Allow command expansion in config value evaluation.
`--verbose`		Enable verbose output.
`--additional-config`	<file>	Configuration file with additional configuration.
`--format`	`text` or `json`	The format to use for the returned information. Default is `text`.
`--from-path`	<path>	Path to databases directory.
`<database>`		Name of the database to show info for. Can contain `*` and `?` for globbing. Default is `*`.

Examples

Example 103. Invoke `neo4j-admin database info` against a database store

```
bin/neo4j-admin database info mygraph.db
```

Output:

```
Store format version:         record-aligned-1.1
Store format introduced in:   5.0.0
```

Example 104. Invoke `neo4j-admin database info` against all databases

The command can also be invoked against all databases, as follows:

`neo4j-admin database info --from-path=<databases-directory>`

```
bin/neo4j-admin database info data/databases
```

Output:

```
Database name:                foo
Database in use:              false
Store format version:         record-aligned-1.1
Store format introduced in:   5.0.0
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 database info` against a database and output JSON

If you are parsing the results of this command you may use the `--format=json` option to receive the output as JSON. All the same fields are included and all values are strings.

```
bin/neo4j-admin database info --from-path data/databases --format=json foo
```

Output:

```
{"databaseName":"foo",
"inUse":"false",
"storeFormat”:"record-aligned-1.1",
"storeFormatIntroduced”:"5.0.0",
"lastCommittedTransaction":"2",
"recoveryRequired":"true"}
```

Store formats and entity limits

The store formats are:

- `aligned`
- `standard`
- `high_limit` Enterprise edition

Aligned

Table 92. Store versions — `aligned`

Store Format Name	Store Format Version	Introduced in	Unsupported from
`ALIGNED_V5_0`	`record-aligned-1.1`	`5.0.0`	
`ALIGNED_V4_3`	`AF4.3.0`	`4.3.0`	`5.0.0`
`ALIGNED_V4_1`	`AF4.1.a`	`4.1.0`	`5.0.0`

Table 93. Store limits — `aligned`

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)

Standard

Table 94. Store versions — `standard`

Store Format Name	Store Format Version	Introduced in	Unsupported from
`STANDARD_V5_0`	`record-standard-1.1`	`5.0.0`	
`STANDARD_V4_3`	`SF4.3.0`	`4.3.0`	`5.0.0`
`STANDARD_V4_0`	`SF4.0.0`	`4.0.0`	`5.0.0`
`STANDARD_V3_4`	`v0.A.9`	`3.4.0`	`5.0.0`

Table 95. 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)

High_limit Enterprise edition

Table 96. Store versions — `high_limit`

Store Format Name	Store Format Version	Introduced in	Unsupported from
`HIGH_LIMIT_V5_0`	`record-high_limit-1.1`	`5.0.0`	
`HIGH_LIMIT_V4_3_0`	`HL4.3.0`	`4.3.0`	`5.0.0`
`HIGH_LIMIT_V4_0_0`	`HL4.0.0`	`4.0.0`	`5.0.0`
`HIGH_LIMIT_V3_4_0`	`vE.H.4`	`3.4.0`	`5.0.0`
`HIGH_LIMIT_V3_2_0`	`vE.H.3`	`3.2.0`	`5.0.0`
`HIGH_LIMIT_V3_1_0`	`vE.H.2`	`3.1.0`	`5.0.0`
`HIGH_LIMIT_V3_0_6`	`vE.H.0b`	`3.0.6`	`5.0.0`
`HIGH_LIMIT_V3_0_0`	`vE.H.0`	`3.0.0`	`5.0.0`

Table 97. 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)

Memory recommendations

You can use the `neo4j-admin server memory-recommendation` command to get an initial recommendation on how to configure the memory parameters of your Neo4j DBMS.

Syntax

The `neo4j-admin server memory-recommendation` command has the following syntax:

```
neo4j-admin server memory-recommendation [-h]
                                          [--docker]
                                          [--expand-commands]
                                          [--verbose]
                                          [--additional-config=<file>]
                                          [--memory=<size>]
```

The command gives heuristic memory settings recommendations for the Neo4j JVM heap and pagecache. It either uses the total system memory or the amount of memory specified in the `--memory` argument. The heuristic also 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 so you can copy and paste it 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

The `neo4j-admin server memory-recommendation` command has the following options:

Table 98. Options

Option	Default	Description
`-h`, `--help`		Show this help message and exit.
`--docker`		Enable output formatted as 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 that file to a Docker container using `--env-file` Docker option.
`--expand-commands`		Expand the commands to be run.
`--verbose`		Enable verbose output.
`--additional-config`	`<file>`	Configuration file to supply additional configuration in.
`--memory`	`<size> (Default: The memory capacity of the machine.)`	Recommend memory settings for the given amount of memory, instead of the total memory of the system running the command. Valid units are: `k`, `K`, `m`, `M`, `g`, `G`.

Considerations

The `neo4j-admin server memory-recommendation` command calculates a valid starting point for the Neo4j memory settings, based on the provided memory. The specific conditions for your use case may warrant adjustment of these values. See Memory configuration for a description of the memory settings in Neo4j.

Example

Example 106. Use the `memory-recommendation` command of `neo4j-admin`

The following example illustrates how `neo4j-admin server memory-recommendation` provides a recommendation on how to use 16g of memory:

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

...
...
...
# Based on the above, the following memory settings are recommended:
server.memory.heap.initial_size=5g
server.memory.heap.max_size=5g
server.memory.pagecache.size=7g
```

For an example of how to use the `neo4j-admin server memory-recommendation` command, see Inspect the memory settings of all databases in a DBMS.

Import

There are two ways to import data into a Neo4j database. You can use the Cypher command `LOAD CSV` or the `neo4j-admin database import` command.

With `LOAD CSV`, you can import small to medium-sized CSV files into an *existing* database. `LOAD CSV` can be run as many times as needed and does not require an empty database.

With the `neo4j-admin database import` command, you can do batch imports of large amounts of data from CSV files. It is generally faster than `LOAD CSV` because it is run against a stopped or a non-existent empty database.

The `neo4j-admin database import` command has two modes:

- *full* — used to initially import data into a non-existent empty database.
- *incremental* — used to incrementally import data into an existing database.

The user running `neo4j-admin database import` must have `WRITE` capabilities into `server.directories.data` and `server.directories.log`.

This section describes the `neo4j-admin database import` option.

For information on `LOAD CSV`, see the Cypher Manual → `LOAD CSV`. For in-depth examples of using the command `neo4j-admin database import`, refer to the Tutorials → Neo4j Admin import.

These are some things you need to keep in mind when creating your input files:

- Fields are comma-separated by default but a different delimiter can be specified.
- All files must use the same delimiter.
- Multiple data sources can be used for both nodes and relationships.

* A data source can optionally be provided using multiple files.
* A separate file with a header that provides information on the data fields, must be the first specified file of each data source.
* Fields without corresponding information in the header will not be read.
* UTF-8 encoding is used.
* By default, the importer trims extra whitespace at the beginning and end of strings. Quote your data to preserve leading and trailing whitespaces.

Indexes and constraints

Indexes and constraints are not created during the import. Instead, you have to add these afterward (see Cypher Manual → Indexes).

Full import

Syntax

The syntax for importing a set of CSV files is:

```
neo4j-admin database import full [-h]
                                 [--expand-commands]
                                 [--verbose]
                                 [--auto-skip-subsequent-headers[=true|false]]
                                 [--ignore-empty-strings[=true|false]]
                                 [--ignore-extra-columns[=true|false]]
                                 [--legacy-style-quoting[=true|false]]
                                 [--multiline-fields[=true|false]]
                                 [--normalize-types[=true|false]]
                                 [--overwrite-destination[=true|false]]
                                 [--skip-bad-entries-logging[=true|false]]
                                 [--skip-bad-relationships[=true|false]]
                                 [--skip-duplicate-nodes[=true|false]]
                                 [--trim-strings[=true|false]]
                                 [--additional-config=<file>]
                                 [--array-delimiter=<char>]
                                 [--bad-tolerance=<num>]
                                 [--delimiter=<char>]
                                 [--format=<format>]
                                 [--high-parallel-io=on|off|auto]
                                 [--id-type=string|integer|actual]
                                 [--input-encoding=<character-set>]
                                 [--max-off-heap-memory=<size>]
                                 [--quote=<char>]
                                 [--read-buffer-size=<size>]
                                 [--report-file=<path>]
                                 [--threads=<num>]
                                 --nodes=[<label>[:<label>]...=]<files>...
                                 [--nodes=[<label>[:<label>]...=]<files>...]...
                                 [--relationships=[<type>=]<files>...]...
                                 <database>
```

Examples

Example 107. Import data from CSV files

Assume that you have formatted your data as per CSV header format so that you have it in six different files:

1. `movies_header.csv`
2. `movies.csv`
3. `actors_header.csv`
4. `actors.csv`
5. `roles_header.csv`
6. `roles.csv`

The following command imports the three datasets:

```
neo4j_home$ bin/neo4j-admin database import full --nodes import/movies_header.csv,import/movies.csv \
--nodes import/actors_header.csv,import/actors.csv \
--relationships import/roles_header.csv,import/roles.csv
```

Example 108. Import data from CSV files using regular expression

Assume that you want to include a header and then multiple files that match a pattern, e.g. containing numbers. In this case, a regular expression can be used. It is guaranteed that groups of digits will be sorted in numerical order, as opposed to lexicograghic order.

For example:

```
neo4j_home$ bin/neo4j-admin database import full --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 database import full --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.

Parameters and options

`<database>`

Name of the database to import. If the database does not exist prior to importing, you must create it subsequently using `CREATE DATABASE`.

Default: `neo4j`.

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.

Table 99. `neo4j-admin database import full` options

Option	Description	Default
`--additional-config=<file>`	Path to a configuration file that contains additional configuration options.	
`--array-delimiter=<char>`	Delimiter character between array elements within a value in CSV data. Also accepts 'TAB' and e.g. 'U+20AC' for specifying the character using Unicode. • ASCII character — e.g. `--array-delimiter=";"`. • `\ID` — Unicode character with ID, e.g. `--array-delimiter="\59"`. • `U+XXXX` — Unicode character specified with 4 HEX characters, e.g. `--array-delimiter="U+20AC"`. • `\t` — horizontal tabulation (HT), e.g. `--array-delimiter="\t"`. For horizontal tabulation (HT), use `\t` or the Unicode character ID `\9`. Unicode character ID can be used if prepended by `\`.	`;`
`--auto-skip-subsequent-headers`	Automatically skip accidental header lines in subsequent files in file groups with more than one file.	`false`
`--bad-tolerance=<num>`	Number of bad entries before the import is considered failed. This tolerance threshold is about relationships referring to missing nodes. Format errors in input data are still treated as errors.	`1000`

Option	Description	Default
`--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 `\`.	`,`
`--expand-commands`	Allow command expansion in config value evaluation.	
`--format=<format>`	Name of database format. Imported database will be created of the specified format or use format from configuration if not specified.	
`-h, --help`	Show this help message and exit.	
`--high-parallel-io[=on/off/auto]`	Ignore environment-based heuristics and specify whether the target storage subsystem can support parallel IO with high throughput. Typically this is `on` for SSDs, large raid arrays, and network-attached storage.	`auto`
`--id-type=<string|integer|actual>`	Each node must provide a unique ID 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	`string`
`--ignore-empty-strings[=<true/false>]`	Determines whether or not empty string fields, such as `""`, from input source are ignored (treated as null).	`false`

Option	Description	Default
`--ignore-extra-columns[=<true/false>]`	If unspecified columns should be ignored during the import.	`false`
`--input-encoding=<character-set>`	Character set that input data is encoded in.	`UTF-8`
`--legacy-style-quoting[=<true/false>]`	Determines whether or not backslash-escaped quote e.g. `"` is interpreted as an inner quote.	`false`
`--max-off-heap-memory=<size>`	Maximum off-heap memory that `neo4j-admin` can use for various data structures and caching to improve performance. Values can be plain numbers such as `10000000`, or `20G` for 20 gigabytes. It can also be specified as a percentage of the available memory, for example `70%`.	`90%`
`--multiline-fields[=<true/false>]`	Determines whether or not fields from the input source can span multiple lines, i.e. contain newline characters. Setting `--multiline-fields=true` can severely degrade the performance of the importer. Therefore, use it with care, especially with large imports.	`false`
`--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.	
`--normalize-types[=<true/false>]`	Determines whether or not to normalize property types to Cypher types, e.g. `int` becomes `long` and `float` becomes `double`.	`true`

Option	Description	Default
`--overwrite-destination[=<true/false>]`	Deletes any existing database files prior to the import. Use `--overwrite-destination=true` to delete all files of the specified database and then import new data. For example: • When using Neo4j Community Edition. Since the Community Edition only supports one database and does not support `DROP DATABASE name`, the only way to re-import data using `neo4j-admin database import` is to use `--overwrite-destination=true`. • When you first want to see how the data would get imported and maybe do some tweaking before you import your actual data. For example, you can first import a small batch of data (e.g., 1000 rows) and examine it. And then, tweak your actual data (e.g., 10 million rows) and use the option `--overwrite-destination=true` to re-import it.	`false`
`--quote=<char>`	Character to treat as quotation character for values in CSV data. Quotes can be escaped as per RFC 4180 by doubling them, for example `""` would be interpreted as a literal `"`. You cannot escape using `\`.	`"`
`--read-buffer-size=<size>`	Size of each buffer for reading input data. It has to at least be large enough to hold the biggest single value in the input data. Value can be a plain number or byte units string, e.g. `128k`, `1m`.	`4194304`
`--relationships=[<type>=]<files>…`	Relationship CSV header and data. • Multiple files will be logically seen as one big file from the perspective of the importer. • The first line must contain the header. • Multiple data sources like these can be specified in one import, where each data source has its own header. • Files can also be specified using regular expressions. For an example, see Import data from CSV files using regular expression.	

Option	Description	Default
`--report-file=<path>`	File in which to store the report of the csv-import. The location of the import log file can be controlled using the `--report-file` option. If you run large imports of CSV files that have low data quality, the import log file can grow very large. For example, CSV files that contain duplicate node IDs, or that attempt to create relationships between non-existent nodes, could be classed as having low data quality. In these cases, you may wish to direct the output to a location that can handle the large log file. If you are running on a UNIX-like system and you are not interested in the output, you can get rid of it altogether by directing the report file to `/dev/null`. If you need to debug the import, it might be useful to collect the stack trace. This is done by using `--verbose` option.	`import.report`
`--skip-bad-entries-logging[=<true/false>]`	Determines whether or not to skip logging bad entries detected during import.	`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 the node that was not specified by the node input data. Skipped relationships will be logged, containing at most the number of entities specified by `--bad-tolerance`, unless otherwise specified by the `--skip-bad-entries-logging` option.	`false`
`--skip-duplicate-nodes[=<true/false>]`	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.	`false`

Option	Description	Default
`--threads=<num>` Advanced	Max number of worker threads used by the importer. Defaults to the number of available processors reported by the JVM. There is a certain amount of minimum threads needed, so for that reason, there is no lower bound for this value. For optimal performance, this value shouldn't be greater than the number of available processors.	
`--trim-strings[=<true/false>]`	Determines whether or not strings should be trimmed for whitespaces.	`false`
`--verbose`	Enable verbose output.	

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 results in a graph that is larger than 34 billion nodes, 34 billion relationships, or 68 billion properties, you will need to configure the importer to use the `high_limit` record format. This is achieved by using the `format` option of the import command and setting the value to `high_limit`:

```
neo4j-admin database import full --format=high_limit
```

The `high_limit` format is available for Enterprise Edition only.

Incremental import

The `neo4j-admin database import incremental` command imports a large amount of data from CSV files into an existing, non-empty database.

Incremental import needs to be used with care. These options should not be used for experimentation.

You need to append any incremental import command with `--force`.

For more information, please contact Neo4j Professional Services.

Syntax

```
neo4j-admin database import incremental [-h]
                                        [--expand-commands]
                                        --force
                                        [--verbose]
                                        [--auto-skip-subsequent-headers[=true|false]]
                                        [--ignore-empty-strings[=true|false]]
                                        [--ignore-extra-columns[=true|false]]
                                        [--legacy-style-quoting[=true|false]]
                                        [--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=<file>]
                                        [--array-delimiter=<char>]
                                        [--bad-tolerance=<num>]
                                        [--delimiter=<char>]
                                        [--high-parallel-io=on|off|auto]
                                        [--id-type=string|integer|actual]
                                        [--input-encoding=<character-set>]
                                        [--max-off-heap-memory=<size>]
                                        [--quote=<char>]
                                        [--read-buffer-size=<size>]
                                        [--report-file=<path>]
                                        [--stage=all|prepare|build|merge]
                                        [--threads=<num>]
                                        --nodes=[<label>[:<label>]...=]<files>...
                                        [--nodes=[<label>[:<label>]...=]<files>...]...
                                        [--relationships=[<type>=]<files>...]...
                                        <database>
```

Usage and limitations

The incremental import command can be used to add:

- New nodes with labels and properties.
- New relationships between existing or new nodes.

The incremental import command cannot be used to:

- Add new properties to existing nodes or relationships.
- Update or delete properties in nodes or relationships.
- Update or delete labels in nodes.
- Delete existing nodes and relationships.

The importer works well on single instances. In clustering environments with multiple copies of the database, the updated database must be reseeded.

Examples

There are two ways of importing data incrementally:

- If downtime is not a concern, you can run a single command with the option `--stage=all`. This option requires the database to be stopped.

- If you cannot afford a full downtime of your database, you can run the import in three stages:
  - *prepare* stage:

    During this stage, the import tool analyzes the CSV headers and copies the relevant data over to the new increment database path. The import command is run with the option `--stage=prepare` and the database must be stopped.

  - *build* stage:

    During this stage, the import tool imports the data into the database. This is the longest stage and you can put the database in read-only mode to allow read access. The import command is run with the option `--stage=build`.

  - *merge* stage:

    During this stage, the import tool merges the new with the existing data in the database. It also updates the affected indexes and upholds the affected property uniqueness constraints and property existence constraints. The import command is run with the option `--stage=merge` and the database must be stopped.

Example 110. Incremental import in a single command

```
neo4j@system> STOP DATABASE db1 WAIT;
...
$ bin/neo4j-admin database import incremental --stage=all --nodes=N1=../../raw-data/incremental-
import/b.csv db1
```

Example 111. Incremental import in stages

1. `prepare` stage:

   a. Stop the database with the `WAIT` option to ensure a checkpoint happens before you run the incremental import command. The database must be stopped to run `--stage=prepare`.

   ```
   neo4j@system> STOP DATABASE db1 WAIT;
   ```

   b. Run the incremental import command with the `--stage=prepare` option:

   ```
   $ bin/neo4j-admin database import incremental --stage=prepare --nodes=N1=../../raw-
   data/incremental-import/c.csv db1
   ```

2. `build` stage:

   a. Put the database in read-only mode:

   ```
   ALTER DATABASE db1 SET ACCESS READ ONLY;
   ```

   b. Run the incremental import command with the `--stage=build` option:

   ```
   $ bin/neo4j-admin database import incremental --stage=build --nodes=N1=../../raw-
   data/incremental-import/c.csv db1
   ```

3. `merge` stage:

   It is not necessary to include the `--nodes` or `--relationships` options when using `--stage=merge`.

   a. Stop the database with the `WAIT` option to ensure a checkpoint happens before you run the incremental import command.

   ```
   neo4j@system> STOP DATABASE db1 WAIT;
   ```

   b. Run the incremental import command with the `--stage=merge` option:

   ```
   $ bin/neo4j-admin database import incremental --stage=merge db1
   ```

Parameters and options

`<database>`

Name of the database to import. If the database does not exist prior to importing, you must create it subsequently using `CREATE DATABASE`.

Default: `neo4j`.

Table 100. `neo4j-admin database import incremental` *options*

Option	Description	Default
`--additional-config=<file>`	Configuration file with additional configuration.	
`--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 `\`.	`;`
`--auto-skip-subsequent-headers[=<true/false>]`	Automatically skip accidental header lines in subsequent files in file groups with more than one file.	`false`
`--bad-tolerance=<num>`	Number of bad entries before the import is considered failed. This tolerance threshold is about relationships referring to missing nodes. Format errors in input data are still treated as errors.	`1000`
`--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 `\`.	`,`

Option	Description	Default
`--expand-commands`	Allow command expansion in config value evaluation.	
`--force`	Confirm incremental import by setting this flag.	
`-h, --help`	Show this help message and exit.	
`--high-parallel-io=on/off/auto`	Ignore environment-based heuristics and specify whether the target storage subsystem can support parallel IO with high throughput. Typically this is `on` for SSDs, large raid arrays, and network-attached storage.	`auto`
`--id-type=string|integer|actual`	Each node must provide a unique ID 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	`string`
`--ignore-empty-strings[=<true/false>]`	Determines whether or not empty string fields, such as `""`, from input source are ignored (treated as null).	`false`
`--ignore-extra-columns[=<true/false>]`	If unspecified columns should be ignored during the import.	`false`
`--input-encoding=<character-set>`	Character set that input data is encoded in.	`UTF-8`
`--legacy-style-quoting[=<true/false>]`	Determines whether or not backslash-escaped quote e.g. `\"` is interpreted as an inner quote.	`false`
`--max-off-heap-memory=<size>`	Maximum off-heap memory that `neo4j-admin` can use for various data structures and caching to improve performance. Values can be plain numbers such as `10000000`, or `20G` for 20 gigabytes. It can also be specified as a percentage of the available memory, for example `70%`.	`90%`

Option	Description	Default
`--multiline-fields[=<true/false>]`	Determines whether or not fields from the input source can span multiple lines, i.e. contain newline characters. Setting `--multiline-fields=true` can severely degrade the performance of the importer. Therefore, use it with care, especially with large imports.	`false`
`--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.	
`--normalize-types[=<true/false>]`	Determines whether or not to normalize property types to Cypher types, e.g. `int` becomes `long` and `float` becomes `double`.	`true`
`--quote=<char>`	Character to treat as quotation character for values in CSV data. Quotes can be escaped as per RFC 4180 by doubling them, for example `""` would be interpreted as a literal `"`. You cannot escape using `\`.	`"`
`--read-buffer-size=<size>`	Size of each buffer for reading input data. It has to at least be large enough to hold the biggest single value in the input data. Value can be a plain number or byte units string, e.g. `128k`, `1m`.	`4194304`

Option	Description	Default
`--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.	
`--report-file=<path>`	File in which to store the report of the csv-import. The location of the import log file can be controlled using the `--report-file` option. If you run large imports of CSV files that have low data quality, the import log file can grow very large. For example, CSV files that contain duplicate node IDs, or that attempt to create relationships between non-existent nodes, could be classed as having low data quality. In these cases, you may wish to direct the output to a location that can handle the large log file. If you are running on a UNIX-like system and you are not interested in the output, you can get rid of it altogether by directing the report file to `/dev/null`. If you need to debug the import, it might be useful to collect the stack trace. This is done by using `--verbose` option.	`import.report`
`--skip-bad-entries-logging[=<true/false>]`	Determines whether or not to skip logging bad entries detected during import.	`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 a node that was not specified by the node input data. Skipped relationships will be logged, containing at most the number of entities specified by `--bad-tolerance`, unless otherwise specified by the `--skip-bad-entries-logging` option.	`false`

Option	Description	Default
`--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.	`false`
`--stage=all|prepare|build|merge`	Stage of incremental import. For incremental import into an existing database use `all` (which requires the database to be stopped). For semi-online incremental import run `prepare` (on a stopped database) followed by `build` (on a potentially running database) and finally `merge` (on a stopped database)",	`all`
`--threads=<num>`	(advanced) Max number of worker threads used by the importer. Defaults to the number of available processors reported by the JVM. There is a certain amount of minimum threads needed so for that reason there is no lower bound for this value. For optimal performance, this value shouldn't be greater than the number of available processors.	`10`
`--trim-strings[=<true/false>]`	Determines whether or not strings should be trimmed for whitespaces.	`false`
`--verbose`	Enable verbose output.	

CSV header format

The header file of each data source specifies how the data fields should be interpreted. You must use the same delimiter for the header file and the data files.

The header contains information for each field, with the format `<name>:<field_type>`. The `<name>` is used for properties and node IDs. In all other cases, the `<name>` part of the field is ignored.

When using incremental import, you will need to have node property uniqueness constraints in place for the property key and label combinations that form the primary key, or the uniquely identifiable nodes. For example, importing nodes with a `Person` label that are uniquely identified with a `uuid` property key, the format of the header should be `uuid:ID{label:Person}`.

This is also true when working with multiple groups. For example, you can use `uuid:ID(Person){label:Person}`, where the relationship CSV data can refer to different groups for its `:START_ID` and `:END_ID`, just like the full import method.

> - For more information on constraints, see Cypher Manual → Constraints.
> - For examples of creating property uniqueness constraints, see Cypher Manual → Node property uniqueness constraints.

Node files

Files containing node data can have an `ID` field, a `LABEL` field, and properties.

ID

Each node must have a unique ID if it is to be connected by any relationships created in the import. Neo4j uses the IDs to find the correct nodes when creating relationships. Note that the ID has to be unique across all nodes within the group, regardless of their labels. The unique ID is persisted in a property whose name is defined by the `<name>` part of the field definition `<name>:ID`. If no such property `name` is defined, the unique ID will be used for the import but not be available for reference later. If no ID is specified, the node will be imported, but it will not be connected to other nodes during the import. When a property `name` is provided, that property type can only be configured globally via the `--id-type` option and cannot be specified by `<field_type>` in the header field (as for properties).
From Neo4j v5.3, a node header can also contain multiple `ID` columns, where the relationship data references the composite value of all those columns. This also implies using `string` as `id-type`. For each `ID` column, you can specify to store its values as different node properties. However, the composite value cannot be stored as a node property.

LABEL

Read one or more labels from this field. Like array values, multiple labels are separated by `;`, or by the character specified with `--array-delimiter`.

Example 112. Define nodes files

You define the headers for movies in the *movies_header.csv* file. Movies have the properties `movieId`, `year`, and `title`. You also specify a field for labels.

```
movieId:ID,title,year:int,:LABEL
```

You define three movies in the *movies.csv* file. They contain all the properties defined in the header file. All the movies are given the label `Movie`. Two of them are also given the label `Sequel`.

```
tt0133093,"The Matrix",1999,Movie
tt0234215,"The Matrix Reloaded",2003,Movie;Sequel
tt0242653,"The Matrix Revolutions",2003,Movie;Sequel
```

Similarly, you also define three actors in the *actors_header.csv* and *actors.csv* files. They all have the properties `personId` and `name`, and the label `Actor`.

```
personId:ID,name,:LABEL
```

```
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
```

Relationship files

Files containing relationship data have three mandatory fields and can also have properties. The mandatory fields are:

TYPE

The relationship type to use for this relationship.

START_ID

The ID of the start node for this relationship.

END_ID

The ID of the end node for this relationship.

The `START_ID` and `END_ID` refer to the unique node ID defined in one of the node data sources, as explained in the previous section. None of these take a name, e.g. if `<name>:START_ID` or `<name>:END_ID` is defined, the `<name>` part will be ignored. Nor do they take a `<field_type>`, e.g. if `:START_ID:int` or `:END_ID:int` is defined, the `:int` part does not have any meaning in the context of type information.

Example 113. Define relationships files

In this example, you assume that the two node files from the previous example are used together with the following relationships file.

You define relationships between actors and movies in the files *roles_header.csv* and *roles.csv*. Each row connects a start node and an end node with a relationship of relationship type `ACTED_IN`. Notice how you use the unique identifiers `personId` and `movieId` from the nodes files above. The name of the character that the actor is playing in this movie is stored as a `role` property on the relationship.

```
:START_ID,role,:END_ID,:TYPE
```

```
keanu,"Neo",tt0133093,ACTED_IN
keanu,"Neo",tt0234215,ACTED_IN
keanu,"Neo",tt0242653,ACTED_IN
laurence,"Morpheus",tt0133093,ACTED_IN
laurence,"Morpheus",tt0234215,ACTED_IN
laurence,"Morpheus",tt0242653,ACTED_IN
carrieanne,"Trinity",tt0133093,ACTED_IN
carrieanne,"Trinity",tt0234215,ACTED_IN
carrieanne,"Trinity",tt0242653,ACTED_IN
```

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 114. 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 115. Property format for `point` data type

This example illustrates various ways of using the `point` data type in the import header and the data files.

You are going to import the name and location coordinates for cities. First, you define the header as:

```
:ID,name,location:point{crs:WGS-84}
```

You then define cities in the data file.

- The first city's location is defined using `latitude` and `longitude`, as expected when using the coordinate system defined in the header.
- The second city uses `x` and `y` instead. This would normally lead to a point using the coordinate reference system `cartesian`. Since the header defines `crs:WGS-84`, that coordinate reference system will be used.
- The third city overrides the coordinate reference system defined in the header and sets it explicitly to `WGS-84-3D`.

```
:ID,name,location:point{crs:WGS-84}
city01,"Malmö","{latitude:55.6121514, longitude:12.9950357}"
city02,"London","{y:51.507222, x:-0.1275}"
city03,"San Mateo","{latitude:37.554167, longitude:-122.313056, height: 100, crs:'WGS-84-3D'}"
```

Note that all point maps are within double quotation marks `"` in order to prevent the enclosed `,` character from being interpreted as a column separator. An alternative approach would be to use `--delimiter='\t'` and reformat the file with tab separators, in which case the `"` characters are not required.

```
:ID name    location:point{crs:WGS-84}
city01  Malmö   {latitude:55.6121514, longitude:12.9950357}
city02  London  {y:51.507222, x:-0.1275}
city03  San Mateo   {latitude:37.554167, longitude:-122.313056, height: 100, crs:'WGS-84-3D'}
```

Special considerations for temporal data types

The format for all temporal data types must be defined as described in Cypher Manual → Durations syntax. Two of the temporal types, *Time* and *DateTime*, take a time zone parameter that might be common between all or many of the values in the data file. It is therefore possible to specify a default time zone for *Time* and *DateTime* values in the header, for example: `time{timezone:+02:00}` and: `datetime{timezone:Europe/Stockholm}`. If no default time zone is specified, the default timezone is determined by the `db.temporal.timezone` configuration setting. The default time zone can be explicitly overridden in the values in the data file.

Example 116. Property format for temporal data types

This example illustrates various ways of using the datetime data type in the import header and the data files.

First, you define the header with two *DateTime* columns. The first one defines a time zone, but the second one does not:

```
:ID,date1:datetime{timezone:Europe/Stockholm},date2:datetime
```

You then define dates in the data file.

- The first row has two values that do not specify an explicit timezone. The value for date1 will use the Europe/Stockholm time zone that was specified for that field in the header. The value for date2 will use the configured default time zone of the database.
- In the second row, both date1 and date2 set the time zone explicitly to be Europe/Berlin. This overrides the header definition for date1, as well as the configured default time zone of the database.

```
1,2018-05-10T10:30,2018-05-10T12:30
2,2018-05-10T10:30[Europe/Berlin],2018-05-10T12:30[Europe/Berlin]
```

Using ID spaces

By default, the import tool assumes that node identifiers are unique across node files. In many cases, the ID is unique only across each entity file, for example, when your CSV files contain data extracted from a relational database and the ID field is pulled from the primary key column in the corresponding table. To handle this situation you define *ID spaces*. ID spaces are defined in the ID field of node files using the syntax ID(<ID space identifier>). To reference an ID of an ID space in a relationship file, you use the syntax START_ID(<ID space identifier>) and END_ID(<ID space identifier>).

Example 117. Define and use ID spaces

Define a `Movie-ID` ID space in the *movies_header.csv* file.

```
movieId:ID(Movie-ID),title,year:int,:LABEL
```

```
1,"The Matrix",1999,Movie
2,"The Matrix Reloaded",2003,Movie;Sequel
3,"The Matrix Revolutions",2003,Movie;Sequel
```

Define an `Actor-ID` ID space in the header of the *actors_header.csv* file.

```
personId:ID(Actor-ID),name,:LABEL
```

```
1,"Keanu Reeves",Actor
2,"Laurence Fishburne",Actor
3,"Carrie-Anne Moss",Actor
```

Now use the previously defined ID spaces when connecting the actors to movies.

```
:START_ID(Actor-ID),role,:END_ID(Movie-ID),:TYPE
```

```
1,"Neo",1,ACTED_IN
1,"Neo",2,ACTED_IN
1,"Neo",3,ACTED_IN
2,"Morpheus",1,ACTED_IN
2,"Morpheus",2,ACTED_IN
2,"Morpheus",3,ACTED_IN
3,"Trinity",1,ACTED_IN
3,"Trinity",2,ACTED_IN
3,"Trinity",3,ACTED_IN
```

Skipping columns

IGNORE

If there are fields in the data that you wish to ignore completely, this can be done using the `IGNORE` keyword in the header file. `IGNORE` must be prepended with a `:`.

Example 118. Skip a column

In this example, you are not interested in the data in the third column of the nodes file and wish to skip over it. Note that the `IGNORE` keyword is prepended by a `:`.

```
personId:ID,name,:IGNORE,:LABEL
```

```
keanu,"Keanu Reeves","male",Actor
laurence,"Laurence Fishburne","male",Actor
carrieanne,"Carrie-Anne Moss","female",Actor
```

If all your superfluous data is placed in columns located to the right of all the columns that you wish to import, you can instead use the command line option `--ignore-extra-columns`.

Import compressed files

The import tool can handle files compressed with `zip` or `gzip`. Each compressed file must contain a single file.

Example 119. 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 database import --nodes import/movies-header.csv,import/movies.csv.gz
--nodes import/actors-header.csv,import/actors.csv.zip --relationships import/roles-
header.csv,import/roles.csv.gz
```

Resuming a stopped or canceled import Enterprise edition

An import that is stopped or fails before completing can be resumed from a point closer to where it was stopped. An import can be resumed from the following points:

- Linking of relationships
- Post-processing

Unbind a Neo4j cluster server

You can use the `neo4j-admin server unbind` command to remove and archive the cluster state of a cluster server so that it can rebind to a cluster.

Syntax

The `neo4j-admin server unbind` command has the following syntax:

```
neo4j-admin server unbind [-h]
                          [--expand-commands]
                          [--verbose]
                          [--archive-cluster-state[=true|false]]
                          [--additional-config=<file>]
                          [--archive-path=<path>]
```

Options

The `neo4j-admin server unbind` command has the following options:

Option	Default	Description
`-h`, `--help`		Show this help message and exit.

Option	Default	Description
`--expand-commands`		Allow command expansion in config value evaluation.
`--verbose`		Enable verbose output.
`--archive-cluster-state`	`true` or `false` (Default: `false`)	Enable or disable the cluster state archiving.
`--additional-config`	`<file>`	Configuration file with additional configuration.
`--archive-path`	`<path`	Destination (file or folder) of the cluster state archive.

Limitations

The Neo4j server must be shut down before running the `neo4j-admin server unbind` command.

Usage

You can use the `neo4j-admin server unbind` command to remove the cluster state of a cluster server, turn a cluster server into a standalone server, or remove and archive the cluster state of a cluster server.

Remove the cluster state of a server

To remove the cluster state of a server, run the `neo4j-admin server unbind` command from the *<neo4j-home>* folder of that server. When restarted, an unbound server rejoins the cluster as a new server and has to be enabled using the `ENABLE SERVER` command.

Note that the `unbind` command preserves all database stores on the server. When the unbound server is restarted and enabled, it is seen as an entirely new server. Therefore, there is no guarantee that the allocator will pick this server to host the same databases. This may lead to having orphaned database stores on this server.

Turn a cluster member into a standalone server

To start the Neo4j server in single (standalone) mode after unbinding it from the cluster, verify that `initial.server.mode_constraint` is set to `NONE` in The neo4j.conf file.

Archive cluster state

To archive the cluster state, from the *<neo4j-home>* folder, run the `neo4j-admin server unbind` command with the arguments `--archive-cluster-state=true` and `--archive-path=<destination-folder>`:

```
bin/neo4j-admin server unbind --archive-path=/path/to/archive-folder --archive-cluster-state=true
```

The default resultant file is named:

```
unbound_cluster_state.<YYYYMMDDHH24MM>.zip
```

Upload to Neo4j Aura

The `neo4j-admin database upload` command uploads a local Neo4j database dump into a Neo4j Aura instance.
The following table shows the compatibility between the dump version that you want to upload and the version of the Neo4j Aura instance.

Dump version	Aura version
v5.x	v5.latest
v4.4	v4 and v5.latest
v4.3	v4 and v5.latest

This operation is secured and TLS encrypted end to end.

Syntax

The `neo4j-admin database upload` command has the following syntax:

```
neo4j-admin database upload [-h]
                            [--expand-commands]
                            [--verbose]
                            [--overwrite-destination[=true|false]]
                            --from-path=<path>
                            [--to=<destination>]
                            [--to-password=<password>]
                            --to-uri=<uri>
                            [--to-user=<username>]
                            <database>
```

Options

The `neo4j-admin database upload` command has the following options:

Table 101. Options

Option	Possible values	Description
`-h, --help`		Show this help message and exit.
`--expand-commands`		Allow command expansion in config value evaluation.
`--verbose`		Enable verbose output.
`--overwrite-destination`	`true` or `false` (Default: `false`)	Overwrite the data in the target database.
`--from-path`	`<path>`	Path to a directory containing a database dump to upload.
`--to`	`<destination>` (Default: `aura`)	The destination for the upload.
`--to-password`	`<password>`	The password for the target Aura instance where you will upload this database. If you do not provide a password, you will be prompted to provide one. Alternatively, the `NEO4J_PASSWORD` environment variable can be used.
`--to-uri`	`<uri>`	The Bolt URI of the target Aura database. For example, `neo4j+s://your-aura-instance-id.databases.neo4j.io`.
`--to-user`	`<username>`	The username of the target Aura instance where you will upload this database. If you do not provide a username, you will be prompted to provide one. Alternatively, you can use the `NEO4J_USERNAME` environment variable.
`<database>`		Name of the database dump that should be uploaded.

Limitations

- A Neo4j Aura instance must already be provisioned and running.

Output

If the `upload` function completes successfully, it exits with the following log line:

```
“Your data was successfully pushed to Aura and is now running”.
```

If the `upload` function encounters an error at any point, you will be provided with instructions on how to try again or to contact Neo4j Aura support.

Additionally, you can use the `--verbose` option to enable verbose output.

Example

The following examples show how to use the `neo4j-admin database upload` command to upload a database dump to a Neo4j Aura instance. You need your Aura instance URI (`neo4j+s://your-aura-instance-id.databases.neo4j.io`), as can be seen in the Aura console, and your Aura instance password.

You should use the `--overwrite-destination=true` option to overwrite the target database. Otherwise, the command aborts and throws an error.

```
$neo4j-home> bin/neo4j-admin database upload <database> --from-path=<path-to-directory-with-database-dump>
--to-uri=<neo4j+s://your-aura-instance-id.databases.neo4j.io> --overwrite-destination=true
Neo4j cloud database user name: neo4j
Neo4j cloud database password:
Upload
....................  10%
....................  20%
....................  30%
....................  40%
....................  50%
....................  60%
....................  70%
....................  80%
....................  90%
.................... 100%
We have received your export and it is currently being loaded into your Aura instance.
You can wait here, or abort this command and head over to the console to be notified of when your database
is running.
Import progress
....................  10%
....................  20%
....................  30%
....................  40%
....................  50%
....................  60%
....................  70%
....................  80%
....................  90%
.................... 100%
Your data was successfully pushed to Aura and is now running.
```

Migrate a database

You can use the `neo4j-admin database migrate` command to migrate a Neo4j database from one store format to another or to a later `MAJOR` version of the same format.

A store format defines how the data of a database is stored on the file system.

As of Neo4j 5, the store format of a database is versioned with the `MAJOR.MINOR` scheme, independent of Neo4j versioning. An upgrade to the latest `MINOR` format version is an automatic operation performed on database startup. A migration to a higher `MAJOR` format version or another format is a manual action performed with the `migrate` command.

The store format for new databases can be set with the `db.format` configuration setting.

Syntax

The `neo4j-admin database migrate` has the following syntax:

```
neo4j-admin database migrate [-h]
                             [--expand-commands]
                             [--verbose]
                             [--additional-config=<file>]
                             [--pagecache=<size>]
                             [--to-format=standard|high_limit|aligned]
                             [--force-btree-indexes-to-range]
                             <database>
```

The `neo4j-admin database migrate` command is run only on a stopped database.

Options

The following table describes the available options:

Option	Possible values	Description
`-h`, `--help`		Show this help message and exit.
`--expand-commands`		Allow command expansion in config value evaluation.
`--verbose`		Enable verbose output.
`--additional-config`	`<file>`	Configuration file to supply additional configuration in.
`--pagecache`	`<size>`	The size of the page cache to use for the migration process. The general rule is that values up to the size of the database proportionally increase performance.
`--to-format`	`standard`, `high_limit`, and `aligned`	If the format is specified, the target database is migrated to the latest known combination of `MAJOR` and `MINOR` versions of the specified format. If not specified, the tool migrates the target database to the latest known combination of `MAJOR` and `MINOR` versions of the current format.

Option	Possible values	Description
`--force-btree-indexes-to-range`		Forces the removal of BTREE indexes and replaces them with RANGE indexes. BTREE indexes are not supported in Neo4j v5 and the `neo4j-admin database migrate` command will remove all BTREE indexes and BTREE index-backed constraints. It is recommended to create valid replacements before migrating the database. For more information, see [Upgrade and Migration Guide → Prepare indexes.
`<database>`		The database to migrate.

Example

The following example migrates the `movies` database to the latest known combination of `MAJOR` and `MINOR` versions of the `high_limit` format:

```
neo4j-admin database migrate --to-format=high_limit movies
```

Example output

```
2022-11-10 12:55:35.542+0000 INFO  [o.n.c.d.MigrateStoreCommand] Starting migration for database 'movies'
2022-11-10 12:55:36.061+0000 INFO  [o.n.k.i.s.StoreMigrator] 'record-aligned-1.1' has been identified as
the current version of the store
2022-11-10 12:55:36.061+0000 INFO  [o.n.k.i.s.StoreMigrator] 'record-high_limit-1.1' has been identified
as the target version of the store migration
2022-11-10 12:55:36.065+0000 INFO  [o.n.k.i.s.StoreMigrator] Starting migration of database
2022-11-10 12:55:36.154+0000 INFO  [o.n.k.i.s.StoreMigrator] Migrating Store files (1/7):
2022-11-10 12:55:36.828+0000 INFO  [o.n.k.i.s.f.RecordFormatSelector] Selected
RecordFormat:HighLimitV5_0[high_limit-1.1] record format from store
$NEO4J_HOME/data/databases/movies/migrate
2022-11-10 12:55:36.828+0000 INFO  [o.n.k.i.s.f.RecordFormatSelector] Selected format from the store
files: RecordFormat:HighLimitV5_0[high_limit-1.1]
2022-11-10 12:55:37.020+0000 INFO  [o.n.i.b.ImportLogic] Import starting
2022-11-10 12:55:37.924+0000 INFO  [o.n.k.i.s.StoreMigrator]   10% completed
2022-11-10 12:55:37.924+0000 INFO  [o.n.k.i.s.StoreMigrator]   20% completed
2022-11-10 12:55:37.924+0000 INFO  [o.n.k.i.s.StoreMigrator]   30% completed
2022-11-10 12:55:37.924+0000 INFO  [o.n.k.i.s.StoreMigrator]   40% completed
2022-11-10 12:55:37.924+0000 INFO  [o.n.k.i.s.StoreMigrator]   50% completed
2022-11-10 12:55:37.924+0000 INFO  [o.n.k.i.s.StoreMigrator]   60% completed
2022-11-10 12:55:37.924+0000 INFO  [o.n.k.i.s.StoreMigrator]   70% completed
2022-11-10 12:55:37.924+0000 INFO  [o.n.k.i.s.StoreMigrator]   80% completed
2022-11-10 12:55:37.924+0000 INFO  [o.n.k.i.s.StoreMigrator]   90% completed
2022-11-10 12:55:37.924+0000 INFO  [o.n.k.i.s.StoreMigrator]   100% completed
2022-11-10 12:55:37.925+0000 INFO  [o.n.i.b.ImportLogic] Import completed successfully, took 903ms.
Imported:
  171 nodes
  253 relationships
  564 properties
2022-11-10 12:55:38.515+0000 INFO  [o.n.k.i.s.StoreMigrator] Migrating text-1.0 (2/7):
2022-11-10 12:55:38.515+0000 INFO  [o.n.k.i.s.StoreMigrator]   10% completed
2022-11-10 12:55:38.515+0000 INFO  [o.n.k.i.s.StoreMigrator]   20% completed
2022-11-10 12:55:38.515+0000 INFO  [o.n.k.i.s.StoreMigrator]   30% completed
2022-11-10 12:55:38.515+0000 INFO  [o.n.k.i.s.StoreMigrator]   40% completed
2022-11-10 12:55:38.515+0000 INFO  [o.n.k.i.s.StoreMigrator]   50% completed
2022-11-10 12:55:38.515+0000 INFO  [o.n.k.i.s.StoreMigrator]   60% completed
2022-11-10 12:55:38.515+0000 INFO  [o.n.k.i.s.StoreMigrator]   70% completed
2022-11-10 12:55:38.515+0000 INFO  [o.n.k.i.s.StoreMigrator]   80% completed
2022-11-10 12:55:38.516+0000 INFO  [o.n.k.i.s.StoreMigrator]   90% completed
2022-11-10 12:55:38.516+0000 INFO  [o.n.k.i.s.StoreMigrator]   100% completed
2022-11-10 12:55:38.516+0000 INFO  [o.n.k.i.s.StoreMigrator] Migrating range-1.0 indexes (3/7):
```

```
2022-11-10 12:55:38.516+0000 INFO  [o.n.k.i.s.StoreMigrator]   10% completed
2022-11-10 12:55:38.516+0000 INFO  [o.n.k.i.s.StoreMigrator]   20% completed
2022-11-10 12:55:38.516+0000 INFO  [o.n.k.i.s.StoreMigrator]   30% completed
2022-11-10 12:55:38.516+0000 INFO  [o.n.k.i.s.StoreMigrator]   40% completed
2022-11-10 12:55:38.516+0000 INFO  [o.n.k.i.s.StoreMigrator]   50% completed
2022-11-10 12:55:38.516+0000 INFO  [o.n.k.i.s.StoreMigrator]   60% completed
2022-11-10 12:55:38.516+0000 INFO  [o.n.k.i.s.StoreMigrator]   70% completed
2022-11-10 12:55:38.516+0000 INFO  [o.n.k.i.s.StoreMigrator]   80% completed
2022-11-10 12:55:38.516+0000 INFO  [o.n.k.i.s.StoreMigrator]   90% completed
2022-11-10 12:55:38.516+0000 INFO  [o.n.k.i.s.StoreMigrator]   100% completed
2022-11-10 12:55:38.517+0000 INFO  [o.n.k.i.s.StoreMigrator] Migrating Fulltext indexes (4/7):
2022-11-10 12:55:38.517+0000 INFO  [o.n.k.i.s.StoreMigrator]   10% completed
2022-11-10 12:55:38.517+0000 INFO  [o.n.k.i.s.StoreMigrator]   20% completed
2022-11-10 12:55:38.517+0000 INFO  [o.n.k.i.s.StoreMigrator]   30% completed
2022-11-10 12:55:38.517+0000 INFO  [o.n.k.i.s.StoreMigrator]   40% completed
2022-11-10 12:55:38.517+0000 INFO  [o.n.k.i.s.StoreMigrator]   50% completed
2022-11-10 12:55:38.517+0000 INFO  [o.n.k.i.s.StoreMigrator]   60% completed
2022-11-10 12:55:38.517+0000 INFO  [o.n.k.i.s.StoreMigrator]   70% completed
2022-11-10 12:55:38.517+0000 INFO  [o.n.k.i.s.StoreMigrator]   80% completed
2022-11-10 12:55:38.517+0000 INFO  [o.n.k.i.s.StoreMigrator]   90% completed
2022-11-10 12:55:38.517+0000 INFO  [o.n.k.i.s.StoreMigrator]   100% completed
2022-11-10 12:55:38.517+0000 INFO  [o.n.k.i.s.StoreMigrator] Migrating point-1.0 indexes (5/7):
2022-11-10 12:55:38.517+0000 INFO  [o.n.k.i.s.StoreMigrator]   10% completed
2022-11-10 12:55:38.518+0000 INFO  [o.n.k.i.s.StoreMigrator]   20% completed
2022-11-10 12:55:38.518+0000 INFO  [o.n.k.i.s.StoreMigrator]   30% completed
2022-11-10 12:55:38.518+0000 INFO  [o.n.k.i.s.StoreMigrator]   40% completed
2022-11-10 12:55:38.518+0000 INFO  [o.n.k.i.s.StoreMigrator]   50% completed
2022-11-10 12:55:38.518+0000 INFO  [o.n.k.i.s.StoreMigrator]   60% completed
2022-11-10 12:55:38.518+0000 INFO  [o.n.k.i.s.StoreMigrator]   70% completed
2022-11-10 12:55:38.518+0000 INFO  [o.n.k.i.s.StoreMigrator]   80% completed
2022-11-10 12:55:38.518+0000 INFO  [o.n.k.i.s.StoreMigrator]   90% completed
2022-11-10 12:55:38.518+0000 INFO  [o.n.k.i.s.StoreMigrator]   100% completed
2022-11-10 12:55:38.518+0000 INFO  [o.n.k.i.s.StoreMigrator] Migrating Token indexes (6/7):
2022-11-10 12:55:38.518+0000 INFO  [o.n.k.i.s.StoreMigrator]   10% completed
2022-11-10 12:55:38.519+0000 INFO  [o.n.k.i.s.StoreMigrator]   20% completed
2022-11-10 12:55:38.519+0000 INFO  [o.n.k.i.s.StoreMigrator]   30% completed
2022-11-10 12:55:38.519+0000 INFO  [o.n.k.i.s.StoreMigrator]   40% completed
2022-11-10 12:55:38.519+0000 INFO  [o.n.k.i.s.StoreMigrator]   50% completed
2022-11-10 12:55:38.519+0000 INFO  [o.n.k.i.s.StoreMigrator]   60% completed
2022-11-10 12:55:38.519+0000 INFO  [o.n.k.i.s.StoreMigrator]   70% completed
2022-11-10 12:55:38.519+0000 INFO  [o.n.k.i.s.StoreMigrator]   80% completed
2022-11-10 12:55:38.519+0000 INFO  [o.n.k.i.s.StoreMigrator]   90% completed
2022-11-10 12:55:38.519+0000 INFO  [o.n.k.i.s.StoreMigrator]   100% completed
2022-11-10 12:55:38.519+0000 INFO  [o.n.k.i.s.StoreMigrator] Migrating text-2.0 (7/7):
2022-11-10 12:55:38.519+0000 INFO  [o.n.k.i.s.StoreMigrator]   10% completed
2022-11-10 12:55:38.519+0000 INFO  [o.n.k.i.s.StoreMigrator]   20% completed
2022-11-10 12:55:38.519+0000 INFO  [o.n.k.i.s.StoreMigrator]   30% completed
2022-11-10 12:55:38.520+0000 INFO  [o.n.k.i.s.StoreMigrator]   40% completed
2022-11-10 12:55:38.520+0000 INFO  [o.n.k.i.s.StoreMigrator]   50% completed
2022-11-10 12:55:38.520+0000 INFO  [o.n.k.i.s.StoreMigrator]   60% completed
2022-11-10 12:55:38.520+0000 INFO  [o.n.k.i.s.StoreMigrator]   70% completed
2022-11-10 12:55:38.520+0000 INFO  [o.n.k.i.s.StoreMigrator]   80% completed
2022-11-10 12:55:38.520+0000 INFO  [o.n.k.i.s.StoreMigrator]   90% completed
2022-11-10 12:55:38.520+0000 INFO  [o.n.k.i.s.StoreMigrator]   100% completed
2022-11-10 12:55:38.628+0000 INFO  [o.n.k.i.s.StoreMigrator] Starting transaction logs migration.
2022-11-10 12:55:38.660+0000 INFO  [o.n.k.i.s.StoreMigrator] Transaction logs migration completed.
2022-11-10 12:55:38.696+0000 INFO  [o.n.k.i.s.StoreMigrator] Successfully finished migration of database,
took 2s 631ms
2022-11-10 12:55:38.698+0000 INFO  [o.n.c.d.MigrateStoreCommand] Database migration completed successfully
```

Migrate the Neo4j configuration file

You can use the `migrate-configuration` command to migrate a legacy Neo4j configuration file to the current format. The new version will be written in a target configuration directory. The default location for both the source and target configuration directory is the configuration directory specified by `NEO_CONF` or the default configuration directory for this installation. If the source and target directories are the same, the original configuration files will be renamed. A configuration provided using `--additional-config` option will not be migrated.

Why use the command?

- Configuration migration is a purely mechanical process, and using the explicit migration with the `migrate-configuration` command allows you to inspect and customize the output.
- The command output provides valuable insight into the migration process, including notification about settings that could not have been meaningfully migrated, for instance, because the concept or behavior no longer exists in the new `MAJOR` version of the DBMS.

Syntax

The `migrate-configuration` command has the following syntax:

```
neo4j-admin server migrate-configuration [--expand-commands]
                                         [--verbose]
                                         [--from-path=<path>]
                                         [--to-path=<path>]
                                         [-h]
```

Options

The `neo4j-admin migrate-configuration` command has the following options:

Table 102. Options

Option	Default	Description
`--expand-commands`		Allow command expansion in config value evaluation.
`--verbose`		Enable verbose output.
`--from-path`	`<path>`	Path to the configuration directory used as a source for the migration.
`--to-path`	`<path>`	Path to a directory where the migrated configuration files should be written.

Example

The following example shows how to migrate a legacy configuration file to the current format:

```
$ neo4j-admin server migrate-configuration --from-path=/path/to/legacy/neo4j-enterprise-4.4.10/conf/ --to
-path=/path/to/new/neo4j-enterprise-5.1.0/conf/
```

Example output

```
Keeping original user-logs.xml file at: /neo4j-enterprise-5.1.0/conf/user-logs.xml.old
User logging configuration xml file generated: /neo4j-enterprise-5.1.0/conf/user-logs.xml
Keeping original server-logs.xml file at: /neo4j-enterprise-5.1.0/conf/server-logs.xml.old
Server logging configuration xml file generated: /neo4j-enterprise-5.1.0/conf/server-logs.xml
dbms.directories.import=import MIGRATED -> server.directories.import=import
dbms.tx_state.memory_allocation=ON_HEAP MIGRATED -> db.tx_state.memory_allocation=ON_HEAP
dbms.connector.bolt.enabled=true MIGRATED -> server.bolt.enabled=true
dbms.connector.http.enabled=true MIGRATED -> server.http.enabled=true
dbms.connector.https.enabled=false MIGRATED -> server.https.enabled=false
metrics.csv.rotation.compression=zip MIGRATED -> server.metrics.csv.rotation.compression=zip
dbms.jvm.additional=-XX:+UseG1GC MIGRATED -> server.jvm.additional=-XX:+UseG1GC
dbms.jvm.additional=-XX:-OmitStackTraceInFastThrow MIGRATED -> server.jvm.additional=-XX:-
OmitStackTraceInFastThrow
dbms.jvm.additional=-XX:+AlwaysPreTouch MIGRATED -> server.jvm.additional=-XX:+AlwaysPreTouch
dbms.jvm.additional=-XX:+UnlockExperimentalVMOptions MIGRATED -> server.jvm.additional=-
XX:+UnlockExperimentalVMOptions
dbms.jvm.additional=-XX:+TrustFinalNonStaticFields MIGRATED -> server.jvm.additional=-
XX:+TrustFinalNonStaticFields
dbms.jvm.additional=-XX:+DisableExplicitGC MIGRATED -> server.jvm.additional=-XX:+DisableExplicitGC
dbms.jvm.additional=-XX:MaxInlineLevel=15 MIGRATED -> server.jvm.additional=-XX:MaxInlineLevel=15
dbms.jvm.additional=-XX:-UseBiasedLocking MIGRATED -> server.jvm.additional=-XX:-UseBiasedLocking
dbms.jvm.additional=-Djdk.nio.maxCachedBufferSize=262144 MIGRATED -> server.jvm.additional=-
Djdk.nio.maxCachedBufferSize=262144
dbms.jvm.additional=-Dio.netty.tryReflectionSetAccessible=true MIGRATED -> server.jvm.additional=-
Dio.netty.tryReflectionSetAccessible=true
dbms.jvm.additional=-Djdk.tls.ephemeralDHKeySize=2048 MIGRATED -> server.jvm.additional=-
Djdk.tls.ephemeralDHKeySize=2048
dbms.jvm.additional=-Djdk.tls.rejectClientInitiatedRenegotiation=true MIGRATED -> server.jvm.additional=-
Djdk.tls.rejectClientInitiatedRenegotiation=true
dbms.jvm.additional=-XX:FlightRecorderOptions=stackdepth=256 MIGRATED -> server.jvm.additional=-
XX:FlightRecorderOptions=stackdepth=256
dbms.jvm.additional=-XX:+UnlockDiagnosticVMOptions MIGRATED -> server.jvm.additional=-
XX:+UnlockDiagnosticVMOptions
dbms.jvm.additional=-XX:+DebugNonSafepoints MIGRATED -> server.jvm.additional=-XX:+DebugNonSafepoints
dbms.jvm.additional=-Dlog4j2.disable.jmx=true MIGRATED -> server.jvm.additional=-Dlog4j2.disable.jmx=true
dbms.windows_service_name=neo4j MIGRATED -> server.windows_service_name=neo4j
Keeping original configuration file at: /neo4j-enterprise-5.1.0/conf/neo4j.conf.old
```

Cypher Shell

About Cypher Shell CLI

Cypher Shell is a command-line tool that comes with the Neo4j distribution. 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.

Syntax

The Cypher Shell CLI is located in the `bin` directory if installed as part of the product.

Syntax:

```
cypher-shell  [-h, --help]
              [-a ADDRESS, --address ADDRESS, --uri ADDRESS]
              [-u USERNAME, --username USERNAME]
              [-p PASSWORD, --password PASSWORD]
              [--encryption {true,false,default}]
              [-d DATABASE, --database DATABASE]
              [--impersonate IMPERSONATE]
              [--format {auto,verbose,plain}]
              [-P PARAM, --param PARAM]
              [--debug]
              [--non-interactive]
              [--sample-rows SAMPLE-ROWS]
              [--wrap {true,false}]
              [-v, --version]
              [--driver-version]
              [-f FILE, --file FILE]
              [--change-password]
              [--fail-fast]
              [--fail-at-end]
              [--log [LOG-FILE]]
              [cypher]
```

Arguments:

Argument	Type	Description	Default value

`-h`
 	
`--help`
	Optional argument	Show help message and exit.	

`-a ADDRESS`
 	
`--address ADDRESS`
 	
`--uri ADDRESS`
	Connection argument	Address and port to connect to. It can also be specified by the environment variable `NEO4J_ADDRESS` or `NEO4J_URI`.	`neo4j://localhost:7687`

`-u USERNAME`
 	
`--username USERNAME`
	Connection argument	Username to connect as. It can also be specified by the environment variable `NEO4J_USERNAME`.	

`-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.

Argument	Type	Description	Default value

`-d DATABASE`
`--database DATABASE`
	Connection argument	Database to connect to. It can also be specified by the environment variable `NEO4J_DATABASE`.	

`--format {auto,verbose,plain}`	Optional argument	Desired output format.	`auto` (default): displays results in tabular format if you use the shell interactively and with minimal formatting if you use it for scripting. `verbose`: displays results in tabular format and prints statistics. `plain`: displays data with minimal formatting.
`--impersonate USER-TO-IMPERSONATE`	Connection argument	Impersonate the specified user.	No impersonation

`--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.	

`--debug`	Optional argument	Print additional debug information.	`false`
`--non-interactive`	Optional argument	Force non-interactive mode; only useful if auto-detection fails (e.g. Windows).	`false`
`--sample-rows SAMPLE-ROWS`	Optional argument	Number of rows sampled to compute table widths (only for format=`VERBOSE`).	`1000`
`--wrap {true,false}`	Optional argument	Wrap table column values if column is too narrow (only for format=`VERBOSE`).	`true`

Argument	Type	Description	Default value
`-v` `--version`	Optional argument	Print version of cypher-shell and exit.	`false`
`--driver-version`	Optional argument	Print version of the Neo4j Driver used and exit.	`false`

`-f FILE`
`--file FILE`
	Optional argument	Pass a file with cypher statements to be executed. After the statements have been executed cypher-shell shuts down.	

`--change-password`	Optional argument	Change Neo4j user password and exit	`false`
`--fail-fast`	Optional argument	Exit and report failure on first error when reading from file.	This is the default behavior.
`--fail-at-end`	Optional argument	Exit and report failures at end of input when reading from file.	
`--log [LOG-FILE]`	Optional argument	Enables logging to the specified file, or `stderr` if the file is omitted.	
`cypher`	Positional argument	An optional string of cypher to execute and then exit.	

Running Cypher Shell within the Neo4j distribution

You can connect to a live Neo4j DBMS by running `cypher-shell` and passing in a username and a password argument:

```
bin/cypher-shell -u neo4j -p <password>
```

The output is the following:

```
Connected to Neo4j at neo4j://localhost:7687 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
```

Running Cypher Shell from a different server

You can also install the Cypher Shell tool on a different server (without Neo4j) and connect to a Neo4j DBMS. Cypher Shell requires a JDK and Java 11.

> DEB/RPM distributions both install OpenJDK if it is not already installed. The *cypher-shell* files are available in the same DEB/RPM Linux repositories as Neo4j.
>
> The TAR distribution contains only the *cypher-shell* files, so you must install the JDK manually.

1. Download Cypher Shell from Neo4j Download Center.
2. Connect to a Neo4j DBMS by running the `cypher-shell` command providing the Neo4j address, a username, and a password:

   ```
   cypher-shell/cypher-shell -a neo4j://IP-address:7687 -u neo4j -p <password>
   ```

   The output is the following:

   ```
   Connected to Neo4j at neo4j://IP-address:7687 as user neo4j.
   Type :help for a list of available commands or :exit to exit the shell.
   Note that Cypher queries must end with a semicolon.
   ```

Available commands

Once in the interactive shell, run the following command to display all available commands:

Example 120. Running `help`

```
:help
```

The output is the following:

```
Available commands:
  *:begin*       Open a transaction
  *:commit*      Commit the currently open transaction
  *:connect*     Connects to a database
  *:disconnect*  Disconnects from database
  *:exit*        Exit the logger
  *:help*        Show this help message
  *:history*     Statement history
  *:impersonate* Impersonate user
  *:param*       Set the value of a query parameter
  *:params*      Print all query parameter values
  *:rollback*    Rollback the currently open transaction
  *:source*      Executes Cypher statements from a file
  *:use*         Set the active database

For help on a specific command type:
    :help *command*

Keyboard shortcuts:
    Up and down arrows to access statement history.
    Tab for autocompletion of commands, hit twice to select suggestion from list using arrow keys.
```

Running Cypher statements

You can run Cypher statements in the following ways:

- Typing Cypher statements directly into the interactive shell.
- Running Cypher statements from a file with the interactive shell.
- Running Cypher statements from a file as a `cypher-shell` argument.

The examples in this section use the `MATCH (n) RETURN n LIMIT 5` Cypher statement and will return 5 nodes from the database.

Example 121. Typing a Cypher statement directly into the interactive shell

```
MATCH (n) RETURN n LIMIT 5;
```

The following two examples assume a file exists in the same folder you run the `cypher-shell` command from called `example.cypher` with the following contents:

```
MATCH (n) RETURN n LIMIT 5;
```

Example 122. Running Cypher statements from a file with the interactive shell

You can use the `:source` command followed by the file name to run the Cypher statements in that file when in the Cypher interactive shell:

```
:source example.cypher
```

Example 123. Running Cypher statements from a file as a `cypher-shell` argument.

You can pass a file containing Cypher statements as an argument when running `cypher-shell`.

The examples here use the `--format plain` flag for a simple output.

Using `cat` (UNIX)

```
cat example.cypher | bin/cypher-shell -u neo4j -p <password> --format plain
```

Using `type` (Windows)

```
type example.cypher | bin/cypher-shell.bat -u neo4j -p <password> --format plain
```

Query parameters

Cypher Shell CLI supports querying based on parameters. This is often used while scripting.

Example 124. Use parameters within Cypher Shell

1. Set the parameter `thisAlias` to `Robin` using the `:param` keyword:

   ```
   :param thisAlias => 'Robin'
   ```

2. Check the parameter using the `:params` keyword:

   ```
   :params
   ```

   ```
   :param thisAlias => 'Robin'
   ```

3. Now use the parameter `thisAlias` in a Cypher query:

   ```
   CREATE (:Person {name : 'Dick Grayson', alias : $thisAlias });
   ```

   ```
   Added 1 nodes, Set 2 properties, Added 1 labels
   ```

4. Verify the result:

   ```
   MATCH (n) RETURN n;
   ```

   ```
   +--------------------------------------------------------------------+
   | n                                                                  |
   +--------------------------------------------------------------------+
   | (:Person {name: "Bruce Wayne", alias: "Batman"})                   |
   | (:Person {name: "Selina Kyle", alias: ["Catwoman", "The Cat"]})    |
   | (:Person {name: "Dick Grayson", alias: "Robin"})                   |
   +--------------------------------------------------------------------+
   3 rows available after 2 ms, consumed after another 2 ms
   ```

Transactions

Cypher Shell supports explicit transactions. Transaction states are controlled using the keywords `:begin`, `:commit`, and `:rollback`.

Both explicit and implicit transactions run from Cypher Shell will have default transaction metadata attached that follows the convention (see Attach metadata to a transaction).

Example 125. Use fine-grained transaction control

The example uses the dataset from the built-in Neo4j Browser guide, called MovieGraph. For more information, see the Neo4j Browser documentation.

1. Run a query that shows there is only one person in the database, who is born in 1964.

```
MATCH (n:Person) WHERE n.born=1964 RETURN n.name AS name;
```

```
+-----------------+
| name            |
+-----------------+
| "Keanu Reeves" |
+-----------------+

1 row
ready to start consuming query after 9 ms, results consumed after another 0 ms
```

2. Start a transaction and create another person born in the same year:

```
:begin
neo4j# CREATE (:Person {name : 'Edward Mygma', born:1964});
```

```
0 rows
ready to start consuming query after 38 ms, results consumed after another 0 ms
Added 1 nodes, Set 2 properties, Added 1 labels
```

3. If you open a second Cypher Shell session and run the query from step 1, you will notice no changes from the latest `CREATE` statement.

```
MATCH (n:Person) WHERE n.born=1964 RETURN n.name AS name;
```

```
+-----------------+
| name            |
+-----------------+
| "Keanu Reeves" |
+-----------------+

1 row
ready to start consuming query after 9 ms, results consumed after another 0 ms
```

4. Go back to the first session and commit the transaction.

```
neo4j# :commit
```

5. Now, if you run the query from step 1, you will see that Edward Mygma has been added to the database.

```
MATCH (n:Person) WHERE n.born=1964 RETURN n.name AS name;
```

```
+-----------------+
| name            |
+-----------------+
| "Keanu Reeves" |
| "Edward Mygma" |
+-----------------+

2 rows
ready to start consuming query after 1 ms, results consumed after another 1 ms
```

Procedures

Cypher Shell supports running any procedures for which the current user is authorized.

Example 126. Call the `dbms.showCurrentUser` *procedure*

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

```
+------------------------------+
| username | roles     | flags |
+------------------------------+
| "neo4j"  | ["admin"] | []    |
+------------------------------+

1 row available after 66 ms, consumed after another 2 ms
```

Supported operating systems

You can use the Cypher Shell CLI via `cmd` on Windows systems, and `bash` on Unix systems.

Other shells may work as intended, but there is no test coverage to guarantee compatibility.

Keyboard shortcuts

The following keyboard commands are available in interactive mode.

Key	Operation
↑ and ↓ (arrow keys)	Access statement history.
▯ (tab)	Autocompletion of commands and Cypher syntax. Suggestions for Cypher syntax is not complete.

Appendix A: Reference

This appendix contains the following topics:

- All configuration settings
- Dynamic configuration settings

* Procedures

Configuration settings

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

All settings

* browser.allow_outgoing_connections: Enterprise edition Configure the policy for outgoing Neo4j Browser connections.
* browser.credential_timeout: Enterprise edition 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: Enterprise edition Configure the Neo4j Browser to store or not store user credentials.
* browser.retain_editor_history: Enterprise edition Configure the Neo4j Browser to store or not store user editor history.
* client.allow_telemetry: Configure client applications such as Browser and Bloom to send Product Analytics data.
* db.checkpoint: Configures the general policy for when check-points should occur.
* db.checkpoint.interval.time: Configures the time interval between check-points.
* db.checkpoint.interval.tx: Configures the transaction interval between check-points.
* db.checkpoint.interval.volume: Configures the volume of transaction logs between check-points.
* db.checkpoint.iops.limit: Limit the number of IOs the background checkpoint process will consume per second.
* db.cluster.catchup.pull_interval: Enterprise edition Interval of pulling updates from cores.
* db.cluster.raft.apply.buffer.max_bytes: Enterprise edition The maximum number of bytes in the apply buffer.
* db.cluster.raft.apply.buffer.max_entries: Enterprise edition The maximum number of entries in the raft log entry prefetch buffer.
* db.cluster.raft.in_queue.batch.max_bytes: Enterprise edition Largest batch processed by RAFT in bytes.
* db.cluster.raft.in_queue.max_bytes: Enterprise edition Maximum number of bytes in the RAFT in-queue.
* db.cluster.raft.leader_transfer.priority_group: Enterprise edition The name of a server_group whose members should be prioritized as leaders.
* db.cluster.raft.log.prune_strategy: Enterprise edition RAFT log pruning strategy that determines which

logs are to be pruned.

- db.cluster.raft.log_shipping.buffer.max_bytes: Enterprise edition The maximum number of bytes in the in-flight cache.
- db.cluster.raft.log_shipping.buffer.max_entries: Enterprise edition The maximum number of entries in the in-flight cache.
- db.filewatcher.enabled: Allows the enabling or disabling of the file watcher service.
- db.format: Database format.
- db.import.csv.buffer_size: The size of the internal buffer in bytes used by `LOAD CSV`.
- db.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`.
- db.index.fulltext.default_analyzer: The name of the analyzer that the fulltext indexes should use by default.
- db.index.fulltext.eventually_consistent: Whether or not fulltext indexes should be eventually consistent by default or not.
- db.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.
- db.index_sampling.background_enabled: Enable or disable background index sampling.
- db.index_sampling.sample_size_limit: Index sampling chunk size limit.
- db.index_sampling.update_percentage: Percentage of index updates of total index size required before sampling of a given index is triggered.
- db.lock.acquisition.timeout: The maximum time interval within which lock should be acquired.
- db.logs.query.early_raw_logging_enabled: Log query text and parameters without obfuscating passwords.
- db.logs.query.enabled: Log executed queries.
- db.logs.query.max_parameter_length: Sets a maximum character length use for each parameter in the log.
- db.logs.query.obfuscate_literals: Obfuscates all literals of the query before writing to the log.
- db.logs.query.parameter_logging_enabled: Log parameters for the executed queries being logged.
- db.logs.query.plan_description_enabled: Log query plan description table, useful for debugging purposes.
- db.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.
- db.logs.query.transaction.enabled: Log the start and end of a transaction.
- db.logs.query.transaction.threshold: If the transaction is open for more time than this threshold, the transaction is logged once completed - provided transaction logging (db.logs.query.transaction.enabled) is set to `INFO`.
- db.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.

- db.memory.pagecache.warmup.preload: Page cache warmup can be configured to prefetch files, preferably when cache size is bigger than store size.
- db.memory.pagecache.warmup.preload.allowlist: Page cache warmup prefetch file allowlist regex.
- db.memory.pagecache.warmup.profile.interval: The profiling frequency for the page cache.
- db.memory.transaction.max: 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').
- db.memory.transaction.total.max: 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').
- db.recovery.fail_on_missing_files: If `true`, Neo4j will abort recovery if transaction log files are missing.
- db.relationship_grouping_threshold: Relationship count threshold for considering a node to be dense.
- db.shutdown_transaction_end_timeout: The maximum amount of time to wait for running transactions to complete before allowing initiated database shutdown to continue.
- db.store.files.preallocate: Specify if Neo4j should try to preallocate store files as they grow.
- db.temporal.timezone: Database timezone for temporal functions.
- db.track_query_cpu_time: Enables or disables tracking of how much time a query spends actively executing on the CPU.
- db.transaction.bookmark_ready_timeout: The maximum amount of time to wait for the database state represented by the bookmark.
- db.transaction.concurrent.maximum: The maximum number of concurrently running transactions.
- db.transaction.monitor.check.interval: Configures the time interval between transaction monitor checks.
- db.transaction.sampling.percentage: Transaction sampling percentage.
- db.transaction.timeout: The maximum time interval of a transaction within which it should be completed.
- db.transaction.tracing.level: Transaction creation tracing level.
- db.tx_log.buffer.size: On serialization of transaction logs, they will be temporary stored in the byte buffer that will be flushed at the end of the transaction or at any moment when buffer will be full.
- db.tx_log.preallocate: Specify if Neo4j should try to preallocate logical log file in advance.
- db.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.
- db.tx_log.rotation.size: Specifies at which file size the logical log will auto-rotate.
- db.tx_state.memory_allocation: Defines whether memory for transaction state should be allocated on- or off-heap.
- dbms.cluster.catchup.client_inactivity_timeout: Enterprise edition The catch up protocol times out if the given duration elapses with no network activity.
- dbms.cluster.discovery.endpoints: Enterprise edition A comma-separated list of endpoints which a server should contact in order to discover other cluster members.

* dbms.cluster.discovery.log_level: Enterprise edition The level of middleware logging.
* dbms.cluster.discovery.type: Enterprise edition Configure the discovery type used for cluster name resolution.
* dbms.cluster.minimum_initial_system_primaries_count: Enterprise edition Minimum number of machines initially required to formed a clustered DBMS.
* dbms.cluster.network.handshake_timeout: Enterprise edition Time out for protocol negotiation handshake.
* dbms.cluster.network.max_chunk_size: Enterprise edition Maximum chunk size allowable across network by clustering machinery.
* dbms.cluster.network.supported_compression_algos: Enterprise edition Network compression algorithms that this instance will allow in negotiation as a comma-separated list.
* dbms.cluster.raft.binding_timeout: Enterprise edition The time allowed for a database on a Neo4j server to either join a cluster or form a new cluster with the other Neo4j Servers provided by `dbms.cluster.discovery.endpoints`.
* dbms.cluster.raft.client.max_channels: Enterprise edition The maximum number of TCP channels between two nodes to operate the raft protocol.
* dbms.cluster.raft.election_failure_detection_window: Enterprise edition The rate at which leader elections happen.
* dbms.cluster.raft.leader_failure_detection_window: Enterprise edition The time window within which the loss of the leader is detected and the first re-election attempt is held.
* dbms.cluster.raft.leader_transfer.balancing_strategy: Enterprise edition Which strategy to use when transferring database leaderships around a cluster.
* dbms.cluster.raft.log.pruning_frequency: Enterprise edition RAFT log pruning frequency.
* dbms.cluster.raft.log.reader_pool_size: Enterprise edition RAFT log reader pool size.
* dbms.cluster.raft.log.rotation_size: Enterprise edition RAFT log rotation size.
* dbms.cluster.raft.membership.join_max_lag: Enterprise edition Maximum amount of lag accepted for a new follower to join the Raft group.
* dbms.cluster.raft.membership.join_timeout: Enterprise edition Time out for a new member to catch up.
* dbms.cluster.store_copy.max_retry_time_per_request: Enterprise edition Maximum retry time per request during store copy.
* dbms.cypher.forbid_exhaustive_shortestpath: This setting is associated with performance optimization.
* dbms.cypher.forbid_shortestpath_common_nodes: This setting is associated with performance optimization.
* dbms.cypher.hints_error: Set this to specify the behavior when Cypher planner or runtime hints cannot be fulfilled.
* dbms.cypher.lenient_create_relationship: Set this to change the behavior for Cypher create relationship when the start or end node is missing.
* dbms.cypher.min_replan_interval: The minimum time between possible cypher query replanning

events.

- dbms.cypher.planner: Set this to specify the default planner for the default language version.
- dbms.cypher.render_plan_description: If set to `true` a textual representation of the plan description will be rendered on the server for all queries running with `EXPLAIN` or `PROFILE`.
- dbms.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.
- dbms.databases.seed_from_uri_providers: Enterprise edition Databases may be created from an existing 'seed' (a database backup or dump) stored at some source URI.
- dbms.db.timezone: Database timezone.
- dbms.kubernetes.address: Enterprise edition Address for Kubernetes API.
- dbms.kubernetes.ca_crt: Enterprise edition File location of CA certificate for Kubernetes API.
- dbms.kubernetes.cluster_domain: Enterprise edition Kubernetes cluster domain.
- dbms.kubernetes.label_selector: Enterprise edition LabelSelector for Kubernetes API.
- dbms.kubernetes.namespace: Enterprise edition File location of namespace for Kubernetes API.
- dbms.kubernetes.service_port_name: Enterprise edition Service port name for discovery for Kubernetes API.
- dbms.kubernetes.token: Enterprise edition File location of token for Kubernetes API.
- dbms.logs.http.enabled: Enable HTTP request logging.
- dbms.memory.tracking.enable: Enable off heap and on heap memory tracking.
- dbms.memory.transaction.total.max: 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.netty.ssl.provider: Netty SSL provider.
- dbms.routing.client_side.enforce_for_domains: Always use client side routing (regardless of the default router) for neo4j:// protocol connections to these domains.
- dbms.routing.default_router: Routing strategy for neo4j:// protocol connections. Default is `CLIENT`, using client-side routing, with server-side routing as a fallback (if enabled). When set to `SERVER`, client-side routing is short-circuited, and requests will rely on server-side routing (which must be enabled for proper operation, i.e.
- dbms.routing.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 server-side routing in clusters using an additional bolt connector. When configured, this allows requests to be forwarded from one cluster member to another, if the requests can't be satisfied by the first member (e.g.
- dbms.routing.load_balancing.plugin: Enterprise edition The load balancing plugin to use.
- dbms.routing.load_balancing.shuffle_enabled: Enterprise edition Enables shuffling of the returned load balancing result.
- dbms.routing.reads_on_primaries_enabled: Enterprise edition Configure if the `dbms.routing.getRoutingTable()` procedure should include non-writer primaries as read endpoints or return only secondaries.
- dbms.routing.reads_on_writers_enabled: Enterprise edition Configure if the `dbms.routing.getRoutingTable()` procedure should include the writer as read endpoint or return only non-writers (non writer primaries and secondaries) Note: writer is returned as read endpoint if no other member is present all.
- 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: Enterprise edition The maximum capacity for authentication and authorization caches (respectively).
- dbms.security.auth_cache_ttl: Enterprise edition 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: Enterprise edition 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: Enterprise edition A list of security authentication providers containing the users and roles.
- dbms.security.authorization_providers: Enterprise edition A list of security authorization providers containing the users and roles.
- dbms.security.cluster_status_auth_enabled: Enterprise edition 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_strict_transport_security: Value of the HTTP Strict-Transport-Security (HSTS) response header.
- dbms.security.key.name: Enterprise edition Name of the 256 length AES encryption key, which is used for the symmetric encryption.
- dbms.security.keystore.password: Enterprise edition Password for accessing the keystore holding a 256 length AES encryption key, which is used for the symmetric encryption.
- dbms.security.keystore.path: Enterprise edition Location of the keystore holding a 256 length AES encryption key, which is used for the symmetric encryption of secrets held in system database.
- dbms.security.ldap.authentication.attribute: Enterprise edition The attribute to use when looking up users. Using this setting requires `dbms.security.ldap.authentication.search_for_attribute` to be true and thus `dbms.security.ldap.authorization.system_username` and `dbms.security.ldap.authorization.system_password` to be configured.
- dbms.security.ldap.authentication.cache_enabled: Enterprise edition Determines if the result of authentication via the LDAP server should be cached or not.
- dbms.security.ldap.authentication.mechanism: Enterprise edition LDAP authentication mechanism.
- dbms.security.ldap.authentication.search_for_attribute: Enterprise edition Perform authentication by searching for an unique attribute of a user. Using this setting requires `dbms.security.ldap.authorization.system_username` and `dbms.security.ldap.authorization.system_password` to be configured.
- dbms.security.ldap.authentication.user_dn_template: Enterprise edition LDAP user DN template.
- dbms.security.ldap.authorization.access_permitted_group: Enterprise edition The LDAP group to which a user must belong to get any access to the system.Set this to restrict access to a subset of LDAP users belonging to a particular group.
- dbms.security.ldap.authorization.group_membership_attributes: Enterprise edition 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: Enterprise edition An authorization mapping from LDAP group names to Neo4j role names.

- dbms.security.ldap.authorization.nested_groups_enabled: Enterprise edition This setting determines whether multiple LDAP search results will be processed (as is required for the lookup of nested groups).
- dbms.security.ldap.authorization.nested_groups_search_filter: Enterprise edition The search template which will be used to find the nested groups which the user is a member of.
- dbms.security.ldap.authorization.system_password: Enterprise edition 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: Enterprise edition 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: Enterprise edition 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: Enterprise edition 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: Enterprise edition The LDAP search filter to search for a user principal when LDAP authorization is enabled.
- dbms.security.ldap.connection_timeout: Enterprise edition The timeout for establishing an LDAP connection.
- dbms.security.ldap.host: Enterprise edition URL of LDAP server to use for authentication and authorization.
- dbms.security.ldap.read_timeout: Enterprise edition The timeout for an LDAP read request (i.e.
- dbms.security.ldap.referral: Enterprise edition The LDAP referral behavior when creating a connection.
- dbms.security.ldap.use_starttls: Enterprise edition Use secure communication with the LDAP server using opportunistic TLS.
- dbms.security.log_successful_authentication: Enterprise edition Set to log successful authentication events to the security log.
- dbms.security.oidc.<provider>.audience: Enterprise edition Expected values of the Audience (aud) claim in the id token.
- dbms.security.oidc.<provider>.auth_endpoint: Enterprise edition The OIDC authorization endpoint.
- dbms.security.oidc.<provider>.auth_flow: Enterprise edition The OIDC flow to use.
- dbms.security.oidc.<provider>.auth_params: Enterprise edition Optional additional parameters that the auth endpoint requires.
- dbms.security.oidc.<provider>.authorization.group_to_role_mapping: Enterprise edition An authorization mapping from IdP group names to Neo4j role names.
- dbms.security.oidc.<provider>.claims.groups: Enterprise edition The claim to use as the list of groups in Neo4j.
- dbms.security.oidc.<provider>.claims.username: Enterprise edition The claim to use as the username in

Neo4j.

- dbms.security.oidc.<provider>.client_id: Enterprise edition Client id needed if token contains multiple Audience (aud) claims.
- dbms.security.oidc.<provider>.config: Enterprise edition
- dbms.security.oidc.<provider>.display_name: Enterprise edition The user-facing name of the provider as provided by the discovery endpoint to clients (Bloom, Browser etc.).
- dbms.security.oidc.<provider>.get_groups_from_user_info: Enterprise edition When turned on, Neo4j gets the groups from the provider user info endpoint.
- dbms.security.oidc.<provider>.get_username_from_user_info: Enterprise edition When turned on, Neo4j gets the username from the provider user info endpoint.
- dbms.security.oidc.<provider>.issuer: Enterprise edition The expected value of the iss claim in the id token.
- dbms.security.oidc.<provider>.jwks_uri: Enterprise edition The location of the JWK public key set for the identity provider.
- dbms.security.oidc.<provider>.params: Enterprise edition The map is a semicolon separated list of key-value pairs.
- dbms.security.oidc.<provider>.token_endpoint: Enterprise edition The OIDC token endpoint.
- dbms.security.oidc.<provider>.token_params: Enterprise edition Optional query parameters that the token endpoint requires.
- dbms.security.oidc.<provider>.user_info_uri: Enterprise edition The identity providers user info uri.
- dbms.security.oidc.<provider>.well_known_discovery_uri: Enterprise edition The 'well known' OpenID Connect Discovery endpoint used to fetch identity provider settings.
- dbms.security.procedures.allowlist: A list of procedures (comma separated) that are to be loaded.
- dbms.security.procedures.unrestricted: A list of procedures and user defined functions (comma separated) that are allowed full access to the database.
- initial.dbms.database_allocator: Enterprise edition Name of the initial database allocator.
- initial.dbms.default_database: Name of the default database (aliases are not supported).
- initial.dbms.default_primaries_count: Enterprise edition Initial default number of primary instances of user databases.
- initial.dbms.default_secondaries_count: Enterprise edition Initial default number of secondary instances of user databases.
- initial.server.allowed_databases: Enterprise edition The names of databases that are allowed on this server - all others are denied.
- initial.server.denied_databases: Enterprise edition The names of databases that are not allowed on this server.
- initial.server.mode_constraint: Enterprise edition An instance can restrict itself to allow databases to be hosted only as primaries or secondaries.
- server.backup.enabled: Enterprise edition Enable support for running online backups.
- server.backup.listen_address: Enterprise edition Network interface and port for the backup server to

listen on.

- server.backup.store_copy_max_retry_time_per_request: Enterprise edition Maximum retry time per request during store copy.
- server.bolt.advertised_address: Advertised address for this connector.
- server.bolt.connection_keep_alive: The maximum time to wait before sending a NOOP on connections waiting for responses from active ongoing queries.The minimum value is 1 millisecond.
- server.bolt.connection_keep_alive_for_requests: The type of messages to enable keep-alive messages for (ALL, STREAMING or OFF).
- server.bolt.connection_keep_alive_probes: The total amount of probes to be missed before a connection is considered stale.The minimum for this value is 1.
- server.bolt.connection_keep_alive_streaming_scheduling_interval: The interval between every scheduled keep-alive check on all connections with active queries.
- server.bolt.enabled: Enable the bolt connector.
- server.bolt.listen_address: Address the connector should bind to.
- server.bolt.ocsp_stapling_enabled: Enable server OCSP stapling for bolt and http connectors.
- server.bolt.thread_pool_keep_alive: The maximum time an idle thread in the thread pool bound to this connector will wait for new tasks.
- server.bolt.thread_pool_max_size: The maximum number of threads allowed in the thread pool bound to this connector.
- server.bolt.thread_pool_min_size: The number of threads to keep in the thread pool bound to this connector, even if they are idle.
- server.bolt.tls_level: Encryption level to require this connector to use.
- server.cluster.advertised_address: Enterprise edition Advertised hostname/IP address and port for the transaction shipping server.
- server.cluster.catchup.connect_randomly_to_server_group: Enterprise edition Comma separated list of groups to be used by the connect-randomly-to-server-group selection strategy.
- server.cluster.catchup.upstream_strategy: Enterprise edition An ordered list in descending preference of the strategy which secondaries use to choose the upstream server from which to pull transactional updates.
- server.cluster.catchup.user_defined_upstream_strategy: Enterprise edition Configuration of a user-defined upstream selection strategy.
- server.cluster.listen_address: Enterprise edition Network interface and port for the transaction shipping server to listen on.
- server.cluster.network.native_transport_enabled: Enterprise edition Use native transport if available.
- server.cluster.raft.advertised_address: Enterprise edition Advertised hostname/IP address and port for the RAFT server.
- server.cluster.raft.listen_address: Enterprise edition Network interface and port for the RAFT server to listen on.
- server.cluster.system_database_mode: Enterprise edition Users must manually specify the mode for

the system database on each instance.

- server.config.strict_validation.enabled: A strict configuration validation will prevent the database from starting up if unknown configuration options are specified in the neo4j settings namespace (such as dbms., cypher., etc) or if settings are declared multiple times.
- server.databases.default_to_read_only: Whether or not any database on this instance are read_only by default.
- server.databases.read_only: List of databases for which to prevent write queries.
- server.databases.writable: List of databases for which to allow write queries.
- server.db.query_cache_size: The number of cached Cypher query execution plans per database.
- server.default_advertised_address: Default hostname or IP address the server uses to advertise itself.
- server.default_listen_address: Default network interface to listen for incoming connections.
- server.directories.cluster_state: Enterprise edition Directory to hold cluster state including Raft log.
- server.directories.data: Path of the data directory.
- server.directories.dumps.root: Root location where Neo4j will store database dumps optionally produced when dropping said databases.
- server.directories.import: Sets the root directory for file URLs used with the Cypher `LOAD CSV` clause.
- server.directories.lib: Path of the lib directory.
- server.directories.licenses: Path of the licenses directory.
- server.directories.logs: Path of the logs directory.
- server.directories.metrics: Enterprise edition The target location of the CSV files: a path to a directory wherein a CSV file per reported field will be written.
- server.directories.neo4j_home: Root relative to which directory settings are resolved.
- server.directories.plugins: Location of the database plugin directory.
- server.directories.run: Path of the run directory.
- server.directories.script.root: Root location where Neo4j will store scripts for configured databases.
- server.directories.transaction.logs.root: Root location where Neo4j will store transaction logs for configured databases.
- server.discovery.advertised_address: Enterprise edition Advertised cluster member discovery management communication.
- server.discovery.listen_address: Enterprise edition Host and port to bind the cluster member discovery management communication.
- server.dynamic.setting.allowlist: Enterprise edition A list of setting name patterns (comma separated) that are allowed to be dynamically changed.
- server.groups: Enterprise edition A list of tag names for the server used when configuring load balancing and replication policies.
- server.http.advertised_address: Advertised address for this connector.
- server.http.enabled: Enable the http connector.

- server.http.listen_address: Address the connector should bind to.
- server.http_enabled_modules: Defines the set of modules loaded into the Neo4j web server.
- server.https.advertised_address: Advertised address for this connector.
- server.https.enabled: Enable the https connector.
- server.https.listen_address: Address the connector should bind to.
- server.jvm.additional: Additional JVM arguments.
- server.logs.config: Path to the logging configuration for debug, query, http and security logs.
- server.logs.debug.enabled: Enable the debug log.
- server.logs.gc.enabled: Enable GC Logging.
- server.logs.gc.options: GC Logging Options.
- server.logs.gc.rotation.keep_number: Number of GC logs to keep.
- server.logs.gc.rotation.size: Size of each GC log that is kept.
- server.logs.user.config: Path to the logging configuration of user logs.
- server.max_databases: Enterprise edition The maximum number of databases.
- server.memory.heap.initial_size: Initial heap size.
- server.memory.heap.max_size: Maximum heap size.
- server.memory.off_heap.block_cache_size: Defines the size of the off-heap memory blocks cache.
- server.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.
- server.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.
- server.memory.pagecache.directio: Use direct I/O for page cache.
- server.memory.pagecache.flush.buffer.enabled: Page cache can be configured to use a temporal buffer for flushing purposes.
- server.memory.pagecache.flush.buffer.size_in_pages: Page cache can be configured to use a temporal buffer for flushing purposes.
- server.memory.pagecache.scan.prefetchers: The maximum number of worker threads to use for pre-fetching data when doing sequential scans.
- server.memory.pagecache.size: The amount of memory to use for mapping the store files.
- server.metrics.csv.enabled: Enterprise edition Set to `true` to enable exporting metrics to CSV files.
- server.metrics.csv.interval: Enterprise edition The reporting interval for the CSV files.
- server.metrics.csv.rotation.compression: Enterprise edition Decides what compression to use for the csv history files.
- server.metrics.csv.rotation.keep_number: Enterprise edition Maximum number of history files for the csv files.
- server.metrics.csv.rotation.size: Enterprise edition The file size in bytes at which the csv files will auto-rotate.

- server.metrics.enabled: Enterprise edition Enable metrics.
- server.metrics.filter: Enterprise edition Specifies which metrics should be enabled by using a comma separated list of globbing patterns.
- server.metrics.graphite.enabled: Enterprise edition Set to `true` to enable exporting metrics to Graphite.
- server.metrics.graphite.interval: Enterprise edition The reporting interval for Graphite.
- server.metrics.graphite.server: Enterprise edition The hostname or IP address of the Graphite server.
- server.metrics.jmx.enabled: Enterprise edition Set to `true` to enable the JMX metrics endpoint.
- server.metrics.prefix: Enterprise edition A common prefix for the reported metrics field names.
- server.metrics.prometheus.enabled: Enterprise edition Set to `true` to enable the Prometheus endpoint.
- server.metrics.prometheus.endpoint: Enterprise edition The hostname and port to use as Prometheus endpoint.
- server.panic.shutdown_on_panic: Enterprise edition If there is a Database Management System Panic (an irrecoverable error) should the neo4j process shut down or continue running.
- server.routing.advertised_address: Enterprise edition The advertised address for the intra-cluster routing connector.
- server.routing.listen_address: The address the routing connector should bind to.
- server.threads.worker_count: Number of Neo4j worker threads.
- server.unmanaged_extension_classes: Comma-separated list of <classname>=<mount point> for unmanaged extensions.
- server.windows_service_name: Name of the Windows Service managing Neo4j when installed using `neo4j install-service`.

Table 103. browser.allow_outgoing_connections

Description	Enterprise edition Configure the policy for outgoing Neo4j Browser connections.
Valid values	browser.allow_outgoing_connections, a boolean
Default value	`true`

Table 104. browser.credential_timeout

Description	Enterprise edition 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 105. 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 106. 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 107. browser.retain_connection_credentials

Description	Enterprise edition Configure the Neo4j Browser to store or not store user credentials.
Valid values	browser.retain_connection_credentials, a boolean
Default value	`true`

Table 108. browser.retain_editor_history

Description	Enterprise edition Configure the Neo4j Browser to store or not store user editor history.
Valid values	browser.retain_editor_history, a boolean
Default value	`true`

Table 109. client.allow_telemetry

Description	Configure client applications such as Browser and Bloom to send Product Analytics data.
Valid values	client.allow_telemetry, a boolean
Default value	`true`

Table 110. db.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 'db.checkpoint.interval.tx' and 'db.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 'db.tx_log.rotation.retention_policy' setting.
Valid values	db.checkpoint, one of [PERIODIC, CONTINUOUS, VOLUME, VOLUMETRIC]
Default value	`PERIODIC`

Table 111. db.checkpoint.interval.time

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

Table 112. db.checkpoint.interval.tx

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

Table 113. db.checkpoint.interval.volume

Description	Configures the volume of transaction logs between check-points. The database will not check-point more often than this (unless check pointing is triggered by a different event), but might check-point less often than this interval, if performing a check-point takes longer time than the configured interval. A check-point is a point in the transaction logs, which recovery would start from. Longer check-point intervals typically mean that recovery will take longer to complete in case of a crash. On the other hand, a longer check-point interval can also reduce the I/O load that the database places on the system, as each check-point implies a flushing and forcing of all the store files.
Valid values	db.checkpoint.interval.volume, a byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `TiB`, `TB`, `PiB`, `PB`, `EiB`, `EB`) which is minimum `1.00KiB`
Default value	`250.00MiB`

Table 114. db.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	db.checkpoint.iops.limit, an integer
Dynamic	true
Default value	`600`

Table 115. db.cluster.catchup.pull_interval

Description	Enterprise edition Interval of pulling updates from cores.
Valid values	db.cluster.catchup.pull_interval, a duration (Valid units are: `ns`, `μs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`)
Default value	`1s`

Table 116. db.cluster.raft.apply.buffer.max_bytes

Description	Enterprise edition The maximum number of bytes in the apply buffer. This parameter limits the amount of memory that can be consumed by the apply buffer. If the bytes limit is reached, buffer size will be limited even if max_entries is not exceeded.
Valid values	db.cluster.raft.apply.buffer.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	`1.00GiB`

Table 117. db.cluster.raft.apply.buffer.max_entries

Description	Enterprise edition The maximum number of entries in the raft log entry prefetch buffer.
Valid values	db.cluster.raft.apply.buffer.max_entries, an integer
Default value	`1024`

Table 118. db.cluster.raft.in_queue.batch.max_bytes

Description	Enterprise edition Largest batch processed by RAFT in bytes.
Valid values	db.cluster.raft.in_queue.batch.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	`8.00MiB`

Table 119. db.cluster.raft.in_queue.max_bytes

Description	Enterprise edition Maximum number of bytes in the RAFT in-queue.
Valid values	db.cluster.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 120. db.cluster.raft.leader_transfer.priority_group

Description	Enterprise edition 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 `db.cluster.raft.leader_transfer.priority_group`.<database> the specified priority group will apply to that database only. If no database is specified that group will be the default and apply to all databases which have no priority group explicitly set. Using this setting will disable leadership balancing.
Valid values	db.cluster.raft.leader_transfer.priority_group, a string identifying a Server Tag
Default value	

Table 121. db.cluster.raft.log.prune_strategy

Description	Enterprise edition RAFT log pruning strategy that determines which logs are to be pruned. Neo4j only prunes log entries up to the last applied index, which guarantees that logs are only marked for pruning once the transactions within are safely copied over to the local transaction logs and safely committed by a majority of cluster members. Possible values are a byte size or a number of transactions (e.g., 200K txs).
Valid values	db.cluster.raft.log.prune_strategy, a string
Default value	`1g size`

Table 122. db.cluster.raft.log_shipping.buffer.max_bytes

Description	Enterprise edition The maximum number of bytes in the in-flight cache. This parameter limits the amount of memory that can be consumed by cache. If the bytes limit is reached, cache size will be limited even if max_entries is not exceeded.
Valid values	db.cluster.raft.log_shipping.buffer.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	`1.00GiB`

Table 123. db.cluster.raft.log_shipping.buffer.max_entries

Description	Enterprise edition The maximum number of entries in the in-flight cache. Increasing size will require more memory but might improve performance in high load situations.
Valid values	db.cluster.raft.log_shipping.buffer.max_entries, an integer
Default value	`1024`

Table 124. db.filewatcher.enabled

Description	Allows the enabling or disabling of the file watcher service. This is an auxiliary service but should be left enabled in almost all cases.
Valid values	db.filewatcher.enabled, a boolean
Default value	`true`

Table 125. db.format

Description	Database format. This is the format that will be used for new databases. Valid values are `standard`, `aligned`, or `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.
Valid values	db.format, a string
Dynamic	true
Default value	`aligned`

Table 126. db.import.csv.buffer_size

Description	The size of the internal buffer in bytes used by `LOAD CSV`. If the csv file contains huge fields this value may have to be increased.
Valid values	db.import.csv.buffer_size, a long which is minimum `1`
Default value	`2097152`

Table 127. db.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	db.import.csv.legacy_quote_escaping, a boolean
Default value	`true`

Table 128. db.index.fulltext.default_analyzer

Description	The name of the analyzer that the fulltext indexes should use by default.
Valid values	db.index.fulltext.default_analyzer, a string
Default value	`standard-no-stop-words`

Table 129. db.index.fulltext.eventually_consistent

Description	Whether or not fulltext indexes should be eventually consistent by default or not.
Valid values	db.index.fulltext.eventually_consistent, a boolean
Default value	`false`

Table 130. db.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	db.index.fulltext.eventually_consistent_index_update_queue_max_length, an integer which is in the range `1` to `50000000`
Default value	`10000`

Table 131. db.index_sampling.background_enabled

Description	Enable or disable background index sampling.
Valid values	db.index_sampling.background_enabled, a boolean
Default value	`true`

Table 132. db.index_sampling.sample_size_limit

Description	Index sampling chunk size limit.
Valid values	db.index_sampling.sample_size_limit, an integer which is in the range `1048576` to `2147483647`
Default value	`8388608`

Table 133. db.index_sampling.update_percentage

Description	Percentage of index updates of total index size required before sampling of a given index is triggered.
Valid values	db.index_sampling.update_percentage, an integer which is minimum `0`
Default value	`5`

Table 134. db.lock.acquisition.timeout

Description	The maximum time interval within which lock should be acquired. Zero (default) means timeout is disabled.
Valid values	db.lock.acquisition.timeout, a duration (Valid units are: `ns`, `μs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`)
Dynamic	true
Default value	`0s`

Table 135. db.logs.query.early_raw_logging_enabled

Description	Log query text and parameters without obfuscating passwords. This allows queries to be logged earlier before parsing starts.
Valid values	db.logs.query.early_raw_logging_enabled, a boolean
Dynamic	true
Default value	`false`

Table 136. db.logs.query.enabled

Description	Log executed queries. Valid values are `OFF`, `INFO`, or `VERBOSE`.
	
	`OFF`
	no logging.
	
	`INFO`
	log queries at the end of execution, that take longer than the configured threshold, `db.logs.query.threshold`.
	
	`VERBOSE`
	log queries at the start and end of execution, regardless of `db.logs.query.threshold`.
	
	Log entries are written to the query log.
	
	This feature is available in the Neo4j Enterprise Edition.

Valid values	db.logs.query.enabled, one of [OFF, INFO, VERBOSE]
Dynamic	true
Default value	`VERBOSE`

Table 137. db.logs.query.max_parameter_length

Description	Sets a maximum character length use for each parameter in the log. This only takes effect if `db.logs.query.parameter_logging_enabled = true`.
Valid values	db.logs.query.max_parameter_length, an integer
Dynamic	true
Default value	`2147483647`

Table 138. db.logs.query.obfuscate_literals

Description	Obfuscates all literals of the query before writing to the log. Note that node labels, relationship types and map property keys are still shown. Changing the setting will not affect queries that are cached. So, if you want the switch to have immediate effect, you must also call `CALL db.clearQueryCaches()`.
Valid values	db.logs.query.obfuscate_literals, a boolean
Dynamic	true
Default value	`false`

Table 139. db.logs.query.parameter_logging_enabled

Description	Log parameters for the executed queries being logged.
Valid values	db.logs.query.parameter_logging_enabled, a boolean
Dynamic	true
Default value	`true`

Table 140. db.logs.query.plan_description_enabled

Description	Log query plan description table, useful for debugging purposes.
Valid values	db.logs.query.plan_description_enabled, a boolean
Dynamic	true
Default value	`false`

Table 141. db.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	db.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 142. db.logs.query.transaction.enabled

Description	Log the start and end of a transaction. Valid values are 'OFF', 'INFO', or 'VERBOSE'. OFF: no logging. INFO: log start and end of transactions that take longer than the configured threshold, db.logs.query.transaction.threshold. VERBOSE: log start and end of all transactions. Log entries are written to the query log. This feature is available in the Neo4j Enterprise Edition.
Valid values	db.logs.query.transaction.enabled, one of [OFF, INFO, VERBOSE]
Dynamic	true

Default value	`OFF`

Table 143. db.logs.query.transaction.threshold

Description	If the transaction is open for more time than this threshold, the transaction is logged once completed - provided transaction logging (db.logs.query.transaction.enabled) is set to `INFO`. Defaults to 0 seconds (all transactions are logged).
Valid values	db.logs.query.transaction.threshold, a duration (Valid units are: `ns`, `μs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`)
Dynamic	true
Default value	`0s`

Table 144. db.memory.pagecache.warmup.enable

Description	Page cache can be configured to perform usage sampling of loaded pages that can be used to construct active load profile. According to that profile pages can be reloaded on the restart, replication, etc. This setting allows disabling that behavior. This feature is available in Neo4j Enterprise Edition.
Valid values	db.memory.pagecache.warmup.enable, a boolean
Default value	`true`

Table 145. db.memory.pagecache.warmup.preload

Description	Page cache warmup can be configured to prefetch files, preferably when cache size is bigger than store size. Files to be prefetched can be filtered by 'dbms.memory.pagecache.warmup.preload.allowlist'. Enabling this disables warmup by profile.
Valid values	db.memory.pagecache.warmup.preload, a boolean
Default value	`false`

Table 146. db.memory.pagecache.warmup.preload.allowlist

Description	Page cache warmup prefetch file allowlist regex. By default matches all files.
Valid values	db.memory.pagecache.warmup.preload.allowlist, a string
Default value	`.*`

Table 147. db.memory.pagecache.warmup.profile.interval

Description	The profiling frequency for the page cache. Accurate profiles allow the page cache to do active warmup after a restart, reducing the mean time to performance. This feature is available in Neo4j Enterprise Edition.
Valid values	db.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 148. db.memory.transaction.max

Description	Limit the amount of memory that a single transaction can consume, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g'). Zero means 'largest possible value'.
Valid values	db.memory.transaction.max, 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`
Dynamic	true
Default value	`0B`

Table 149. db.memory.transaction.total.max

Description	Limit the amount of memory that all transactions in one database can consume, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g'). Zero means 'unlimited'.
Valid values	db.memory.transaction.total.max, 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 150. db.recovery.fail_on_missing_files

Description	If `true`, Neo4j will abort recovery if transaction log files are missing. Setting this to `false` will allow Neo4j to create new empty missing files for the already existing database, but the integrity of the database might be compromised.
Valid values	db.recovery.fail_on_missing_files, a boolean

Default value	`true`

Table 151. db.relationship_grouping_threshold

Description	Relationship count threshold for considering a node to be dense.
Valid values	db.relationship_grouping_threshold, an integer which is minimum `1`
Default value	`50`

Table 152. db.shutdown_transaction_end_timeout

Description	The maximum amount of time to wait for running transactions to complete before allowing initiated database shutdown to continue.
Valid values	db.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 153. db.store.files.preallocate

Description	Specify if Neo4j should try to preallocate store files as they grow.
Valid values	db.store.files.preallocate, a boolean
Default value	`true`

Table 154. 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 155. db.track_query_cpu_time

Description	Enables or disables tracking of how much time a query spends actively executing on the CPU. Calling `SHOW TRANSACTIONS` will display the time. This can also be logged in the query log by using `db.logs.query.time_logging_enabled`.
Valid values	db.track_query_cpu_time, a boolean

Dynamic	true
Default value	`false`

Table 156. db.transaction.bookmark_ready_timeout

Description	The maximum amount of time to wait for the database state represented by the bookmark.
Valid values	db.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`
Dynamic	true
Default value	`30s`

Table 157. db.transaction.concurrent.maximum

Description	The maximum number of concurrently running transactions. If set to 0, limit is disabled.
Valid values	db.transaction.concurrent.maximum, an integer
Dynamic	true
Default value	`1000`

Table 158. db.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	db.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 159. db.transaction.sampling.percentage

Description	Transaction sampling percentage.
Valid values	db.transaction.sampling.percentage, an integer which is in the range `1` to `100`
Dynamic	true

Default value	`5`

Table 160. db.transaction.timeout

Description	The maximum time interval of a transaction within which it should be completed.
Valid values	db.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 161. db.transaction.tracing.level

Description	Transaction creation tracing level.
Valid values	db.transaction.tracing.level, one of [DISABLED, SAMPLE, ALL]
Dynamic	true
Default value	`DISABLED`

Table 162. db.tx_log.buffer.size

Description	On serialization of transaction logs, they will be temporary stored in the byte buffer that will be flushed at the end of the transaction or at any moment when buffer will be full.
Valid values	db.tx_log.buffer.size, a long which is minimum `131072`
Default value	`By default the size of byte buffer is based on number of available cpu's with minimal buffer size of 512KB. Every another 4 cpu's will add another 512KB into the buffer size. Maximal buffer size in this default scheme is 4MB taking into account that we can have one transaction log writer per database in multi-database env.For example, runtime with 4 cpus will have buffer size of 1MB; runtime with 8 cpus will have buffer size of 1MB 512KB; runtime with 12 cpus will have buffer size of 2MB.`

Table 163. db.tx_log.preallocate

Description	Specify if Neo4j should try to preallocate logical log file in advance.
Valid values	db.tx_log.preallocate, a boolean
Dynamic	true
Default value	`true`

Table 164. db.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	db.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	`2 days`

Table 165. db.tx_log.rotation.size

Description	Specifies at which file size the logical log will auto-rotate. Minimum accepted value is 128 KiB.
Valid values	db.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	`256.00MiB`

Table 166. db.tx_state.memory_allocation

Description	Defines whether memory for transaction state should be allocated on- or off-heap. Note that for small transactions you can gain up to 25% write speed by setting it to `ON_HEAP`.
Valid values	db.tx_state.memory_allocation, one of [ON_HEAP, OFF_HEAP]
Default value	`ON_HEAP`

Table 167. dbms.cluster.catchup.client_inactivity_timeout

Description	Enterprise edition 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	dbms.cluster.catchup.client_inactivity_timeout, a duration (Valid units are: `ns`, `μs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`)
Default value	`10m`

Table 168. dbms.cluster.discovery.endpoints

Description	Enterprise edition A comma-separated list of endpoints which a server should contact in order to discover other cluster members.
Valid values	dbms.cluster.discovery.endpoints, a ',' separated list with elements of type 'a socket address in the format 'hostname:port', 'hostname' or ':port''.

Table 169. dbms.cluster.discovery.log_level

Description	Enterprise edition The level of middleware logging.
Valid values	dbms.cluster.discovery.log_level, one of [DEBUG, INFO, WARN, ERROR, NONE]
Default value	`WARN`

Table 170. dbms.cluster.discovery.type

Description	Enterprise edition Configure the discovery type used for cluster name resolution.
Valid values	dbms.cluster.discovery.type, one of [DNS, LIST, SRV, K8S] which may require different settings depending on the discovery type: `DNS requires [dbms.cluster.discovery.endpoints], LIST requires [], SRV requires [dbms.cluster.discovery.endpoints], K8S requires [dbms.kubernetes.label_selector, dbms.kubernetes.service_port_name]`
Default value	`LIST`

Table 171. dbms.cluster.minimum_initial_system_primaries_count

Description	Enterprise edition Minimum number of machines initially required to form a clustered DBMS. The cluster is considered formed when at least this many members have discovered each other, bound together and bootstrapped a highly available system database. As a result, at least this many of the cluster's initial machines must have 'server.cluster.system_database_mode' set to 'PRIMARY'.NOTE: If 'dbms.cluster.discovery.type' is set to 'LIST' and 'dbms.cluster.discovery.endpoints' is empty then the user is assumed to be deploying a standalone DBMS, and the value of this setting is ignored.
Valid values	dbms.cluster.minimum_initial_system_primaries_count, an integer which is minimum `1`

Default value	`3`

Table 172. *dbms.cluster.network.handshake_timeout*

Description	Enterprise edition Time out for protocol negotiation handshake.
Valid values	dbms.cluster.network.handshake_timeout, a duration (Valid units are: `ns`, `μs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`)
Default value	`20s`

Table 173. *dbms.cluster.network.max_chunk_size*

Description	Enterprise edition Maximum chunk size allowable across network by clustering machinery.
Valid values	dbms.cluster.network.max_chunk_size, an integer which is in the range `4096` to `10485760`
Default value	`32768`

Table 174. *dbms.cluster.network.supported_compression_algos*

Description	Enterprise edition 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	dbms.cluster.network.supported_compression_algos, a ',' separated list with elements of type 'a string'.
Default value	

Table 175. *dbms.cluster.raft.binding_timeout*

Description	Enterprise edition The time allowed for a database on a Neo4j server to either join a cluster or form a new cluster with the other Neo4j Servers provided by `dbms.cluster.discovery.endpoints`.
Valid values	dbms.cluster.raft.binding_timeout, a duration (Valid units are: `ns`, `μs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`)
Default value	`10m`

Table 176. dbms.cluster.raft.client.max_channels

Description	Enterprise edition The maximum number of TCP channels between two nodes to operate the raft protocol. Each database gets allocated one channel, but a single channel can be used by more than one database.
Valid values	dbms.cluster.raft.client.max_channels, an integer
Default value	`8`

Table 177. dbms.cluster.raft.election_failure_detection_window

Description	Enterprise edition 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	dbms.cluster.raft.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 178. dbms.cluster.raft.leader_failure_detection_window

Description	Enterprise edition 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	dbms.cluster.raft.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 179. dbms.cluster.raft.leader_transfer.balancing_strategy

Description	Enterprise edition 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	dbms.cluster.raft.leader_transfer.balancing_strategy, one of [NO_BALANCING, EQUAL_BALANCING]
Default value	`EQUAL_BALANCING`

Table 180. dbms.cluster.raft.log.pruning_frequency

Description	Enterprise edition RAFT log pruning frequency.
Valid values	dbms.cluster.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 181. dbms.cluster.raft.log.reader_pool_size

Description	Enterprise edition RAFT log reader pool size.
Valid values	dbms.cluster.raft.log.reader_pool_size, an integer
Default value	`8`

Table 182. dbms.cluster.raft.log.rotation_size

Description	Enterprise edition RAFT log rotation size.
Valid values	dbms.cluster.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 183. dbms.cluster.raft.membership.join_max_lag

Description	Enterprise edition Maximum amount of lag accepted for a new follower to join the Raft group.
Valid values	dbms.cluster.raft.membership.join_max_lag, a duration (Valid units are: `ns`, `μs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`)
Default value	`10s`

Table 184. dbms.cluster.raft.membership.join_timeout

Description	Enterprise edition Time out for a new member to catch up.
Valid values	dbms.cluster.raft.membership.join_timeout, a duration (Valid units are: `ns`, `μs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`)
Default value	`10m`

Table 185. dbms.cluster.store_copy.max_retry_time_per_request

Description	Enterprise edition 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	dbms.cluster.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 186. dbms.cypher.forbid_exhaustive_shortestpath

Description	This setting is associated with performance optimization. Set this to `true` in situations where it is preferable to have any queries using the 'shortestPath' function terminate as soon as possible with no answer, rather than potentially running for a long time attempting to find an answer (even if there is no path to be found). For most queries, the 'shortestPath' algorithm will return the correct answer very quickly. However there are some cases where it is possible that the fast bidirectional breadth-first search algorithm will find no results even if they exist. This can happen when the predicates in the `WHERE` clause applied to 'shortestPath' cannot be applied to each step of the traversal, and can only be applied to the entire path. When the query planner detects these special cases, it will plan to perform an exhaustive depth-first search if the fast algorithm finds no paths. However, the exhaustive search may be orders of magnitude slower than the fast algorithm. If it is critical that queries terminate as soon as possible, it is recommended that this option be set to `true`, which means that Neo4j will never consider using the exhaustive search for shortestPath queries. However, please note that if no paths are found, an error will be thrown at run time, which will need to be handled by the application.
Valid values	dbms.cypher.forbid_exhaustive_shortestpath, a boolean
Default value	`false`

Table 187. dbms.cypher.forbid_shortestpath_common_nodes

Description	This setting is associated with performance optimization. The shortest path algorithm does not work when the start and end nodes are the same. With this setting set to `false` no path will be returned when that happens. The default value of `true` will instead throw an exception. This can happen if you perform a shortestPath search after a cartesian product that might have the same start and end nodes for some of the rows passed to shortestPath. If it is preferable to not experience this exception, and acceptable for results to be missing for those rows, then set this to `false`. If you cannot accept missing results, and really want the shortestPath between two common nodes, then re-write the query using a standard Cypher variable length pattern expression followed by ordering by path length and limiting to one result.

Valid values	dbms.cypher.forbid_shortestpath_common_nodes, a boolean
Default value	`true`

Table 188. dbms.cypher.hints_error

Description	Set this to specify the behavior when Cypher planner or runtime hints cannot be fulfilled. If true, then non-conformance will result in an error, otherwise only a warning is generated.
Valid values	dbms.cypher.hints_error, a boolean
Default value	`false`

Table 189. dbms.cypher.lenient_create_relationship

Description	Set this to change the behavior for Cypher create relationship when the start or end node is missing. By default this fails the query and stops execution, but by setting this flag the create operation is simply not performed and execution continues.
Valid values	dbms.cypher.lenient_create_relationship, a boolean
Default value	`false`

Table 190. dbms.cypher.min_replan_interval

Description	The minimum time between possible cypher query replanning events. After this time, the graph statistics will be evaluated, and if they have changed by more than the value set by dbms.cypher.statistics_divergence_threshold, the query will be replanned. If the statistics have not changed sufficiently, the same interval will need to pass before the statistics will be evaluated again. Each time they are evaluated, the divergence threshold will be reduced slightly until it reaches 10% after 7h, so that even moderately changing databases will see query replanning after a sufficiently long time interval.
Valid values	dbms.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 191. dbms.cypher.planner

Description	Set this to specify the default planner for the default language version.
Valid values	dbms.cypher.planner, one of [DEFAULT, COST]

Default value	`DEFAULT`

Table 192. dbms.cypher.render_plan_description

Description	If set to `true` a textual representation of the plan description will be rendered on the server for all queries running with `EXPLAIN` or `PROFILE`. This allows clients such as the neo4j browser and Cypher shell to show a more detailed plan description.
Valid values	dbms.cypher.render_plan_description, a boolean
Dynamic	true
Default value	`false`

Table 193. dbms.cypher.statistics_divergence_threshold

Description	The threshold for statistics above which a plan is considered stale. If any of the underlying statistics used to create the plan have changed more than this value, the plan will be considered stale and will be replanned. Change is calculated as `abs(a-b)/max(a,b)`. This means that a value of `0.75` requires the database to quadruple in size before query replanning. A value of `0` means that the query will be replanned as soon as there is any change in statistics and the replan interval has elapsed. This interval is defined by `dbms.cypher.min_replan_interval` and defaults to 10s. After this interval, the divergence threshold will slowly start to decline, reaching 10% after about 7h. This will ensure that long running databases will still get query replanning on even modest changes, while not replanning frequently unless the changes are very large.
Valid values	dbms.cypher.statistics_divergence_threshold, a double which is in the range `0.0` to `1.0`
Default value	`0.75`

Table 194. dbms.databases.seed_from_uri_providers

Description	Enterprise edition Databases may be created from an existing 'seed' (a database backup or dump) stored at some source URI. Different types of seed source are supported by different implementations of `com.neo4j.dbms.seeding.SeedProvider`. For example, seeds stored at 's3://' and 'https://' URIs are supported by the builtin `S3SeedProvider` and `URLConnectionSeedProvider` respectively. This list specifies enabled seed providers. If a seed source (URI scheme) is supported by multiple providers in the list, the first matching provider will be used. If the list is set to empty, the seed from uri functionality is effectively disabled.

Valid values	dbms.databases.seed_from_uri_providers, a ',' separated list with elements of type 'a string'.
Default value	`S3SeedProvider`

Table 195. dbms.db.timezone

Description	Database timezone. Among other things, this setting influences the monitoring procedures.
Valid values	dbms.db.timezone, one of [UTC, SYSTEM]
Default value	`UTC`

Table 196. dbms.kubernetes.address

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

Table 197. dbms.kubernetes.ca_crt

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

Table 198. dbms.kubernetes.cluster_domain

Description	Enterprise edition Kubernetes cluster domain.
Valid values	dbms.kubernetes.cluster_domain, a string
Default value	`cluster.local`

Table 199. dbms.kubernetes.label_selector

Description	Enterprise edition LabelSelector for Kubernetes API.
Valid values	dbms.kubernetes.label_selector, a string

Table 200. dbms.kubernetes.namespace

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

Table 201. dbms.kubernetes.service_port_name

Description	Enterprise edition Service port name for discovery for Kubernetes API.
Valid values	dbms.kubernetes.service_port_name, a string

Table 202. dbms.kubernetes.token

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

Table 203. dbms.logs.http.enabled

Description	Enable HTTP request logging.
Valid values	dbms.logs.http.enabled, a boolean
Default value	`false`

Table 204. 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 205. dbms.memory.transaction.total.max

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

Valid values	dbms.memory.transaction.total.max, 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 206. dbms.netty.ssl.provider

Description	Netty SSL provider.
Valid values	dbms.netty.ssl.provider, one of [JDK, OPENSSL, OPENSSL_REFCNT]
Default value	`JDK`

Table 207. dbms.routing.client_side.enforce_for_domains

Description	Always use client side routing (regardless of the default router) for neo4j:// protocol connections to these domains. A comma separated list of domains. Wildcards (*) are supported.
Valid values	dbms.routing.client_side.enforce_for_domains, a ',' separated set with elements of type 'a string'.
Dynamic	true
Default value	

Table 208. dbms.routing.default_router

Description	Routing strategy for neo4j:// protocol connections. Default is `CLIENT`, using client-side routing, with server-side routing as a fallback (if enabled). When set to `SERVER`, client-side routing is short-circuited, and requests will rely on server-side routing (which must be enabled for proper operation, i.e. `dbms.routing.enabled=true`). Can be overridden by `dbms.routing.client_side.enforce_for_domains`.
Valid values	dbms.routing.default_router, one of [SERVER, CLIENT]
Default value	`CLIENT`

Table 209. 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 210. 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 211. 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 212. 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 213. 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 214. 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	`INFO`

Table 215. dbms.routing.enabled

Description	Enable server-side routing in clusters using an additional bolt connector. When configured, this allows requests to be forwarded from one cluster member to another, if the requests can't be satisfied by the first member (e.g. write requests received by a non-leader).
Valid values	dbms.routing.enabled, a boolean
Default value	`true`

Table 216. dbms.routing.load_balancing.plugin

Description	Enterprise edition The load balancing plugin to use.
Valid values	dbms.routing.load_balancing.plugin, a string which specified load balancer plugin exist.
Default value	`server_policies`

Table 217. dbms.routing.load_balancing.shuffle_enabled

Description	Enterprise edition Enables shuffling of the returned load balancing result.

Valid values	dbms.routing.load_balancing.shuffle_enabled, a boolean
Default value	`true`

Table 218. dbms.routing.reads_on_primaries_enabled

Description	Enterprise edition Configure if the `dbms.routing.getRoutingTable()` procedure should include non-writer primaries as read endpoints or return only secondaries. Note: if there are no secondaries for the given database primaries are returned as read end points regardless the value of this setting. Defaults to true so that non-writer primaries are available for read-only queries in a typical heterogeneous setup.
Valid values	dbms.routing.reads_on_primaries_enabled, a boolean
Default value	`true`

Table 219. dbms.routing.reads_on_writers_enabled

Description	Enterprise edition Configure if the `dbms.routing.getRoutingTable()` procedure should include the writer as read endpoint or return only non-writers (non writer primaries and secondaries) Note: writer is returned as read endpoint if no other member is present all.
Valid values	dbms.routing.reads_on_writers_enabled, a boolean
Dynamic	true
Default value	`false`

Table 220. 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 221. 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 222. dbms.security.auth_cache_max_capacity

Description	Enterprise edition The maximum capacity for authentication and authorization caches (respectively).
Valid values	dbms.security.auth_cache_max_capacity, an integer
Default value	`10000`

Table 223. dbms.security.auth_cache_ttl

Description	Enterprise edition 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 224. dbms.security.auth_cache_use_ttl

Description	Enterprise edition 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 225. dbms.security.auth_enabled

Description	Enable auth requirement to access Neo4j.
Valid values	dbms.security.auth_enabled, a boolean
Default value	`true`

Table 226. dbms.security.auth_minimum_password_length

Description	Version number The minimum number of characters required in a password.

Valid values	dbms.security.auth_minimum_password_length, an integer
Default value	`8`

Table 227. *dbms.security.auth_lock_time*

Description	The amount of time user account should be locked after a configured number of unsuccessful authentication attempts. The locked out user will not be able to log in until the lock period expires, even if correct credentials are provided. Setting this configuration option to a low value is not recommended because it might make it easier for an attacker to brute force the password.
Valid values	dbms.security.auth_lock_time, 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 228. *dbms.security.auth_max_failed_attempts*

Description	The maximum number of unsuccessful authentication attempts before imposing a user lock for the configured amount of time, 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 229. *dbms.security.authentication_providers*

Description	Enterprise edition 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 230. *dbms.security.authorization_providers*

Description	Enterprise edition 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 231. dbms.security.cluster_status_auth_enabled

Description	Enterprise edition Require authorization for access to the Causal Clustering status endpoints.
Valid values	dbms.security.cluster_status_auth_enabled, a boolean
Default value	`true`

Table 232. dbms.security.http_access_control_allow_origin

Description	Value of the Access-Control-Allow-Origin header sent over any HTTP or HTTPS connector. This defaults to '*', which allows broadest compatibility. Note that any URI provided here limits HTTP/HTTPS access to that URI only.
Valid values	dbms.security.http_access_control_allow_origin, a string
Default value	`*`

Table 233. 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 234. dbms.security.http_strict_transport_security

Description	Value of the HTTP Strict-Transport-Security (HSTS) response header. This header tells browsers that a webpage should only be accessed using HTTPS instead of HTTP. It is attached to every HTTPS response. Setting is not set by default so 'Strict-Transport-Security' header is not sent. Value is expected to contain directives like 'max-age', 'includeSubDomains' and 'preload'.

Valid values	dbms.security.http_strict_transport_security, a string

Table 235. dbms.security.key.name

Description	Enterprise edition Name of the 256 length AES encryption key, which is used for the symmetric encryption.
Valid values	dbms.security.key.name, a string
Dynamic	true
Default value	`aesKey`

Table 236. dbms.security.keystore.password

Description	Enterprise edition Password for accessing the keystore holding a 256 length AES encryption key, which is used for the symmetric encryption.
Valid values	dbms.security.keystore.password, a secure string
Dynamic	true

Table 237. dbms.security.keystore.path

Description	Enterprise edition Location of the keystore holding a 256 length AES encryption key, which is used for the symmetric encryption of secrets held in system database.
Valid values	dbms.security.keystore.path, a path
Dynamic	true

Table 238. dbms.security.ldap.authentication.attribute

Description	Enterprise edition The attribute to use when looking up users. Using this setting requires `dbms.security.ldap.authentication.search_for_attribute` to be true and thus `dbms.security.ldap.authorization.system_username` and `dbms.security.ldap.authorization.system_password` to be configured.
Valid values	dbms.security.ldap.authentication.attribute, a string which matches the pattern `[A-Za-z0-9-]*` (has to be a valid LDAP attribute name, only containing letters [A-Za-z], digits [0-9] and hyphens [-].)
Dynamic	true

Default value	`samaccountname`

Table 239. *dbms.security.ldap.authentication.cache_enabled*

Description	Enterprise edition 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 240. *dbms.security.ldap.authentication.mechanism*

Description	Enterprise edition 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 241. *dbms.security.ldap.authentication.search_for_attribute*

Description	Enterprise edition Perform authentication by searching for an unique attribute of a user. Using this setting requires `dbms.security.ldap.authorization.system_username` and `dbms.security.ldap.authorization.system_password` to be configured.
Valid values	dbms.security.ldap.authentication.search_for_attribute, a boolean
Default value	`false`

Table 242. *dbms.security.ldap.authentication.user_dn_template*

Description	Enterprise edition LDAP user DN template. An LDAP object is referenced by its distinguished name (DN), and a user DN is an LDAP fully-qualified unique user identifier. This setting is used to generate an LDAP DN that conforms with the LDAP directory's schema from the user principal that is submitted with the authentication token when logging in. The special token \{0} is a placeholder where the user principal will be substituted into the DN string.
Valid values	dbms.security.ldap.authentication.user_dn_template, a string which Must be a string containing '\{0}' to understand where to insert the runtime authentication principal.
Dynamic	true
Default value	`uid={0},ou=users,dc=example,dc=com`

Table 243. dbms.security.ldap.authorization.access_permitted_group

Description	Enterprise edition The LDAP group to which a user must belong to get any access to the system.Set this to restrict access to a subset of LDAP users belonging to a particular group. If this is not set, any user to successfully authenticate via LDAP will have access to the PUBLIC role and any other roles assigned to them via dbms.security.ldap.authorization.group_to_role_mapping.
Valid values	dbms.security.ldap.authorization.access_permitted_group, a string
Dynamic	true
Default value	

Table 244. dbms.security.ldap.authorization.group_membership_attributes

Description	Enterprise edition A list of attribute names on a user object that contains groups to be used for mapping to roles when LDAP authorization is enabled. This setting is ignored when `dbms.ldap_authorization_nested_groups_enabled` is `true`.
Valid values	dbms.security.ldap.authorization.group_membership_attributes, a ',' separated list with elements of type 'a string'. which Can not be empty
Dynamic	true
Default value	`memberOf`

Table 245. dbms.security.ldap.authorization.group_to_role_mapping

Description	Enterprise edition 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 which must be semicolon separated list of key-value pairs or empty
Dynamic	true
Default value	

Table 246. dbms.security.ldap.authorization.nested_groups_enabled

Description	Enterprise edition This setting determines whether multiple LDAP search results will be processed (as is required for the lookup of nested groups). If set to `true` then instead of using attributes on the user object to determine group membership (as specified by `dbms.security.ldap.authorization.group_membership_attributes`), the `user` object will only be used to determine the user's Distinguished Name, which will subsequently be used with `dbms.security.ldap.authorization.user_search_filter` in order to perform a nested group search. The Distinguished Names of the resultant group search results will be used to determine roles.
Valid values	dbms.security.ldap.authorization.nested_groups_enabled, a boolean
Dynamic	true
Default value	`false`

Table 247. dbms.security.ldap.authorization.nested_groups_search_filter

Description	Enterprise edition The search template which will be used to find the nested groups which the user is a member of. The filter should contain the placeholder token `\{0}` which will be substituted with the user's Distinguished Name (which is found for the specified user principle using `dbms.security.ldap.authorization.user_search_filter`). The default value specifies Active Directory's LDAP_MATCHING_RULE_IN_CHAIN (aka 1.2.840.113556.1.4.1941) implementation which will walk the ancestry of group membership for the specified user.
Valid values	dbms.security.ldap.authorization.nested_groups_search_filter, a string
Dynamic	true
Default value	`(&(objectclass=group)(member:1.2.840.113556.1.4.1941:={0}))`

Table 248. dbms.security.ldap.authorization.system_password

Description	Enterprise edition 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 249. dbms.security.ldap.authorization.system_username

Description	Enterprise edition 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 250. dbms.security.ldap.authorization.use_system_account

Description	Enterprise edition 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 251. dbms.security.ldap.authorization.user_search_base

Description	Enterprise edition The name of the base object or named context to search for user objects when LDAP authorization is enabled. A common case is that this matches the last part of `dbms.security.ldap.authentication.user_dn_template`.
Valid values	dbms.security.ldap.authorization.user_search_base, a string which Can not be empty
Dynamic	true
Default value	`ou=users,dc=example,dc=com`

Table 252. dbms.security.ldap.authorization.user_search_filter

Description	Enterprise edition 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
Dynamic	true
Default value	`(&(objectClass=*)(uid={0}))`

Table 253. dbms.security.ldap.connection_timeout

Description	Enterprise edition 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 254. dbms.security.ldap.host

Description	Enterprise edition 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 255. dbms.security.ldap.read_timeout

Description	Enterprise edition 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 256. dbms.security.ldap.referral

Description	Enterprise edition 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 257. dbms.security.ldap.use_starttls

Description	Enterprise edition 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 258. dbms.security.log_successful_authentication

Description	Enterprise edition 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 259. dbms.security.oidc.<provider>.audience

Description	Enterprise edition Expected values of the Audience (aud) claim in the id token.
Valid values	dbms.security.oidc.<provider>.audience, a ',' separated list with elements of type 'a string'. which Can not be empty
Dynamic	true

Table 260. dbms.security.oidc.<provider>.auth_endpoint

Description	Enterprise edition The OIDC authorization endpoint. If this is not supplied Neo4j will attempt to discover it from the well_known_discovery_uri.
Valid values	dbms.security.oidc.<provider>.auth_endpoint, a URI
Dynamic	true

Table 261. dbms.security.oidc.<provider>.auth_flow

Description	Enterprise edition The OIDC flow to use. This is exposed to clients via the discovery endpoint. Supported values are `pkce` and `implicit`

Valid values	dbms.security.oidc.<provider>.auth_flow, one of [PKCE, IMPLICIT]
Dynamic	true
Default value	`PKCE`

Table 262. dbms.security.oidc.<provider>.auth_params

Description	Enterprise edition Optional additional parameters that the auth endpoint requires. Please use params instead. The map is a semicolon separated list of key-value pairs. For example: `k1=v1;k2=v2`.
Valid values	dbms.security.oidc.<provider>.auth_params, A simple key value map pattern `k1=v1;k2=v2`.
Dynamic	true
Default value	`{}`
Deprecated	The `dbms.security.oidc.<provider>.auth_params` configuration setting has been deprecated.

Table 263. dbms.security.oidc.<provider>.authorization.group_to_role_mapping

Description	Enterprise edition An authorization mapping from IdP group names to Neo4j role names. The map should be formatted as a semicolon separated list of key-value pairs, where the key is the IdP group name and the value is a comma separated list of corresponding role names. For example: group1=role1;group2=role2;group3=role3,role4,role5 You could also use whitespaces and quotes around group names to make this mapping more readable, for example:
	`dbms.security.oidc.<provider>.authorization.group_to_role_mapping=\`
	`         "Neo4j Read Only"      = reader;    \`
	`         "Neo4j Read-Write"     = publisher; \`
	`         "Neo4j Schema Manager" = architect; \`
	`         "Neo4j Administrator"  = admin`

Valid values	dbms.security.oidc.<provider>.authorization.group_to_role_mapping, a string which must be semicolon separated list of key-value pairs or empty
Dynamic	true

Table 264. dbms.security.oidc.<provider>.claims.groups

Description	Enterprise edition The claim to use as the list of groups in Neo4j. These could be Neo4J roles directly, or can be mapped using dbms.security.oidc.<provider>.authorization.group_to_role_mapping.
Valid values	dbms.security.oidc.<provider>.claims.groups, a string
Dynamic	true

Table 265. dbms.security.oidc.<provider>.claims.username

Description	Enterprise edition The claim to use as the username in Neo4j. This would typically be sub, but in some situations it may be be desirable to use something else such as email.
Valid values	dbms.security.oidc.<provider>.claims.username, a string
Dynamic	true
Default value	`sub`

Table 266. dbms.security.oidc.<provider>.client_id

Description	Enterprise edition Client id needed if token contains multiple Audience (aud) claims.
Valid values	dbms.security.oidc.<provider>.client_id, a string
Dynamic	true

Table 267. dbms.security.oidc.<provider>.config

Description	Enterprise edition The accepted values (all optional) are: • `principal`: in which JWT claim the user's email address is specified, email is the default. This is the value that will be shown in browser. • `code_challenge_method`: default is `S256` and it's the only supported method at this moment. This setting applies only for pkce auth flow • `token_type_principal`: the options are almost always either `access_token`, which is the default, or `id_token`. • `token_type_authentication`: the options are almost always either `access_token`, which is the default, or `id_token`. • `implicit_flow_requires_nonce`: true or false. Defaults to false.

Valid values	dbms.security.oidc.<provider>.config, A simple key value map pattern `k1=v1;k2=v2`. Valid key options are: `[implicit_flow_requires_nonce, token_type_authentication, token_type_principal, principal, code_challenge_method]`.
Dynamic	true
Default value	`{}`

Table 268. dbms.security.logs.oidc.jwt_claims_at_debug_level_enabled

Description	When set to `true`, it logs the claims from the JWT. This will only take effect when the security log level is set to `DEBUG`. WARNING: It is strongly advised that this is set to `false` when running in a production environment in order to prevent logging of sensitive information. Also note that the contents of the JWT claims set can change over time because they are dependent entirely upon the ID provider.
Valid values	dbms.security.logs.oidc.jwt_claims_at_debug_level_enabled, a boolean
Default value	`false`

Table 269. dbms.security.oidc.<provider>.display_name

Description	Enterprise edition The user-facing name of the provider as provided by the discovery endpoint to clients (Bloom, Browser etc.).
Valid values	dbms.security.oidc.<provider>.display_name, a string

Table 270. dbms.security.oidc.<provider>.get_groups_from_user_info

Description	Enterprise edition When turned on, Neo4j gets the groups from the provider user info endpoint.
Valid values	dbms.security.oidc.<provider>.get_groups_from_user_info, a boolean
Dynamic	true
Default value	`false`

Table 271. dbms.security.oidc.<provider>.get_username_from_user_info

Description	Enterprise edition When turned on, Neo4j gets the username from the provider user info endpoint.

Valid values	dbms.security.oidc.<provider>.get_username_from_user_info, a boolean
Dynamic	true
Default value	`false`

Table 272. dbms.security.oidc.<provider>.issuer

Description	Enterprise edition The expected value of the iss claim in the id token. If this is not supplied Neo4j will attempt to discover it from the well_known_discovery_uri.
Valid values	dbms.security.oidc.<provider>.issuer, a string
Dynamic	true

Table 273. dbms.security.oidc.<provider>.jwks_uri

Description	Enterprise edition The location of the JWK public key set for the identity provider. If this is not supplied Neo4j will attempt to discover it from the well_known_discovery_uri.
Valid values	dbms.security.oidc.<provider>.jwks_uri, a URI
Dynamic	true

Table 274. dbms.security.oidc.<provider>.params

Description	Enterprise edition The map is a semicolon separated list of key-value pairs. For example: `k1=v1;k2=v2`. The user should at least provide:
	`client_id: the SSO Idp client idenfifier.`
	`response_type: code if auth_flow is pkce or token for implicit auth_flow.`
	`scope: often containing a subset of 'email profile openid groups'.`
	For example: `client_id=my-client-id;response_type=code;scope=openid profile email`.

Valid values	dbms.security.oidc.<provider>.params, A simple key value map pattern `k1=v1;k2=v2`. Required key options are: `[scope, client_id, response_type]`.
Dynamic	true
Default value	`{}`

Table 275. dbms.security.oidc.<provider>.token_endpoint

Description	Enterprise edition The OIDC token endpoint. If this is not supplied Neo4j will attempt to discover it from the well_known_discovery_uri.
Valid values	dbms.security.oidc.<provider>.token_endpoint, a URI
Dynamic	true

Table 276. *dbms.security.oidc.<provider>.token_params*

Description	Enterprise edition Optional query parameters that the token endpoint requires. The map is a semicolon separated list of key-value pairs. For example: `k1=v1;k2=v2`.If the token endpoint requires a client_secret then this parameter should contain `client_secret=super-secret`
Valid values	dbms.security.oidc.<provider>.token_params, A simple key value map pattern `k1=v1;k2=v2`.
Dynamic	true
Default value	`{}`

Table 277. *dbms.security.oidc.<provider>.user_info_uri*

Description	Enterprise edition The identity providers user info uri.
Valid values	dbms.security.oidc.<provider>.user_info_uri, a URI
Dynamic	true

Table 278. *dbms.security.oidc.<provider>.well_known_discovery_uri*

Description	Enterprise edition The 'well known' OpenID Connect Discovery endpoint used to fetch identity provider settings. If not provided, `issuer`, `jwks_uri`, `auth_endpoint` should be present. If the auth_flow is pkce, `token_endpoint` should also be provided.
Valid values	dbms.security.oidc.<provider>.well_known_discovery_uri, a URI
Dynamic	true

Table 279. *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 280. 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 281. initial.dbms.database_allocator

Description	Enterprise edition Name of the initial database allocator. After the creation of the dbms it can be set with the 'dbms.setDatabaseAllocator' procedure.
Valid values	initial.dbms.database_allocator, a string
Default value	`EQUAL_NUMBERS`

Table 282. initial.dbms.default_database

Description	Name of the default database (aliases are not supported).
Valid values	initial.dbms.default_database, A valid database name containing only alphabetic characters, numbers, dots and dashes with a length between 3 and 63 characters, starting with an alphabetic character but not with the name 'system'
Default value	`neo4j`

Table 283. initial.dbms.default_primaries_count

Description	Enterprise edition Initial default number of primary instances of user databases. If the user does not specify the number of primaries in 'CREATE DATABASE', this value will be used, unless it is overwritten with the 'dbms.setDefaultAllocationNumbers' procedure.

Valid values	initial.dbms.default_primaries_count, an integer which is minimum `1` and is maximum `11`. The same value applies to runtime max number.
Default value	`1`

Table 284. *initial.dbms.default_secondaries_count*

Description	Enterprise edition Initial default number of secondary instances of user databases. If the user does not specify the number of secondaries in 'CREATE DATABASE', this value will be used, unless it is overwritten with the 'dbms.setDefaultAllocationNumbers' procedure.
Valid values	initial.dbms.default_secondaries_count, an integer which is minimum `0` and is maximum `20`. The same value applies to runtime max number.
Default value	`0`

Table 285. *initial.server.allowed_databases*

Description	Enterprise edition The names of databases that are allowed on this server - all others are denied. Empty means all are allowed. Can be overridden when enabling the server, or altered at runtime, without changing this setting. Exclusive with 'server.initial_denied_databases'
Valid values	initial.server.allowed_databases, a ',' separated set with elements of type 'a string'.
Default value	

Table 286. *initial.server.denied_databases*

Description	Enterprise edition The names of databases that are not allowed on this server. Empty means nothing is denied. Can be overridden when enabling the server, or altered at runtime, without changing this setting. Exclusive with 'server.initial_allowed_databases'
Valid values	initial.server.denied_databases, a ',' separated set with elements of type 'a string'.
Default value	

Table 287. *initial.server.mode_constraint*

Description	Enterprise edition An instance can restrict itself to allow databases to be hosted only as primaries or secondaries. This setting is the default input for the `ENABLE SERVER` command - the user can overwrite it when executing the procedure.
Valid values	initial.server.mode_constraint, one of [PRIMARY, SECONDARY, NONE]

Default value	NONE

Table 288. *server.backup.enabled*

Description	Enterprise edition Enable support for running online backups.
Valid values	server.backup.enabled, a boolean
Default value	true

Table 289. *server.backup.listen_address*

Description	Enterprise edition Network interface and port for the backup server to listen on.
Valid values	server.backup.listen_address, a socket address in the format 'hostname:port', 'hostname' or ':port'
Default value	127.0.0.1:6362

Table 290. *server.backup.store_copy_max_retry_time_per_request*

Description	Enterprise edition 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	server.backup.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 291. *server.bolt.advertised_address*

Description	Advertised address for this connector.
Valid values	server.bolt.advertised_address, a socket address in the format 'hostname:port', 'hostname' or ':port' which accessible address. If missing port or hostname it is acquired from server.default_advertised_address
Default value	:7687

Table 292. *server.bolt.connection_keep_alive*

Description	The maximum time to wait before sending a NOOP on connections waiting for responses from active ongoing queries.The minimum value is 1 millisecond.

Valid values	server.bolt.connection_keep_alive, a duration (Valid units are: `ns`, `μs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) which is minimum `1ms`
Default value	`1m`

Table 293. server.bolt.connection_keep_alive_for_requests

Description	The type of messages to enable keep-alive messages for (ALL, STREAMING or OFF)
Valid values	server.bolt.connection_keep_alive_for_requests, one of [ALL, STREAMING, OFF]
Default value	`STREAMING`

Table 294. server.bolt.connection_keep_alive_probes

Description	The total amount of probes to be missed before a connection is considered stale.The minimum for this value is 1.
Valid values	server.bolt.connection_keep_alive_probes, an integer which is minimum `1`
Default value	`2`

Table 295. server.bolt.connection_keep_alive_streaming_scheduling_interval

Description	The interval between every scheduled keep-alive check on all connections with active queries. Zero duration turns off keep-alive service.
Valid values	server.bolt.connection_keep_alive_streaming_scheduling_interval, a duration (Valid units are: `ns`, `μs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) which is minimum `0s`
Default value	`1m`

Table 296. server.bolt.enabled

Description	Enable the bolt connector.
Valid values	server.bolt.enabled, a boolean
Default value	`true`

Table 297. server.bolt.listen_address

Description	Address the connector should bind to.

Valid values	server.bolt.listen_address, a socket address in the format 'hostname:port', 'hostname' or ':port'. If missing port or hostname it is acquired from server.default_listen_address
Default value	`:7687`

Table 298. server.bolt.ocsp_stapling_enabled

Description	Enable server OCSP stapling for bolt and http connectors.
Valid values	server.bolt.ocsp_stapling_enabled, a boolean
Default value	`false`

Table 299. server.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	server.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 300. server.bolt.thread_pool_max_size

Description	The maximum number of threads allowed in the thread pool bound to this connector.
Valid values	server.bolt.thread_pool_max_size, an integer
Default value	`400`

Table 301. server.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	server.bolt.thread_pool_min_size, an integer
Default value	`5`

Table 302. server.bolt.tls_level

Description	Encryption level to require this connector to use.

Valid values	server.bolt.tls_level, one of [REQUIRED, OPTIONAL, DISABLED]
Default value	`DISABLED`

Table 303. server.cluster.advertised_address

Description	Enterprise edition Advertised hostname/IP address and port for the transaction shipping server.
Valid values	server.cluster.advertised_address, a socket address in the format 'hostname:port', 'hostname' or ':port' which accessible address. If missing port or hostname it is acquired from server.default_advertised_address
Default value	`:6000`

Table 304. server.cluster.catchup.connect_randomly_to_server_group

Description	Enterprise edition 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 (`server.cluster.catchup.upstream_strategy`) includes the value `connect-randomly-to-server-group`.
Valid values	server.cluster.catchup.connect_randomly_to_server_group, a ',' separated list with elements of type 'a string identifying a Server Tag'.
Dynamic	true
Default value	

Table 305. server.cluster.catchup.upstream_strategy

Description	Enterprise edition An ordered list in descending preference of the strategy which secondaries use to choose the upstream server from which to pull transactional updates. If none are valid or the list is empty, there is a default strategy of `typically-connect-to-random-secondary`.
Valid values	server.cluster.catchup.upstream_strategy, a ',' separated list with elements of type 'a string'.
Default value	

Table 306. server.cluster.catchup.user_defined_upstream_strategy

Description	Enterprise edition Configuration of a user-defined upstream selection strategy. The user-defined strategy is used if the list of strategies (`server.cluster.catchup.upstream_strategy`) includes the value `user_defined`.
Valid values	server.cluster.catchup.user_defined_upstream_strategy, a string
Default value	

Table 307. server.cluster.listen_address

Description	Enterprise edition 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	server.cluster.listen_address, a socket address in the format 'hostname:port', 'hostname' or ':port'. If missing port or hostname it is acquired from server.default_listen_address
Default value	`:6000`

Table 308. server.cluster.network.native_transport_enabled

Description	Enterprise edition Use native transport if available. Epoll for Linux or Kqueue for MacOS/BSD. If this setting is set to false, or if native transport is not available, Nio transport will be used.
Valid values	server.cluster.network.native_transport_enabled, a boolean
Default value	`true`

Table 309. server.cluster.raft.advertised_address

Description	Enterprise edition Advertised hostname/IP address and port for the RAFT server.
Valid values	server.cluster.raft.advertised_address, a socket address in the format 'hostname:port', 'hostname' or ':port' which accessible address. If missing port or hostname it is acquired from server.default_advertised_address
Default value	`:7000`

Table 310. server.cluster.raft.listen_address

Description	Enterprise edition Network interface and port for the RAFT server to listen on.

Valid values	server.cluster.raft.listen_address, a socket address in the format 'hostname:port', 'hostname' or ':port'. If missing port or hostname it is acquired from server.default_listen_address
Default value	`:7000`

Table 311. server.cluster.system_database_mode

Description	Enterprise edition Users must manually specify the mode for the system database on each instance.
Valid values	server.cluster.system_database_mode, one of [PRIMARY, SECONDARY]
Default value	`PRIMARY`

Table 312. server.config.strict_validation.enabled

Description	A strict configuration validation will prevent the database from starting up if unknown configuration options are specified in the neo4j settings namespace (such as dbms., cypher., etc) or if settings are declared multiple times.
Valid values	server.config.strict_validation.enabled, a boolean
Default value	`true`

Table 313. server.databases.default_to_read_only

Description	Whether or not any database on this instance are read_only by default. If false, individual databases may be marked as read_only using server.database.read_only. If true, individual databases may be marked as writable using server.databases.writable.
Valid values	server.databases.default_to_read_only, a boolean
Dynamic	true
Default value	`false`

Table 314. server.databases.read_only

Description	List of databases for which to prevent write queries. Databases not included in this list maybe read_only anyway depending upon the value of server.databases.default_to_read_only.

Valid values	server.databases.read_only, a ',' separated set with elements of type 'A valid database name containing only alphabetic characters, numbers, dots and dashes with a length between 3 and 63 characters, starting with an alphabetic character but not with the name 'system''. which Value 'system' can't be included in read only databases collection!
Dynamic	true
Default value	

Table 315. server.databases.writable

Description	List of databases for which to allow write queries. Databases not included in this list will allow write queries anyway, unless server.databases.default_to_read_only is set to true.
Valid values	server.databases.writable, a ',' separated set with elements of type 'A valid database name containing only alphabetic characters, numbers, dots and dashes with a length between 3 and 63 characters, starting with an alphabetic character but not with the name 'system''.
Dynamic	true
Default value	

Table 316. server.db.query_cache_size

Description	The number of cached Cypher query execution plans per database. The max number of query plans that can be kept in cache is the `number of databases` * `server.db.query_cache_size`. With 10 databases and `server.db.query_cache_size`=1000, the caches can keep 10000 plans in total on the instance, assuming that each DB receives queries that fill up its cache.
Valid values	server.db.query_cache_size, an integer which is minimum `0`
Default value	`1000`

Table 317. server.default_advertised_address

Description	Default hostname or IP address the server uses to advertise itself.
Valid values	server.default_advertised_address, a socket address in the format 'hostname:port', 'hostname' or ':port' which has no specified port and accessible address
Default value	`localhost`

Table 318. server.default_listen_address

Description	Default network interface to listen for incoming connections. To listen for connections on all interfaces, use "0.0.0.0".
Valid values	server.default_listen_address, a socket address in the format 'hostname:port', 'hostname' or ':port' which has no specified port
Default value	`localhost`

Table 319. server.directories.cluster_state

Description	Enterprise edition Directory to hold cluster state including Raft log.
Valid values	server.directories.cluster_state, a path. If relative it is resolved from server.directories.data
Default value	`cluster-state`

Table 320. server.directories.data

Description	Path of the data directory. You must not configure more than one Neo4j installation to use the same data directory.
Valid values	server.directories.data, a path. If relative it is resolved from server.directories.neo4j_home
Default value	`data`

Table 321. server.directories.dumps.root

Description	Root location where Neo4j will store database dumps optionally produced when dropping said databases.
Valid values	server.directories.dumps.root, a path. If relative it is resolved from server.directories.data
Default value	`dumps`

Table 322. server.directories.import

Description	Sets the root directory for file URLs used with the Cypher `LOAD CSV` clause. This should be set to a directory relative to the Neo4j installation path, restricting access to only those files within that directory and its subdirectories. For example the value "import" will only enable access to files within the 'import' folder. Removing this setting will disable the security feature, allowing all files in the local system to be imported. Setting this to an empty field will allow access to all files within the Neo4j installation folder.
Valid values	server.directories.import, a path. If relative it is resolved from server.directories.neo4j_home

Table 323. server.directories.lib

Description	Path of the lib directory.
Valid values	server.directories.lib, a path. If relative it is resolved from server.directories.neo4j_home
Default value	`lib`

Table 324. server.directories.licenses

Description	Path of the licenses directory.
Valid values	server.directories.licenses, a path. If relative it is resolved from server.directories.neo4j_home
Default value	`licenses`

Table 325. server.directories.logs

Description	Path of the logs directory.
Valid values	server.directories.logs, a path. If relative it is resolved from server.directories.neo4j_home
Default value	`logs`

Table 326. server.directories.metrics

Description	Enterprise edition The target location of the CSV files: a path to a directory wherein a CSV file per reported field will be written.
Valid values	server.directories.metrics, a path. If relative it is resolved from server.directories.neo4j_home

Default value	`metrics`

Table 327. server.directories.neo4j_home

Description	Root relative to which directory settings are resolved. Calculated and set by the server on startup.
Valid values	server.directories.neo4j_home, a path which is absolute
Default value	`Defaults to current working directory`

Table 328. server.directories.plugins

Description	Location of the database plugin directory. Compiled Java JAR files that contain database procedures will be loaded if they are placed in this directory.
Valid values	server.directories.plugins, a path. If relative it is resolved from server.directories.neo4j_home
Default value	`plugins`

Table 329. server.directories.run

Description	Path of the run directory. This directory holds Neo4j's runtime state, such as a pidfile when it is running in the background. The pidfile is created when starting neo4j and removed when stopping it. It may be placed on an in-memory filesystem such as tmpfs.
Valid values	server.directories.run, a path. If relative it is resolved from server.directories.neo4j_home
Default value	`run`

Table 330. server.directories.script.root

Description	Root location where Neo4j will store scripts for configured databases.
Valid values	server.directories.script.root, a path. If relative it is resolved from server.directories.data
Default value	`scripts`

Table 331. server.directories.transaction.logs.root

Description	Root location where Neo4j will store transaction logs for configured databases.

Valid values	server.directories.transaction.logs.root, a path. If relative it is resolved from server.directories.data
Default value	`transactions`

Table 332. server.discovery.advertised_address

Description	Enterprise edition Advertised cluster member discovery management communication.
Valid values	server.discovery.advertised_address, a socket address in the format 'hostname:port', 'hostname' or ':port' which accessible address. If missing port or hostname it is acquired from server.default_advertised_address
Default value	`:5000`

Table 333. server.discovery.listen_address

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

Table 334. server.dynamic.setting.allowlist

Description	Enterprise edition 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	server.dynamic.setting.allowlist, a ',' separated list with elements of type 'a string'.
Default value	`*`

Table 335. server.groups

Description	Enterprise edition A list of tag names for the server used when configuring load balancing and replication policies.
Valid values	server.groups, a ',' separated list with elements of type 'a string identifying a Server Tag'.

Dynamic	true
Default value	

Table 336. server.http.advertised_address

Description	Advertised address for this connector.
Valid values	server.http.advertised_address, a socket address in the format 'hostname:port', 'hostname' or ':port' which accessible address. If missing port or hostname it is acquired from server.default_advertised_address
Default value	`:7474`

Table 337. server.http.enabled

Description	Enable the http connector.
Valid values	server.http.enabled, a boolean
Default value	`true`

Table 338. server.http.listen_address

Description	Address the connector should bind to.
Valid values	server.http.listen_address, a socket address in the format 'hostname:port', 'hostname' or ':port'. If missing port or hostname it is acquired from server.default_listen_address
Default value	`:7474`

Table 339. server.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	server.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 340. server.https.advertised_address

Description	Advertised address for this connector.
Valid values	server.https.advertised_address, a socket address in the format 'hostname:port', 'hostname' or ':port' which accessible address. If missing port or hostname it is acquired from server.default_advertised_address
Default value	`:7473`

Table 341. server.https.enabled

Description	Enable the https connector.
Valid values	server.https.enabled, a boolean
Default value	`false`

Table 342. server.https.listen_address

Description	Address the connector should bind to.
Valid values	server.https.listen_address, a socket address in the format 'hostname:port', 'hostname' or ':port'. If missing port or hostname it is acquired from server.default_listen_address
Default value	`:7473`

Table 343. server.jvm.additional

Description	Additional JVM arguments. Argument order can be significant. To use a Java commercial feature, the argument to unlock commercial features must precede the argument to enable the specific feature in the config value string. For example, to use Flight Recorder, `-XX:+UnlockCommercialFeatures` must come before `-XX:+FlightRecorder`.
Valid values	server.jvm.additional, one or more jvm arguments

Table 344. server.logs.config

Description	Path to the logging configuration for debug, query, http and security logs.
Valid values	server.logs.config, a path. If relative it is resolved from server.directories.neo4j_home
Default value	`conf/server-logs.xml`

Table 345. server.logs.debug.enabled

Description	Enable the debug log.
Valid values	server.logs.debug.enabled, a boolean
Default value	`true`

Table 346. server.logs.gc.enabled

Description	Enable GC Logging.
Valid values	server.logs.gc.enabled, a boolean
Default value	`false`

Table 347. server.logs.gc.options

Description	GC Logging Options.
Valid values	server.logs.gc.options, a string
Default value	`-Xlog:gc*,safepoint,age*=trace`

Table 348. server.logs.gc.rotation.keep_number

Description	Number of GC logs to keep.
Valid values	server.logs.gc.rotation.keep_number, an integer
Default value	`5`

Table 349. server.logs.gc.rotation.size

Description	Size of each GC log that is kept.
Valid values	server.logs.gc.rotation.size, a byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `TiB`, `TB`, `PiB`, `PB`, `EiB`, `EB`)
Default value	`20.00MiB`

Table 350. server.logs.user.config

Description	Path to the logging configuration of user logs.
Valid values	server.logs.user.config, a path. If relative it is resolved from server.directories.neo4j_home

Default value	`conf/user-logs.xml`

Table 351. *server.max_databases*

Description	Enterprise edition The maximum number of databases.
Valid values	server.max_databases, a long which is minimum `2`
Default value	`100`

Table 352. *server.memory.heap.initial_size*

Description	Initial heap size. By default it is calculated based on available system resources.
Valid values	server.memory.heap.initial_size, a byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `TiB`, `TB`, `PiB`, `PB`, `EiB`, `EB`)

Table 353. *server.memory.heap.max_size*

Description	Maximum heap size. By default it is calculated based on available system resources.
Valid values	server.memory.heap.max_size, a byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `TiB`, `TB`, `PiB`, `PB`, `EiB`, `EB`)

Table 354. *server.memory.off_heap.block_cache_size*

Description	Defines the size of the off-heap memory blocks cache. The cache will contain this number of blocks for each block size that is power of two. Thus, maximum amount of memory used by blocks cache can be calculated as 2 * server.memory.off_heap.max_cacheable_block_size * `server.memory.off_heap.block_cache_size`
Valid values	server.memory.off_heap.block_cache_size, an integer which is minimum `16`
Default value	`128`

Table 355. *server.memory.off_heap.max_cacheable_block_size*

Description	Defines the maximum size of an off-heap memory block that can be cached to speed up allocations. The value must be a power of 2.
Valid values	server.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 356. server.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 db.tx_state.memory_allocation is set to 'OFF_HEAP'.
Valid values	server.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 357. server.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	server.memory.pagecache.directio, a boolean
Default value	`false`

Table 358. server.memory.pagecache.flush.buffer.enabled

Description	Page cache can be configured to use a temporal buffer for flushing purposes. It is used to combine, if possible, sequence of several cache pages into one bigger buffer to minimize the number of individual IOPS performed and better utilization of available I/O resources, especially when those are restricted.
Valid values	server.memory.pagecache.flush.buffer.enabled, a boolean
Dynamic	true
Default value	`false`

Table 359. server.memory.pagecache.flush.buffer.size_in_pages

Description	Page cache can be configured to use a temporal buffer for flushing purposes. It is used to combine, if possible, sequence of several cache pages into one bigger buffer to minimize the number of individual IOPS performed and better utilization of available I/O resources, especially when those are restricted. Use this setting to configure individual file flush buffer size in pages (8KiB). To be able to utilize this buffer during page cache flushing, buffered flush should be enabled.
Valid values	server.memory.pagecache.flush.buffer.size_in_pages, an integer which is in the range `1` to `512`

Dynamic	true
Default value	`128`

Table 360. *server.memory.pagecache.scan.prefetchers*

Description	The maximum number of worker threads to use for pre-fetching data when doing sequential scans. Set to '0' to disable pre-fetching for scans.
Valid values	server.memory.pagecache.scan.prefetchers, an integer which is in the range `0` to `255`
Default value	`4`

Table 361. *server.memory.pagecache.size*

Description	The amount of memory to use for mapping the store files. If Neo4j is running on a dedicated server, then it is generally recommended to leave about 2-4 gigabytes for the operating system, give the JVM enough heap to hold all your transaction state and query context, and then leave the rest for the page cache. If no page cache memory is configured, then a heuristic setting is computed based on available system resources.
Valid values	server.memory.pagecache.size, a byte size (valid multipliers are `B`, `KiB`, `KB`, `K`, `kB`, `kb`, `k`, `MiB`, `MB`, `M`, `mB`, `mb`, `m`, `GiB`, `GB`, `G`, `gB`, `gb`, `g`, `TiB`, `TB`, `PiB`, `PB`, `EiB`, `EB`)
Default value	`By default the size of page cache will be 50% och available RAM minus the max heap size.The size of the page cache will also not be larger than 70x the max heap size (due to some overhead of the page cache in the heap.`

Table 362. *server.metrics.csv.enabled*

Description	Enterprise edition Set to `true` to enable exporting metrics to CSV files.
Valid values	server.metrics.csv.enabled, a boolean
Default value	`true`

Table 363. *server.metrics.csv.interval*

Description	Enterprise edition The reporting interval for the CSV files. That is, how often new rows with numbers are appended to the CSV files.
Valid values	server.metrics.csv.interval, a duration (Valid units are: `ns`, `μs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`) which is minimum `1ms`
Default value	`30s`

Table 364. server.metrics.csv.rotation.compression

Description	Enterprise edition Decides what compression to use for the csv history files.
Valid values	server.metrics.csv.rotation.compression, one of [NONE, ZIP, GZ]
Default value	`NONE`

Table 365. server.metrics.csv.rotation.keep_number

Description	Enterprise edition Maximum number of history files for the csv files.
Valid values	server.metrics.csv.rotation.keep_number, an integer which is minimum `1`
Default value	`7`

Table 366. server.metrics.csv.rotation.size

Description	Enterprise edition 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	server.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 367. server.metrics.enabled

Description	Enterprise edition Enable metrics. Setting this to `false` will to turn off all metrics.
Valid values	server.metrics.enabled, a boolean
Default value	`true`

Table 368. server.metrics.filter

Description	Enterprise edition 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	server.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,*cluster.core.is_leader,*cluster.core.last_leader_message,*cluster.core.replication_attempt,*cluster.core.replication_fail,*cluster.core.last_applied,*cluster.core.last_appended,*check_point.duration,*check_point.total_time,*cypher.replan_events,*ids_in_use*,*pool.transaction.*.total_used,*pool.transaction.*.used_heap,*pool.transaction.*.used_native,*store.size*,*transaction.active_read,*transaction.active_write,*transaction.committed*,*transaction.last_committed_tx_id,*transaction.peak_concurrent,*transaction.rollbacks*,*page_cache.hit*,*page_cache.page_faults,*page_cache.usage_ratio,*vm.file.descriptors.count,*vm.gc.time.*,*vm.heap.used,*vm.memory.buffer.direct.used,*vm.memory.pool.g1_eden_space,*vm.memory.pool.g1_old_gen,*vm.pause_time,*vm.thread*,*db.query.execution*`

Table 369. server.metrics.graphite.enabled

Description	Enterprise edition Set to `true` to enable exporting metrics to Graphite.
Valid values	server.metrics.graphite.enabled, a boolean
Default value	`false`

Table 370. server.metrics.graphite.interval

Description	Enterprise edition The reporting interval for Graphite. That is, how often to send updated metrics to Graphite.
Valid values	server.metrics.graphite.interval, a duration (Valid units are: `ns`, `μs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`)
Default value	`30s`

Table 371. server.metrics.graphite.server

Description	Enterprise edition The hostname or IP address of the Graphite server.
Valid values	server.metrics.graphite.server, a socket address in the format 'hostname:port', 'hostname' or ':port'. If missing port or hostname it is acquired from server.default_listen_address
Default value	`:2003`

Table 372. server.metrics.jmx.enabled

Description	Enterprise edition Set to `true` to enable the JMX metrics endpoint.
Valid values	server.metrics.jmx.enabled, a boolean
Default value	`true`

Table 373. server.metrics.prefix

Description	Enterprise edition A common prefix for the reported metrics field names.
Valid values	server.metrics.prefix, a string
Default value	`neo4j`

Table 374. server.metrics.prometheus.enabled

Description	Enterprise edition Set to `true` to enable the Prometheus endpoint.
Valid values	server.metrics.prometheus.enabled, a boolean
Default value	`false`

Table 375. server.metrics.prometheus.endpoint

Description	Enterprise edition The hostname and port to use as Prometheus endpoint.
Valid values	server.metrics.prometheus.endpoint, a socket address in the format 'hostname:port', 'hostname' or ':port'. If missing port or hostname it is acquired from server.default_listen_address
Default value	`localhost:2004`

Table 376. server.panic.shutdown_on_panic

Description	Enterprise edition If there is a Database Management System Panic (an irrecoverable error) should the neo4j process shut down or continue running. Following a DbMS panic it is likely that a significant amount of functionality will be lost. Recovering full functionality will require a Neo4j restart. This feature is available in Neo4j Enterprise Edition.
Valid values	server.panic.shutdown_on_panic, a boolean
Default value	`false except for Neo4j Enterprise Edition deployments running on Kubernetes where it is true.`

Table 377. server.routing.advertised_address

Description	Enterprise edition The advertised address for the intra-cluster routing connector.
Valid values	server.routing.advertised_address, a socket address in the format 'hostname:port', 'hostname' or ':port' which accessible address. If missing port or hostname it is acquired from server.default_advertised_address
Default value	`:7688`

Table 378. server.routing.listen_address

Description	The address the routing connector should bind to.
Valid values	server.routing.listen_address, a socket address in the format 'hostname:port', 'hostname' or ':port'. If missing port or hostname it is acquired from server.default_listen_address
Default value	`:7688`

Table 379. server.threads.worker_count

Description	Number of Neo4j worker threads. This setting is only valid for REST, and does not influence bolt-server. It sets the 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	server.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 380. server.unmanaged_extension_classes

Description	Comma-separated list of <classname>=<mount point> for unmanaged extensions.
Valid values	server.unmanaged_extension_classes, a ',' separated list with elements of type '<classname>=<mount point> string'.
Default value	

Table 381. server.windows_service_name

Description	Name of the Windows Service managing Neo4j when installed using `neo4j install-service`. Only applicable on Windows OS. Note: This must be unique for each individual installation.
Valid values	server.windows_service_name, a string
Default value	`neo4j`

Dynamic configuration settings

This page provides a complete reference to the Neo4j dynamic configuration settings, which can be changed at runtime, without restarting the service. This complete reference is a sub-list of all the Neo4j configuration settings.

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

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

For more information on how to update dynamic configuration settings, see Update dynamic settings.

Table 382. Dynamic settings reference

Name	Description
db.checkpoint.iops.limit	Limit the number of IOs the background checkpoint process will consume per second.
db.format	Database format.
db.lock.acquisition.timeout	The maximum time interval within which lock should be acquired.
db.logs.query.early_raw_logging_enabled	Log query text and parameters without obfuscating passwords.
db.logs.query.enabled	Log executed queries.
db.logs.query.max_parameter_length	Sets a maximum character length use for each parameter in the log.
db.logs.query.obfuscate_literals	Obfuscates all literals of the query before writing to the log.
db.logs.query.parameter_logging_enabled	Log parameters for the executed queries being logged.
db.logs.query.plan_description_enabled	Log query plan description table, useful for debugging purposes.
db.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.
db.logs.query.transaction.enabled	Log the start and end of a transaction.
db.logs.query.transaction.threshold	If the transaction is open for more time than this threshold, the transaction is logged once completed - provided transaction logging (db.logs.query.transaction.enabled) is set to `INFO`.
db.memory.transaction.max	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').
db.memory.transaction.total.max	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').
db.track_query_cpu_time	Enables or disables tracking of how much time a query spends actively executing on the CPU.

Name	Description
db.transaction.bookmark_ready_timeout	The maximum amount of time to wait for the database state represented by the bookmark.
db.transaction.concurrent.maximum	The maximum number of concurrently running transactions.
db.transaction.sampling.percentage	Transaction sampling percentage.
db.transaction.timeout	The maximum time interval of a transaction within which it should be completed.
db.transaction.tracing.level	Transaction creation tracing level.
db.tx_log.preallocate	Specify if Neo4j should try to preallocate logical log file in advance.
db.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.
db.tx_log.rotation.size	Specifies at which file size the logical log will auto-rotate.
dbms.cypher.render_plan_description	If set to `true` a textual representation of the plan description will be rendered on the server for all queries running with `EXPLAIN` or `PROFILE`.
dbms.memory.transaction.total.max	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.routing.client_side.enforce_for_domains	Always use client side routing (regardless of the default router) for neo4j:// protocol connections to these domains.
dbms.routing.reads_on_writers_enabled	Enterprise edition Configure if the `dbms.routing.getRoutingTable()` procedure should include the writer as read endpoint or return only non-writers (non writer primaries and secondaries) Note: writer is returned as read endpoint if no other member is present all.
dbms.security.key.name	Enterprise edition Name of the 256 length AES encryption key, which is used for the symmetric encryption.
dbms.security.keystore.password	Enterprise edition Password for accessing the keystore holding a 256 length AES encryption key, which is used for the symmetric encryption.
dbms.security.keystore.path	Enterprise edition Location of the keystore holding a 256 length AES encryption key, which is used for the symmetric encryption of secrets held in system database.

Name	Description
dbms.security.ldap.authentication.attribute	Enterprise edition The attribute to use when looking up users. Using this setting requires `dbms.security.ldap.authentication.search_for_attribute` to be true and thus `dbms.security.ldap.authorization.system_username` and `dbms.security.ldap.authorization.system_password` to be configured.
dbms.security.ldap.authentication.user_dn_template	Enterprise edition LDAP user DN template.
dbms.security.ldap.authorization.access_permitted_group	Enterprise edition The LDAP group to which a user must belong to get any access to the system.Set this to restrict access to a subset of LDAP users belonging to a particular group.
dbms.security.ldap.authorization.group_membership_attributes	Enterprise edition 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	Enterprise edition An authorization mapping from LDAP group names to Neo4j role names.
dbms.security.ldap.authorization.nested_groups_enabled	Enterprise edition This setting determines whether multiple LDAP search results will be processed (as is required for the lookup of nested groups).
dbms.security.ldap.authorization.nested_groups_search_filter	Enterprise edition The search template which will be used to find the nested groups which the user is a member of.
dbms.security.ldap.authorization.user_search_base	Enterprise edition 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	Enterprise edition The LDAP search filter to search for a user principal when LDAP authorization is enabled.
dbms.security.oidc.<provider>.audience	Enterprise edition Expected values of the Audience (aud) claim in the id token.
dbms.security.oidc.<provider>.auth_endpoint	Enterprise edition The OIDC authorization endpoint.
dbms.security.oidc.<provider>.auth_flow	Enterprise edition The OIDC flow to use.
dbms.security.oidc.<provider>.auth_params	Enterprise edition Optional additional parameters that the auth endpoint requires.
dbms.security.oidc.<provider>.authorization.group_to_role_mapping	Enterprise edition An authorization mapping from IdP group names to Neo4j role names.
dbms.security.oidc.<provider>.claims.groups	Enterprise edition The claim to use as the list of groups in Neo4j.
dbms.security.oidc.<provider>.claims.username	Enterprise edition The claim to use as the username in Neo4j.
dbms.security.oidc.<provider>.client_id	Enterprise edition Client id needed if token contains multiple Audience (aud) claims.

Name	Description
dbms.security.oidc.<provider>.config	Enterprise edition The accepted values (all optional) are: ---- principal: in which JWT claim the user's email address is specified, email is the default.
dbms.security.oidc.<provider>.get_groups_from_user_info	Enterprise edition When turned on, Neo4j gets the groups from the provider user info endpoint.
dbms.security.oidc.<provider>.get_username_from_user_info	Enterprise edition When turned on, Neo4j gets the username from the provider user info endpoint.
dbms.security.oidc.<provider>.issuer	Enterprise edition The expected value of the iss claim in the id token.
dbms.security.oidc.<provider>.jwks_uri	Enterprise edition The location of the JWK public key set for the identity provider.
dbms.security.oidc.<provider>.params	Enterprise edition The map is a semicolon separated list of key-value pairs.
dbms.security.oidc.<provider>.token_endpoint	Enterprise edition The OIDC token endpoint.
dbms.security.oidc.<provider>.token_params	Enterprise edition Optional query parameters that the token endpoint requires.
dbms.security.oidc.<provider>.user_info_uri	Enterprise edition The identity providers user info uri.
dbms.security.oidc.<provider>.well_known_discovery_uri	Enterprise edition The 'well known' OpenID Connect Discovery endpoint used to fetch identity provider settings.
server.cluster.catchup.connect_randomly_to_server_group	Enterprise edition Comma separated list of groups to be used by the connect-randomly-to-server-group selection strategy.
server.databases.default_to_read_only	Whether or not any database on this instance are read_only by default.
server.databases.read_only	List of databases for which to prevent write queries.
server.databases.writable	List of databases for which to allow write queries.
server.groups	Enterprise edition A list of tag names for the server used when configuring load balancing and replication policies.
server.memory.pagecache.flush.buffer.enabled	Page cache can be configured to use a temporal buffer for flushing purposes.
server.memory.pagecache.flush.buffer.size_in_pages	Page cache can be configured to use a temporal buffer for flushing purposes.

Table 383. db.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	db.checkpoint.iops.limit, an integer
Dynamic	true
Default value	`600`

Table 384. db.format

Description	Database format. This is the format that will be used for new databases. Valid values are `standard`, `aligned`, or `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.
Valid values	db.format, a string
Dynamic	true
Default value	`aligned`

Table 385. db.lock.acquisition.timeout

Description	The maximum time interval within which lock should be acquired. Zero (default) means timeout is disabled.
Valid values	db.lock.acquisition.timeout, a duration (Valid units are: `ns`, `μs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`)
Dynamic	true
Default value	`0s`

Table 386. db.logs.query.early_raw_logging_enabled

Description	Log query text and parameters without obfuscating passwords. This allows queries to be logged earlier before parsing starts.
Valid values	db.logs.query.early_raw_logging_enabled, a boolean
Dynamic	true
Default value	`false`

Table 387. db.logs.query.enabled

Description	Log executed queries. Valid values are `OFF`, `INFO`, or `VERBOSE`.
	
	`OFF`
	no logging.
	
	`INFO`
	log queries at the end of execution, that take longer than the configured threshold, `db.logs.query.threshold`.
	
	`VERBOSE`
	log queries at the start and end of execution, regardless of `db.logs.query.threshold`.
	
	Log entries are written to the query log.
	
	This feature is available in the Neo4j Enterprise Edition.

Valid values	db.logs.query.enabled, one of [OFF, INFO, VERBOSE]
Dynamic	true
Default value	`VERBOSE`

Table 388. db.logs.query.max_parameter_length

Description	Sets a maximum character length use for each parameter in the log. This only takes effect if `db.logs.query.parameter_logging_enabled = true`.
Valid values	db.logs.query.max_parameter_length, an integer
Dynamic	true
Default value	`2147483647`

Table 389. db.logs.query.obfuscate_literals

Description	Obfuscates all literals of the query before writing to the log. Note that node labels, relationship types and map property keys are still shown. Changing the setting will not affect queries that are cached. So, if you want the switch to have immediate effect, you must also call `CALL db.clearQueryCaches()`.
Valid values	db.logs.query.obfuscate_literals, a boolean
Dynamic	true
Default value	`false`

Table 390. db.logs.query.parameter_logging_enabled

Description	Log parameters for the executed queries being logged.
Valid values	db.logs.query.parameter_logging_enabled, a boolean
Dynamic	true
Default value	`true`

Table 391. db.logs.query.plan_description_enabled

Description	Log query plan description table, useful for debugging purposes.
Valid values	db.logs.query.plan_description_enabled, a boolean
Dynamic	true
Default value	`false`

Table 392. db.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	db.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 393. db.logs.query.transaction.enabled

Description	Log the start and end of a transaction. Valid values are 'OFF', 'INFO', or 'VERBOSE'. OFF: no logging. INFO: log start and end of transactions that take longer than the configured threshold, db.logs.query.transaction.threshold. VERBOSE: log start and end of all transactions. Log entries are written to the query log. This feature is available in the Neo4j Enterprise Edition.
Valid values	db.logs.query.transaction.enabled, one of [OFF, INFO, VERBOSE]
Dynamic	true
Default value	`OFF`

Table 394. db.logs.query.transaction.threshold

Description	If the transaction is open for more time than this threshold, the transaction is logged once completed - provided transaction logging (db.logs.query.transaction.enabled) is set to `INFO`. Defaults to 0 seconds (all transactions are logged).
Valid values	db.logs.query.transaction.threshold, a duration (Valid units are: `ns`, `μs`, `ms`, `s`, `m`, `h` and `d`; default unit is `s`)
Dynamic	true
Default value	`0s`

Table 395. db.memory.transaction.max

Description	Limit the amount of memory that a single transaction can consume, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g'). Zero means 'largest possible value'.
Valid values	db.memory.transaction.max, 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`
Dynamic	true
Default value	`0B`

Table 396. db.memory.transaction.total.max

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

Valid values	db.memory.transaction.total.max, 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 397. db.track_query_cpu_time

Description	Enables or disables tracking of how much time a query spends actively executing on the CPU. Calling `SHOW TRANSACTIONS` will display the time. This can also be logged in the query log by using `db.logs.query.time_logging_enabled`.
Valid values	db.track_query_cpu_time, a boolean
Dynamic	true
Default value	`false`

Table 398. db.transaction.bookmark_ready_timeout

Description	The maximum amount of time to wait for the database state represented by the bookmark.
Valid values	db.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`
Dynamic	true
Default value	`30s`

Table 399. db.transaction.concurrent.maximum

Description	The maximum number of concurrently running transactions. If set to 0, limit is disabled.
Valid values	db.transaction.concurrent.maximum, an integer
Dynamic	true
Default value	`1000`

Table 400. db.transaction.sampling.percentage

Description	Transaction sampling percentage.
Valid values	db.transaction.sampling.percentage, an integer which is in the range `1` to `100`
Dynamic	true
Default value	`5`

Table 401. db.transaction.timeout

Description	The maximum time interval of a transaction within which it should be completed.
Valid values	db.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 402. db.transaction.tracing.level

Description	Transaction creation tracing level.
Valid values	db.transaction.tracing.level, one of [DISABLED, SAMPLE, ALL]
Dynamic	true
Default value	`DISABLED`

Table 403. db.tx_log.preallocate

Description	Specify if Neo4j should try to preallocate logical log file in advance.
Valid values	db.tx_log.preallocate, a boolean
Dynamic	true
Default value	`true`

Table 404. db.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	db.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	`2 days`

Table 405. db.tx_log.rotation.size

Description	Specifies at which file size the logical log will auto-rotate. Minimum accepted value is 128 KiB.
Valid values	db.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	`256.00MiB`

Table 406. dbms.cypher.render_plan_description

Description	If set to `true` a textual representation of the plan description will be rendered on the server for all queries running with `EXPLAIN` or `PROFILE`. This allows clients such as the neo4j browser and Cypher shell to show a more detailed plan description.
Valid values	dbms.cypher.render_plan_description, a boolean
Dynamic	true
Default value	`false`

Table 407. dbms.memory.transaction.total.max

Description	Limit the amount of memory that all of the running transactions can consume, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g'). Zero means 'unlimited'.
Valid values	dbms.memory.transaction.total.max, 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 408. dbms.routing.client_side.enforce_for_domains

Description	Always use client side routing (regardless of the default router) for neo4j:// protocol connections to these domains. A comma separated list of domains. Wildcards (*) are supported.
Valid values	dbms.routing.client_side.enforce_for_domains, a ',' separated set with elements of type 'a string'.
Dynamic	true
Default value	

Table 409. dbms.routing.reads_on_writers_enabled

Description	Enterprise edition Configure if the `dbms.routing.getRoutingTable()` procedure should include the writer as read endpoint or return only non-writers (non writer primaries and secondaries) Note: writer is returned as read endpoint if no other member is present all.
Valid values	dbms.routing.reads_on_writers_enabled, a boolean
Dynamic	true
Default value	`false`

Table 410. dbms.security.key.name

Description	Enterprise edition Name of the 256 length AES encryption key, which is used for the symmetric encryption.
Valid values	dbms.security.key.name, a string

Dynamic	true
Default value	`aesKey`

Table 411. dbms.security.keystore.password

Description	Enterprise edition Password for accessing the keystore holding a 256 length AES encryption key, which is used for the symmetric encryption.
Valid values	dbms.security.keystore.password, a secure string
Dynamic	true

Table 412. dbms.security.keystore.path

Description	Enterprise edition Location of the keystore holding a 256 length AES encryption key, which is used for the symmetric encryption of secrets held in system database.
Valid values	dbms.security.keystore.path, a path
Dynamic	true

Table 413. dbms.security.ldap.authentication.attribute

Description	Enterprise edition The attribute to use when looking up users. Using this setting requires `dbms.security.ldap.authentication.search_for_attribute` to be true and thus `dbms.security.ldap.authorization.system_username` and `dbms.security.ldap.authorization.system_password` to be configured.
Valid values	dbms.security.ldap.authentication.attribute, a string which matches the pattern `[A-Za-z0-9-]*` (has to be a valid LDAP attribute name, only containing letters [A-Za-z], digits [0-9] and hyphens [-].)
Dynamic	true
Default value	`samaccountname`

Table 414. dbms.security.ldap.authentication.user_dn_template

Description	Enterprise edition LDAP user DN template. An LDAP object is referenced by its distinguished name (DN), and a user DN is an LDAP fully-qualified unique user identifier. This setting is used to generate an LDAP DN that conforms with the LDAP directory's schema from the user principal that is submitted with the authentication token when logging in. The special token \{0} is a placeholder where the user principal will be substituted into the DN string.
Valid values	dbms.security.ldap.authentication.user_dn_template, a string which Must be a string containing '\{0}' to understand where to insert the runtime authentication principal.
Dynamic	true
Default value	`uid={0},ou=users,dc=example,dc=com`

Table 415. dbms.security.ldap.authorization.access_permitted_group

Description	Enterprise edition The LDAP group to which a user must belong to get any access to the system.Set this to restrict access to a subset of LDAP users belonging to a particular group. If this is not set, any user to successfully authenticate via LDAP will have access to the PUBLIC role and any other roles assigned to them via dbms.security.ldap.authorization.group_to_role_mapping.
Valid values	dbms.security.ldap.authorization.access_permitted_group, a string
Dynamic	true
Default value	

Table 416. dbms.security.ldap.authorization.group_membership_attributes

Description	Enterprise edition A list of attribute names on a user object that contains groups to be used for mapping to roles when LDAP authorization is enabled. This setting is ignored when `dbms.ldap_authorization_nested_groups_enabled` is `true`.
Valid values	dbms.security.ldap.authorization.group_membership_attributes, a ',' separated list with elements of type 'a string'. which Can not be empty
Dynamic	true
Default value	`memberOf`

Table 417. dbms.security.ldap.authorization.group_to_role_mapping

Description	Enterprise edition 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 which must be semicolon separated list of key-value pairs or empty
Dynamic	true
Default value	

Table 418. dbms.security.ldap.authorization.nested_groups_enabled

Description	Enterprise edition This setting determines whether multiple LDAP search results will be processed (as is required for the lookup of nested groups). If set to `true` then instead of using attributes on the user object to determine group membership (as specified by `dbms.security.ldap.authorization.group_membership_attributes`), the `user` object will only be used to determine the user's Distinguished Name, which will subsequently be used with `dbms.security.ldap.authorization.user_search_filter` in order to perform a nested group search. The Distinguished Names of the resultant group search results will be used to determine roles.
Valid values	dbms.security.ldap.authorization.nested_groups_enabled, a boolean
Dynamic	true
Default value	`false`

Table 419. dbms.security.ldap.authorization.nested_groups_search_filter

Description	Enterprise edition The search template which will be used to find the nested groups which the user is a member of. The filter should contain the placeholder token `\{0}` which will be substituted with the user's Distinguished Name (which is found for the specified user principle using `dbms.security.ldap.authorization.user_search_filter`). The default value specifies Active Directory's LDAP_MATCHING_RULE_IN_CHAIN (aka 1.2.840.113556.1.4.1941) implementation which will walk the ancestry of group membership for the specified user.
Valid values	dbms.security.ldap.authorization.nested_groups_search_filter, a string
Dynamic	true
Default value	`(&(objectclass=group)(member:1.2.840.113556.1.4.1941:={0}))`

Table 420. dbms.security.ldap.authorization.user_search_base

Description	Enterprise edition The name of the base object or named context to search for user objects when LDAP authorization is enabled. A common case is that this matches the last part of `dbms.security.ldap.authentication.user_dn_template`.
Valid values	dbms.security.ldap.authorization.user_search_base, a string which Can not be empty
Dynamic	true
Default value	`ou=users,dc=example,dc=com`

Table 421. dbms.security.ldap.authorization.user_search_filter

Description	Enterprise edition 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
Dynamic	true
Default value	`(&(objectClass=*)(uid={0}))`

Table 422. dbms.security.oidc.<provider>.audience

Description	Enterprise edition Expected values of the Audience (aud) claim in the id token.

Valid values	dbms.security.oidc.<provider>.audience, a ',' separated list with elements of type 'a string'. which Can not be empty
Dynamic	true

Table 423. dbms.security.oidc.<provider>.auth_endpoint

Description	Enterprise edition The OIDC authorization endpoint. If this is not supplied Neo4j will attempt to discover it from the well_known_discovery_uri.
Valid values	dbms.security.oidc.<provider>.auth_endpoint, a URI
Dynamic	true

Table 424. dbms.security.oidc.<provider>.auth_flow

Description	Enterprise edition The OIDC flow to use. This is exposed to clients via the discovery endpoint. Supported values are `pkce` and `implicit`
Valid values	dbms.security.oidc.<provider>.auth_flow, one of [PKCE, IMPLICIT]
Dynamic	true
Default value	`PKCE`

Table 425. dbms.security.oidc.<provider>.auth_params

Description	Enterprise edition Optional additional parameters that the auth endpoint requires. Please use params instead. The map is a semicolon separated list of key-value pairs. For example: `k1=v1;k2=v2`.
Valid values	dbms.security.oidc.<provider>.auth_params, A simple key value map pattern `k1=v1;k2=v2`.
Dynamic	true
Default value	`{}`
Deprecated	The `dbms.security.oidc.<provider>.auth_params` configuration setting has been deprecated.

Table 426. dbms.security.oidc.<provider>.authorization.group_to_role_mapping

Description	Enterprise edition An authorization mapping from IdP group names to Neo4j role names. The map should be formatted as a semicolon separated list of key-value pairs, where the key is the IdP group name and the value is a comma separated list of corresponding role names. For example: group1=role1;group2=role2;group3=role3,role4,role5 You could also use whitespaces and quotes around group names to make this mapping more readable, for example:
	`dbms.security.oidc.<provider>.authorization.group_to_role_mapping=\`
	`"Neo4j Read Only"      = reader;    \`
	`"Neo4j Read-Write"      = publisher; \`
	`"Neo4j Schema Manager" = architect; \`
	`"Neo4j Administrator"  = admin`

Valid values	dbms.security.oidc.<provider>.authorization.group_to_role_mapping, a string which must be semicolon separated list of key-value pairs or empty
Dynamic	true

Table 427. *dbms.security.oidc.<provider>.claims.groups*

Description	Enterprise edition The claim to use as the list of groups in Neo4j. These could be Neo4J roles directly, or can be mapped using dbms.security.oidc.<provider>.authorization.group_to_role_mapping.
Valid values	dbms.security.oidc.<provider>.claims.groups, a string
Dynamic	true

Table 428. *dbms.security.oidc.<provider>.claims.username*

Description	Enterprise edition The claim to use as the username in Neo4j. This would typically be sub, but in some situations it may be be desirable to use something else such as email.
Valid values	dbms.security.oidc.<provider>.claims.username, a string
Dynamic	true
Default value	`sub`

Table 429. *dbms.security.oidc.<provider>.client_id*

Description	Enterprise edition Client id needed if token contains multiple Audience (aud) claims.

Valid values	dbms.security.oidc.<provider>.client_id, a string
Dynamic	true

Table 430. dbms.security.oidc.<provider>.config

Description	Enterprise edition The accepted values (all optional) are:
	`principal: in which JWT claim the user's email address is specified, email is the default. This is the value that will be shown in browser.`
	`code_challenge_method: default is `S256` and it's the only supported method at this moment. This setting applies only for pkce auth flow`
	`token_type_principal: the options are almost always either access_token, which is the default, or id_token.`
	`token_type_authentication: the options are almost always either access_token, which is the default, or id_token.`
	`implicit_flow_requires_nonce: true or false. Defaults to false.`

Valid values	dbms.security.oidc.<provider>.config, A simple key value map pattern `k1=v1;k2=v2`. Valid key options are: `[principal, code_challenge_method, implicit_flow_requires_nonce, token_type_authentication, token_type_principal]`.
Dynamic	true
Default value	`{}`

Table 431. dbms.security.oidc.<provider>.get_groups_from_user_info

Description	Enterprise edition When turned on, Neo4j gets the groups from the provider user info endpoint.
Valid values	dbms.security.oidc.<provider>.get_groups_from_user_info, a boolean
Dynamic	true
Default value	`false`

Table 432. dbms.security.oidc.<provider>.get_username_from_user_info

Description	Enterprise edition When turned on, Neo4j gets the username from the provider user info endpoint.
Valid values	dbms.security.oidc.<provider>.get_username_from_user_info, a boolean
Dynamic	true

Default value	`false`

Table 433. dbms.security.oidc.<provider>.issuer

Description	Enterprise edition The expected value of the iss claim in the id token. If this is not supplied Neo4j will attempt to discover it from the well_known_discovery_uri.
Valid values	dbms.security.oidc.<provider>.issuer, a string
Dynamic	true

Table 434. dbms.security.oidc.<provider>.jwks_uri

Description	Enterprise edition The location of the JWK public key set for the identity provider. If this is not supplied Neo4j will attempt to discover it from the well_known_discovery_uri.
Valid values	dbms.security.oidc.<provider>.jwks_uri, a URI
Dynamic	true

Table 435. dbms.security.oidc.<provider>.params

Description	Enterprise edition The map is a semicolon separated list of key-value pairs. For example: `k1=v1;k2=v2`. The user should at least provide:
	`client_id: the SSO Idp client idenfifier.`
	`response_type: code if auth_flow is pkce or token for implicit auth_flow.`
	`scope: often containing a subset of 'email profile openid groups'.`
	For example: `client_id=my-client-id;response_type=code;scope=openid profile email`.

Valid values	dbms.security.oidc.<provider>.params, A simple key value map pattern `k1=v1;k2=v2`. Required key options are: `[scope, client_id, response_type]`.
Dynamic	true
Default value	`{}`

Table 436. dbms.security.oidc.<provider>.token_endpoint

Description	Enterprise edition The OIDC token endpoint. If this is not supplied Neo4j will attempt to discover it from the well_known_discovery_uri.

Valid values	dbms.security.oidc.<provider>.token_endpoint, a URI
Dynamic	true

Table 437. dbms.security.oidc.<provider>.token_params

Description	Enterprise edition Optional query parameters that the token endpoint requires. The map is a semicolon separated list of key-value pairs. For example: `k1=v1;k2=v2`.If the token endpoint requires a client_secret then this parameter should contain `client_secret=super-secret`
Valid values	dbms.security.oidc.<provider>.token_params, A simple key value map pattern `k1=v1;k2=v2`.
Dynamic	true
Default value	`{}`

Table 438. dbms.security.oidc.<provider>.user_info_uri

Description	Enterprise edition The identity providers user info uri.
Valid values	dbms.security.oidc.<provider>.user_info_uri, a URI
Dynamic	true

Table 439. dbms.security.oidc.<provider>.well_known_discovery_uri

Description	Enterprise edition The 'well known' OpenID Connect Discovery endpoint used to fetch identity provider settings. If not provided, `issuer`, `jwks_uri`, `auth_endpoint` should be present. If the auth_flow is pkce, `token_endpoint` should also be provided.
Valid values	dbms.security.oidc.<provider>.well_known_discovery_uri, a URI
Dynamic	true

Table 440. server.cluster.catchup.connect_randomly_to_server_group

Description	Enterprise edition 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 (`server.cluster.catchup.upstream_strategy`) includes the value `connect-randomly-to-server-group`.

Valid values	server.cluster.catchup.connect_randomly_to_server_group, a ',' separated list with elements of type 'a string identifying a Server Tag'.
Dynamic	true
Default value	

Table 441. server.databases.default_to_read_only

Description	Whether or not any database on this instance are read_only by default. If false, individual databases may be marked as read_only using server.database.read_only. If true, individual databases may be marked as writable using server.databases.writable.
Valid values	server.databases.default_to_read_only, a boolean
Dynamic	true
Default value	`false`

Table 442. server.databases.read_only

Description	List of databases for which to prevent write queries. Databases not included in this list maybe read_only anyway depending upon the value of server.databases.default_to_read_only.
Valid values	server.databases.read_only, a ',' separated set with elements of type 'A valid database name containing only alphabetic characters, numbers, dots and dashes with a length between 3 and 63 characters, starting with an alphabetic character but not with the name 'system''. which Value 'system' can't be included in read only databases collection!
Dynamic	true
Default value	

Table 443. server.databases.writable

Description	List of databases for which to allow write queries. Databases not included in this list will allow write queries anyway, unless server.databases.default_to_read_only is set to true.

Valid values	server.databases.writable, a ',' separated set with elements of type 'A valid database name containing only alphabetic characters, numbers, dots and dashes with a length between 3 and 63 characters, starting with an alphabetic character but not with the name 'system''.
Dynamic	true
Default value	

Table 444. server.groups

Description	Enterprise edition A list of tag names for the server used when configuring load balancing and replication policies.
Valid values	server.groups, a ',' separated list with elements of type 'a string identifying a Server Tag'.
Dynamic	true
Default value	

Table 445. server.memory.pagecache.flush.buffer.enabled

Description	Page cache can be configured to use a temporal buffer for flushing purposes. It is used to combine, if possible, sequence of several cache pages into one bigger buffer to minimize the number of individual IOPS performed and better utilization of available I/O resources, especially when those are restricted.
Valid values	server.memory.pagecache.flush.buffer.enabled, a boolean
Dynamic	true
Default value	`false`

Table 446. server.memory.pagecache.flush.buffer.size_in_pages

Description	Page cache can be configured to use a temporal buffer for flushing purposes. It is used to combine, if possible, sequence of several cache pages into one bigger buffer to minimize the number of individual IOPS performed and better utilization of available I/O resources, especially when those are restricted. Use this setting to configure individual file flush buffer size in pages (8KiB). To be able to utilize this buffer during page cache flushing, buffered flush should be enabled.
Valid values	server.memory.pagecache.flush.buffer.size_in_pages, an integer which is in the range `1` to `512`

Dynamic	true
Default value	`128`

Procedures

This page provides a complete reference to the Neo4j procedures. Available procedures depend on the type of installation you have:

- Neo4j Enterprise Edition provides a larger set of procedures than Neo4j Community Edition.
- Cluster members have procedures that are not available in standalone mode.

To check which procedures are available in your Neo4j DBMS, use the Cypher command `SHOW PROCEDURES`:

Example 127. List available procedures

```
SHOW PROCEDURES
```

Some procedures can only be run by users with `Admin` privileges. These are labeled with Admin only.

For more information, see Cypher Manual → Manage Privileges.

List of procedures

Table 447. Neo4j procedures

Name	Community Edition	Enterprise Edition	Comment
`db.awaitIndex()`	Yes	Yes	
`db.awaitIndexes()`	Yes	Yes	
`db.checkpoint()`	No	Yes	
`db.clearQueryCaches()`	Yes	Yes	Admin only
`db.createLabel()`	Yes	Yes	
`db.createProperty()`	Yes	Yes	
`db.createRelationshipType()`	Yes	Yes	
`db.index.fulltext.awaitEventuallyConsistentIndexRefresh()`	Yes	Yes	
`db.index.fulltext.listAvailableAnalyzers()`	Yes	Yes	

Name	Community Edition	Enterprise Edition	Comment
`db.index.fulltext.queryNodes()`	Yes	Yes	In 4.1, signature changed to `db.index.fulltext.queryNodes(indexName :: STRING?, queryString :: STRING?, options = {} :: MAP?) :: (node :: NODE?, score :: FLOAT?)`.
`db.index.fulltext.queryRelationships()`	Yes	Yes	In 4.1, signature changed to `db.index.fulltext.queryRelationships(indexName :: STRING?, queryString :: STRING?, options = {} :: MAP?) :: (relationship :: RELATIONSHIP?, score :: FLOAT?)`.
`db.info()`	Yes	Yes	
`db.labels()`	Yes	Yes	
`db.listLocks()`	No	Yes	Admin only In 4.2, signature changed to `db.listLocks() :: (mode :: STRING?, resourceType :: STRING?, resourceId :: INTEGER?, transactionId :: STRING?)`.
`db.ping()`	Yes	Yes	
`db.prepareForReplanning()`	Yes	Yes	Admin only
`db.propertyKeys()`	Yes	Yes	
`db.relationshipTypes()`	Yes	Yes	
`db.resampleIndex()`	Yes	Yes	
`db.resampleOutdatedIndexes()`	Yes	Yes	
`db.schema.nodeTypeProperties()`	Yes	Yes	
`db.schema.relTypeProperties()`	Yes	Yes	
`db.schema.visualization()`	Yes	Yes	
`db.stats.clear()`	Yes	Yes	Admin only
`db.stats.collect()`	Yes	Yes	Admin only
`db.stats.retrieve()`	Yes	Yes	Admin only
`db.stats.retrieveAllAnonymized()`	Yes	Yes	Admin only
`db.stats.status()`	Yes	Yes	Admin only
`db.stats.stop()`	Yes	Yes	Admin only
`dbms.checkConfigValue()`	No	Yes	New in 5.0.
`dbms.cluster.checkConnectivity()`	No	Yes	Admin only
`dbms.cluster.cordonServer()`	No	Yes	Admin only
`dbms.cluster.protocols()`	No	Yes	
`dbms.cluster.readReplicaToggle()`	No	Yes	Admin only

Name	Community Edition	Enterprise Edition	Comment
`dbms.cluster.routing.getRoutingTable()`	Yes	Yes	
`dbms.cluster.uncordonServer()`	No	Yes	Admin only
`dbms.components()`	Yes	Yes	
`dbms.info()`	Yes	Yes	
`dbms.killConnection()`	Yes	Yes	
`dbms.killConnections()`	Yes	Yes	
`dbms.listActiveLocks()`	Yes	Yes	
`dbms.listCapabilities()`	Yes	Yes	
`dbms.listConfig()`	Yes	Yes	Admin only
`dbms.listConnections()`	Yes	Yes	
`dbms.listPools()`	No	Yes	
`dbms.quarantineDatabase()`	No	Yes	Admin only
`dbms.queryJmx()`	Yes	Yes	
`dbms.routing.getRoutingTable()`	Yes	Yes	
`dbms.scheduler.failedJobs()`	No	Yes	Admin only
`dbms.scheduler.groups()`	No	Yes	Admin only
`dbms.scheduler.jobs()`	No	Yes	Admin only
`dbms.security.clearAuthCache()`	No	Yes	Admin only
`dbms.setConfigValue()`	No	Yes	Admin only
`dbms.setDatabaseAllocator()`	No	Yes	Admin only
`dbms.setDefaultAllocationNumbers()`	No	Yes	Admin only
`dbms.setDefaultDatabase()`	No	Yes	Admin only
`dbms.showCurrentUser()`	Yes	Yes	
`dbms.showTopologyGraphConfig()`	No	Yes	Admin only
`dbms.upgrade()`	Yes	Yes	Admin only
`dbms.upgradeStatus()`	Yes	Yes	Admin only
`tx.getMetaData()`	Yes	Yes	
`tx.setMetaData()`	Yes	Yes	

List of removed procedures

Table 448. Removed Neo4j procedures

Name	Community Edition	Enterprise Edition	Comment
`db.constraints()`	Yes	Yes	Removed Replaced by: `SHOW CONSTRAINTS`.
`db.createIndex()`	Yes	Yes	Removed Replaced by: `CREATE INDEX`.
`db.createNodeKey()`	No	Yes	Removed Replaced by: `CREATE CONSTRAINT … IS NODE KEY`.
`db.createUniquePropertyConstraint()`	Yes	Yes	Removed Replaced by: `CREATE CONSTRAINT … IS UNIQUE`.
`db.indexes()`	Yes	Yes	Removed Replaced by: `SHOW INDEXES`.
`db.indexDetails()`	Yes	Yes	Removed Replaced by: `SHOW INDEXES YIELD*`.
`db.index.fulltext.createNodeIndex()`	Yes	Yes	Removed Replaced by: `CREATE FULLTEXT INDEX …`.
`db.index.fulltext.createRelationshipIndex()`	Yes	Yes	Removed Replaced by: `CREATE FULLTEXT INDEX …`.
`db.index.fulltext.drop()`	Yes	Yes	Removed Replaced by: `DROP INDEX …`.
`db.schema.statements()`	Yes	Yes	Removed Replaced by: `SHOW INDEXES YIELD *` and `SHOW CONSTRAINTS YIELD *`.
`dbms.cluster.overview()`	No	Yes	Removed Replaced by: `SHOW SERVERS`.
`dbms.cluster.quarantineDatabase()`	No	Yes	Removed Replaced by: `dbms.quarantineDatabase()`.
`dbms.cluster.role()`	No	Yes	Removed Replaced by: `SHOW DATABASES`.
`dbms.cluster.setDefaultDatabase()`	No	Yes	Removed Replaced by: `dbms.setDefaultDatabase`.
`dbms.database.state()`	Yes	Yes	Removed Replaced by: `SHOW DATABASES`.
`dbms.functions()`	Yes	Yes	Removed Replaced by: `SHOW FUNCTIONS`.
`dbms.killQueries()`	Yes	Yes	Removed Replaced by: `TERMINATE TRANSACTIONS`.
`dbms.killQuery()`	Yes	Yes	Removed Replaced by: `TERMINATE TRANSACTIONS`.

Name	Community Edition	Enterprise Edition	Comment
`dbms.killTransaction()`	Yes	Yes	Removed Replaced by: `TERMINATE TRANSACTIONS`.
`dbms.killTransactions()`	Yes	Yes	Removed Replaced by: `TERMINATE TRANSACTIONS`.
`dbms.listQueries()`	Yes	Yes	Removed Replaced by: `SHOW TRANSACTIONS`.
`dbms.lisTransactions()`	Yes	Yes	Removed Replaced by: `SHOW TRANSACTIONS`.
`dbms.procedures()`	No	Yes	Removed Replaced by: `SHOW PROCEDURES`.
`dbms.security.activateUser()`	No	Yes	Removed Admin only In 4.1, mode changed to `write`. Replaced by: `ALTER USER`.
`dbms.security.addRoleToUser()`	No	Yes	Removed Admin only In 4.1, mode changed to `write`. Replaced by: `GRANT ROLE TO USER`.
`dbms.security.changePassword()`	Yes	Yes	Removed Admin only In 4.1, mode changed to `write`. Replaced by: `ALTER CURRENT USER SET PASSWORD`.
`dbms.security.changeUserPassword()`	No	Yes	Removed Admin only In 4.1, mode changed to `write`. Replaced by: `ALTER USER`.
`dbms.security.createRole()`	No	Yes	Removed Admin only In 4.1, mode changed to `write`. Replaced by: `CREATE ROLE`.
`dbms.security.createUser()`	Yes	Yes	Removed Admin only In 4.1, mode changed to `write`. Replaced by: `CREATE USER`.
`dbms.security.deleteRole()`	No	Yes	Removed Admin only In 4.1, mode changed to `write`. Replaced by: `DROP ROLE`.
`dbms.security.deleteUser()`	Yes	Yes	Removed Admin only In 4.1, mode changed to `write`. Replaced by: `DROP USER`.
`dbms.security.listRoles()`	Yes	Yes	Removed Admin only In 4.1, mode changed to `read`. Replaced by: `SHOW ROLES`.
`dbms.security.listRolesForUser()`	No	Yes	Removed In 4.1, mode changed to `read`. Replaced by: `SHOW USERS`.

Name	Community Edition	Enterprise Edition	Comment
`dbms.security.listUsers()`	Yes	Yes	Removed Admin only In 4.1, mode changed to `read`. Replaced by: `SHOW USERS`.
`dbms.security.listUsersForRole()`	No	Yes	Removed Admin only In 4.1, mode changed to `read`. Replaced by: `SHOW ROLES WITH USERS`.
`dbms.security.removeRoleFromUser()`	No	Yes	Removed Admin only In 4.1, mode changed to `write`. Replaced by: `REVOKE ROLE FROM USER`.
`dbms.security.suspendUser()`	No	Yes	Removed Admin only In 4.1, mode changed to `write`. Replaced by: `ALTER USER`.

Procedure descriptions

Table 449. 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`

Table 450. db.awaitIndexes()

Description	Wait for all indexes to come online. Example: `CALL db.awaitIndexes(300)`
Signature	`db.awaitIndexes(timeOutSeconds = 300 :: INTEGER?) :: VOID`
Mode	`READ`

Table 451. db.checkpoint() Enterprise edition

Description	Initiate and wait for a new check point, or wait any already on-going check point to complete. Note that this temporarily disables the `db.checkpoint.iops.limit` setting in order to make the check point complete faster. This might cause transaction throughput to degrade slightly, due to increased IO load.
Signature	`db.checkpoint() :: (success :: BOOLEAN?, message :: STRING?)`
Mode	`DBMS`

Table 452. db.clearQueryCaches() Admin only

Description	Clears all query caches.
Signature	`db.clearQueryCaches() :: (value :: STRING?)`
Mode	`DBMS`

Table 453. db.createLabel()

Description	Create a label
Signature	`db.createLabel(newLabel :: STRING?) :: VOID`
Mode	`WRITE`

Table 454. db.createNodeKey() Enterprise edition Deprecated

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`
Replaced by	`CREATE CONSTRAINT … IS NODE KEY`. For more information, see Database administration.

Table 455. db.createProperty()

Description	Create a Property
Signature	`db.createProperty(newProperty :: STRING?) :: VOID`
Mode	`WRITE`

Table 456. db.createRelationshipType()

Description	Create a RelationshipType
Signature	`db.createRelationshipType(newRelationshipType :: STRING?) :: VOID`
Mode	`WRITE`

Table 457. db.index.fulltext.awaitEventuallyConsistentIndexRefresh()

Description	Wait for the updates from recently committed transactions to be applied to any eventually-consistent full-text indexes.

Signature	`db.index.fulltext.awaitEventuallyConsistentIndexRefresh() :: VOID`
Mode	`READ`

Table 458. db.index.fulltext.listAvailableAnalyzers()

Description	List the available analyzers that the full-text indexes can be configured with.
Signature	`db.index.fulltext.listAvailableAnalyzers() :: (analyzer :: STRING?, description :: STRING?, stopwords :: LIST? OF STRING?)`
Mode	`READ`

Table 459. db.index.fulltext.queryNodes()

Description	Query the given full-text index. Returns the matching nodes and their Lucene query score, ordered by score. Valid *key: value* pairs for the `options` map are: • `skip: <number>` — skip the top N results. • `limit: <number>` — limit the number of results returned. • `analyzer: <string>` — use the specified analyzer as a search analyzer for this query. The `options` map and any of the keys are optional. An example of the `options` map: `{skip: 30, limit: 10, analyzer: 'whitespace'}`
Signature	`db.index.fulltext.queryNodes(indexName :: STRING?, queryString :: STRING?, options = {} :: MAP?) :: (node :: NODE?, score :: FLOAT?)`
Mode	`READ`

Table 460. db.index.fulltext.queryRelationships()

Description	Query the given full-text index. Returns the matching relationships and their Lucene query score, ordered by score. Valid *key: value* pairs for the `options` map are: • `skip: <number>` — skip the top N results. • `limit: <number>` — limit the number of results returned. • `analyzer: <string>` — use the specified analyzer as a search analyzer for this query. The `options` map and any of the keys are optional. An example of the `options` map: `{skip: 30, limit: 10, analyzer: 'whitespace'}`
Signature	`db.index.fulltext.queryRelationships(indexName :: STRING?, queryString :: STRING?, options = {} :: MAP?) :: (relationship :: RELATIONSHIP?, score :: FLOAT?)`
Mode	`READ`

Table 461. db.info()

Description	Provides information regarding the database.
Signature	`db.info() :: (id :: STRING?, name :: STRING?, creationDate :: STRING?)`
Mode	`READ`

Table 462. db.labels()

Description	List all available labels in the database.
Signature	`db.labels() :: (label :: STRING?)`
Mode	`READ`

Table 463. db.listLocks() Enterprise edition Admin only

Description	List all locks at this database.
Signature	`db.listLocks() :: (mode :: STRING?, resourceType :: STRING?, resourceId :: INTEGER?, transactionId :: STRING?)`
Mode	`DBMS`

Table 464. db.ping()

Description	This procedure can be used by client side tooling to test whether they are correctly connected to a database. The procedure is available in all databases and always returns true. A faulty connection can be detected by not being able to call this procedure.
Signature	`db.ping() :: (success :: BOOLEAN?)`
Mode	`READ`

Table 465. db.prepareForReplanning() Admin only

Description	Triggers an index resample and waits for it to complete, and after that clears query caches. After this procedure has finished queries will be planned using the latest database statistics.
Signature	`db.prepareForReplanning(timeOutSeconds = 300 :: INTEGER?) :: VOID`
Mode	`READ`

Table 466. db.propertyKeys()

Description	List all property keys in the database.
Signature	`db.propertyKeys() :: (propertyKey :: STRING?)`
Mode	`READ`

Table 467. db.relationshipTypes()

Description	List all available relationship types in the database.
Signature	`db.relationshipTypes() :: (relationshipType :: STRING?)`
Mode	`READ`

Table 468. db.resampleIndex()

Description	Schedule resampling of an index. Example: `CALL db.resampleIndex("MyIndex")`
Signature	`db.resampleIndex(indexName :: STRING?) :: VOID`
Mode	`READ`

Table 469. db.resampleOutdatedIndexes()

Description	Schedule resampling of all outdated indexes.
Signature	`db.resampleOutdatedIndexes() :: VOID`
Mode	`READ`

Table 470. db.schema.nodeTypeProperties()

Description	Show the derived property schema of the nodes in tabular form.
Signature	`db.schema.nodeTypeProperties() :: (nodeType :: STRING?, nodeLabels :: LIST? OF STRING?, propertyName :: STRING?, propertyTypes :: LIST? OF STRING?, mandatory :: BOOLEAN?)`
Mode	`READ`

Table 471. db.schema.relTypeProperties()

Description	Show the derived property schema of the relationships in tabular form.
Signature	`db.schema.relTypeProperties() :: (relType :: STRING?, propertyName :: STRING?, propertyTypes :: LIST? OF STRING?, mandatory :: BOOLEAN?)`
Mode	`READ`

Table 472. db.schema.visualization()

Description	Visualize the schema of the data.
Signature	`db.schema.visualization() :: (nodes :: LIST? OF NODE?, relationships :: LIST? OF RELATIONSHIP?)`
Mode	`READ`

Table 473. db.stats.clear() Admin only

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`

Table 474. db.stats.collect() Admin only

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`

Table 475. db.stats.retrieve() Admin only

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`

Table 476. db.stats.retrieveAllAnonymized() Admin only

Description	Retrieve all available statistical data about the current database, in an anonymized form.
Signature	`db.stats.retrieveAllAnonymized(graphToken :: STRING?, config = {} :: MAP?) :: (section :: STRING?, data :: MAP?)`
Mode	`READ`

Table 477. db.stats.status() Admin only

Description	Retrieve the status of all available collector daemons, for this database.
Signature	`db.stats.status() :: (section :: STRING?, status :: STRING?, data :: MAP?)`
Mode	`READ`

Table 478. db.stats.stop() Admin only

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`

Table 479. dbms.checkConfigValue() Enterprise edition

Description	This procedure provides feedback about the validity of a setting value. It does not change the setting. The procedure returns: • `valid`: if the value is valid. A valid value for a non-dynamic setting requires a restart. • `message`: a message describing the reason for the invalidity. The message is empty if the value is `valid` and the setting is dynamic.
Signature	`dbms.checkConfigValue(setting :: STRING?, value :: STRING?) :: (valid :: BOOLEAN?, message :: STRING?)`
Mode	`DBMS`

Table 480. dbms.cluster.checkConnectivity() Enterprise edition Admin only

Description	Checks the connectivity of this instance to other cluster members. Not all ports are relevant to all members. Valid values for 'port-name' are: [CLUSTER, RAFT].
Signature	`dbms.cluster.checkConnectivity`
Mode	`DBMS`

Table 481. dbms.cluster.cordonServer() Enterprise edition Admin only

Description	Marks a server in the topology as not suitable for new allocations. It will not force current allocations off the server. This is useful when deallocating databases when you have multiple unavailable servers.
Signature	`dbms.cluster.cordonServer(server :: STRING?)`
Mode	`WRITE`

Table 482. dbms.cluster.routing.getRoutingTable()

Description	Returns endpoints of this instance. Used in disaster recovery.
Signature	`dbms.cluster.routing.getRoutingTable(context :: MAP?, database = null :: STRING?) :: (ttl :: INTEGER?, servers :: LIST? OF MAP?)`
Mode	`DBMS`

Table 483. dbms.cluster.protocols() Enterprise edition

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`

Table 484. dbms.cluster.readReplicaToggle() Enterprise edition Admin only

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 database which runs in a secondary role on the connected server.
	
	Pause transaction pulling for database `neo4j`
	
	`CALL dbms.cluster.readReplicaToggle("neo4j", true)`
	
	Resume transaction pulling for database `neo4j`
	
	`CALL dbms.cluster.readReplicaToggle("neo4j", false)`

Signature	`dbms.cluster.readReplicaToggle(databaseName :: STRING?, pause :: BOOLEAN?) :: (state :: STRING?)`
Mode	`READ`

Table 485. dbms.cluster.uncordonServer() Enterprise edition Admin only

Description	Removes the cordon on a server, returning it to 'enabled'.
Signature	`dbms.cluster.uncordonServer(server :: STRING?)`
Mode	`WRITE`

Table 486. dbms.components()

Description	List DBMS components and their versions.
Signature	`dbms.components() :: (name :: STRING?, versions :: LIST? OF STRING?, edition :: STRING?)`
Mode	`DBMS`

Table 487. dbms.info()

Description	Provides information regarding the DBMS.
Signature	`dbms.info() :: (id :: STRING?, name :: STRING?, creationDate :: STRING?)`
Mode	`DBMS`

Table 488. dbms.killConnection()

Description	Kill network connection with the given connection id.
Signature	`dbms.killConnection(id :: STRING?) :: (connectionId :: STRING?, username :: STRING?, message :: STRING?)`
Mode	`DBMS`

Table 489. dbms.killConnections()

Description	Kill all network connections with the given connection ids.
Signature	`dbms.killConnections(ids :: LIST? OF STRING?) :: (connectionId :: STRING?, username :: STRING?, message :: STRING?)`
Mode	`DBMS`

Table 490. dbms.listActiveLocks() Enterprise edition

Description	List the active lock requests granted for the transaction executing the query with the given query id.
Signature	`dbms.listActiveLocks(queryId :: STRING?) :: (mode :: STRING?, resourceType :: STRING?, resourceId :: INTEGER?)`
Mode	`DBMS`

Table 491. dbms.listCapabilities()

Description	List capabilities.

Signature	`dbms.listCapabilities() :: (name :: STRING?, description :: STRING?, value :: ANY?)`
Mode	`DBMS`

Table 492. dbms.listConfig() Admin only

Description	List all configuration settings whose names include the `searchString`, or all configuration settings if `searchString` is omitted. Use this procedure to check a setting's values, investigate when a value was set, or find information on valid values. Results include: • `value`: the current value of the setting. • `dynamic`: true if the value can be changed at runtime, or false if it requires a restart. • `defaultValue`: the default value of the setting. • `startupValue`: the value when the server was started, after *neo4j.conf* and command line arguments have been applied. • `explicitlySet`: true if the value was set explicitly, or false if it was set by default. • `validValues`: contains information on data types and possible values for the settings.
Signature	`dbms.listConfig(searchString = :: STRING?) :: (name :: STRING?, description :: STRING?, value :: STRING?, dynamic :: BOOLEAN?, defaultValue :: STRING?, startupValue :: STRING?, explicitlySet :: BOOLEAN?, validValues :: STRING?)`
Mode	`DBMS`

Table 493. dbms.listConnections()

Description	List all accepted network connections at this instance that are visible to the user.
Signature	`dbms.listConnections() :: (connectionId :: STRING?, connectTime :: STRING?, connector :: STRING?, username :: STRING?, userAgent :: STRING?, serverAddress :: STRING?, clientAddress :: STRING?)`
Mode	`DBMS`

Table 494. dbms.listPools() Enterprise edition

Description	List all memory pools, including sub pools, currently registered at this instance that are visible to the user.
Signature	`dbms.listPools() :: (pool :: STRING?, databaseName :: STRING?, heapMemoryUsed :: STRING?, heapMemoryUsedBytes :: STRING?, nativeMemoryUsed :: STRING?, nativeMemoryUsedBytes :: STRING?, freeMemory :: STRING?, freeMemoryBytes :: STRING?, totalPoolMemory :: STRING?, totalPoolMemoryBytes :: STRING?)`
Mode	`DBMS`

Table 495. dbms.quarantineDatabase() Enterprise edition Admin only

Description	Place a database in quarantine or remove thereof. It must be executed over `bolt://`.
Signature	`dbms.quarantineDatabase(databaseName :: STRING?, setStatus :: BOOLEAN?, reason = No reason given :: STRING?) :: (databaseName :: STRING?, quarantined :: BOOLEAN?, result :: STRING?)`
Mode	`DBMS`

Table 496. 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`

Table 497. dbms.routing.getRoutingTable()

Description	Returns endpoints of this instance.
Signature	`dbms.routing.getRoutingTable(context :: MAP?, database = null :: STRING?) :: (ttl :: INTEGER?, servers :: LIST? OF MAP?)`
Mode	`DBMS`

Table 498. dbms.setDatabaseAllocator() Enterprise edition Admin only

Description	With this method you can set the allocator, which is responsible to select servers for hosting databases.
Signature	`dbms.setDatabaseAllocator(allocator :: STRING?)`
Mode	WRITE

Table 499. dbms.setDefaultAllocationNumbers() Enterprise edition Admin only

Description	With this method you can set the default number of primaries and secondaries.
Signature	`dbms.setDefaultAllocationNumbers(primaries :: INTEGER?, secondaries :: INTEGER?)`
Mode	WRITE

Table 500. dbms.setDefaultDatabase() Enterprise edition Admin only

Description	Changes the default database to the provided value. The database must exist and the old default database must be stopped.

Signature	`dbms.setDefaultDatabase(databaseName :: STRING?) :: (result :: STRING?)`
Mode	WRITE

Table 501. dbms.scheduler.failedJobs() Enterprise edition Admin only

Description	List failed job runs. There is a limit for amount of historical data.
Signature	`dbms.scheduler.failedJobs() :: (jobId :: STRING?, group :: STRING?, database :: STRING?, submitter :: STRING?, description :: STRING?, type :: STRING?, submitted :: STRING?, executionStart :: STRING?, failureTime :: STRING?, failureDescription :: STRING?)`
Mode	`DBMS`

Table 502. dbms.scheduler.groups() Enterprise edition Admin only

Description	List the job groups that are active in the database internal job scheduler.
Signature	`dbms.scheduler.groups() :: (group :: STRING?, threads :: INTEGER?)`
Mode	`DBMS`

Table 503. dbms.scheduler.jobs() Enterprise edition Admin only

Description	List all jobs that are active in the database internal job scheduler.
Signature	`dbms.scheduler.jobs() :: (jobId :: STRING?, group :: STRING?, submitted :: STRING?, database :: STRING?, submitter :: STRING?, description :: STRING?, type :: STRING?, scheduledAt :: STRING?, period :: STRING?, state :: STRING?, currentStateDescription :: STRING?)`
Mode	`DBMS`

Table 504. dbms.security.clearAuthCache() Enterprise edition Admin only

Description	Clears authentication and authorization cache.
Signature	`dbms.security.clearAuthCache() :: VOID`
Mode	`DBMS`

Table 505. dbms.setConfigValue() Enterprise edition Admin only

Description	Update a given setting value. Passing an empty value results in removing the configured value and falling back to the default value. Changes do not persist and are lost if the server is restarted. In a clustered environment, `dbms.setConfigValue` affects only the cluster member it is run against.
Signature	`dbms.setConfigValue(setting :: STRING?, value :: STRING?) :: VOID`
Mode	`DBMS`

Table 506. dbms.showCurrentUser()

Description	Show the current user.
Signature	`dbms.showCurrentUser() :: (username :: STRING?, roles :: LIST? OF STRING?, flags :: LIST? OF STRING?)`
Mode	`DBMS`

Table 507. dbms.showTopologyGraphConfig() Enterprise only Admin only

Description	With this method the configuration of the Topology Graph can be displayed.
Signature	`dbms.showTopologyGraphConfig() :: (allocator :: STRING?, defaultPrimariesCount :: INTEGER?, defaultSecondariesCount :: INTEGER?, defaultDatabase :: STRING?)`
Mode	`READ`

Table 508. dbms.upgrade() Admin only

Description	Upgrade the system database schema if it is not the current schema.
Signature	`dbms.upgrade() :: (status :: STRING?, upgradeResult :: STRING?)`
Mode	`WRITE`

Table 509. dbms.upgradeStatus() Admin only

Description	Report the current status of the system database sub-graph schema.
Signature	`dbms.upgradeStatus() :: (status :: STRING?, description :: STRING?, resolution :: STRING?)`
Mode	`READ`

Table 510. tx.getMetaData()

Description	Provides attached transaction metadata.
Signature	`tx.getMetaData() :: (metadata :: MAP?)`
Mode	`DBMS`

Table 511. tx.setMetaData()

Description	Attaches a map of data to the transaction. The data will be printed when listing queries, and inserted into the query log.
Signature	`tx.setMetaData(data :: MAP?) :: VOID`
Mode	`DBMS`

Appendix B: Tutorials

The following step-by-step tutorials cover common operational tasks or otherwise exemplify working with

Neo4j:

- Neo4j Admin import — This tutorial provides detailed examples to illustrate the capabilities of importing data from CSV files with the command `neo4j-admin database import`.
- Set up and use a composite database — This tutorial walks through the basics of setting up and using composite databases.
- SSO configuration — This tutorial presents examples and solutions to common problems when configuring SSO.
- Administering immutable privileges — This tutorial describes methods for administering immutable privileges.

Neo4j-admin import

This tutorial provides detailed examples to illustrate the capabilities of importing data from CSV files with the command `neo4j-admin database import`.

The `neo4j-admin database import` is a command for loading large amounts of data from CSV files into an **unused non-existing** database. Importing data from CSV files with `neo4j-admin database import` can only be done once into an **unused** database, it is used for the initial graph population only. The `neo4j-admin database import` command can be used on the local Neo4j instance even if the instance is running or not.

The `neo4j-admin database import` command does not create a database, the command only imports data and makes it available for the database. The database must not exist before the `neo4j-admin database import` command has been executed, and the database should be created afterward. The command will exit with an error message if the database already exists.

Relationships are created by connecting node IDs, each node should have a unique ID to be able to be referenced when creating relationships between nodes. In the following examples, the node IDs are stored as properties on the nodes. If you do not want the IDs to persist as properties after the import completes, then do not specify a property name in the `:ID` field.

The examples show how to import data in a standalone Neo4j DBMS. They use:

- The Neo4j tarball (Unix console application).
- `$NEO4J_HOME` as the current working directory.
- The default database `neo4j`.
- The *import* directory of the Neo4j installation to store all the CSV files. However, the CSV files can be located in any directory of your file system.
- UNIX-styled paths.
- The `neo4j-admin database import` command.

Handy tips:

- The details of a CSV file header format can be found at CSV header format.
- To show available databases, use the Cypher query `SHOW DATABASES` 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 belong to the same set.

movies.csv

```
movieId:ID,title,year:int,:LABEL
tt0133093,"The Matrix",1999,Movie
tt0234215,"The Matrix Reloaded",2003,Movie;Sequel
tt0242653,"The Matrix Revolutions",2003,Movie;Sequel
```

The actors' data in `actors.csv` consist of an `ID` and a `name`, stored as **properties** in the node. The ID in this case is a shorthand for the actor’s name. All the nodes in `actors.csv` have the label `Actor`.

actors.csv

```
personId:ID,name,:LABEL
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
```

The roles data in `roles.csv` have only one **property**, `role`. Roles are represented by relationship data that connects actor nodes with movie nodes.

There are three mandatory fields for relationship data:

1. `:START_ID` — ID referring to a node.
2. `:END_ID` — ID referring to a node.
3. `:TYPE` — The relationship type.

To create a relationship between two nodes, the IDs defined in `actors.csv` and `movies.csv` are used for the `:START_ID` and `:END_ID` fields. You also need to provide a relationship type (in this case `ACTED_IN`) for the `:TYPE` field.

roles.csv

```
:START_ID,role,:END_ID,:TYPE
keanu,"Neo",tt0133093,ACTED_IN
keanu,"Neo",tt0234215,ACTED_IN
keanu,"Neo",tt0242653,ACTED_IN
laurence,"Morpheus",tt0133093,ACTED_IN
laurence,"Morpheus",tt0234215,ACTED_IN
laurence,"Morpheus",tt0242653,ACTED_IN
carrieanne,"Trinity",tt0133093,ACTED_IN
carrieanne,"Trinity",tt0234215,ACTED_IN
carrieanne,"Trinity",tt0242653,ACTED_IN
```

Importing the data

- Paths to node data are defined with the `--nodes` option.
- Paths to relationship data is defined with the `--relationships` option.

The call to `neo4j-admin database import` would look like this:

shell

```
bin/neo4j-admin database import full --nodes=import/movies.csv --nodes=import/actors.csv
--relationships=import/roles.csv neo4j
```

Query the data

To query the data. Start Neo4j.

The default username and password are `neo4j` and `neo4j`.

shell

```
bin/neo4j start
```

To query the imported data in the graph, try a simple Cypher query.

shell

```
bin/cypher-shell --database=neo4j "MATCH (n) RETURN count(n) as nodes"
```

Stop Neo4j.

shell

```
bin/neo4j stop
```

CSV file delimiters

You can customize the configuration options that the import tool uses (see Options) if your data does not fit the default format.

The details of a CSV file header format can be found at CSV header format.

The data

The following CSV files have:

- `--delimiter=";"`
- `--array-delimiter="|"`
- `--quote="'"`

movies2.csv

```
movieId:ID;title;year:int;:LABEL
tt0133093;'The Matrix';1999;Movie
tt0234215;'The Matrix Reloaded';2003;Movie|Sequel
tt0242653;'The Matrix Revolutions';2003;Movie|Sequel
```

actors2.csv

```
personId:ID;name;:LABEL
keanu;'Keanu Reeves';Actor
laurence;'Laurence Fishburne';Actor
carrieanne;'Carrie-Anne Moss';Actor
```

roles2.csv

```
:START_ID;role;:END_ID;:TYPE
keanu;'Neo';tt0133093;ACTED_IN
keanu;'Neo';tt0234215;ACTED_IN
keanu;'Neo';tt0242653;ACTED_IN
laurence;'Morpheus';tt0133093;ACTED_IN
laurence;'Morpheus';tt0234215;ACTED_IN
laurence;'Morpheus';tt0242653;ACTED_IN
carrieanne;'Trinity';tt0133093;ACTED_IN
carrieanne;'Trinity';tt0234215;ACTED_IN
carrieanne;'Trinity';tt0242653;ACTED_IN
```

Importing the data

The call to `neo4j-admin database import` would look like this:

shell

```
bin/neo4j-admin database import full --delimiter=";" --array-delimiter="|" --quote="'"
--nodes=import/movies2.csv --nodes=import/actors2.csv --relationships=import/roles2.csv neo4j
```

Using separate header files

When dealing with very large CSV files, it is more convenient to have the header in a separate file. This makes it easier to edit the header as you avoid having to open a huge data file just to change it. The header file must be specified before the rest of the files in each file group.

The import tool can also process single-file compressed archives, for example:

- `--nodes=import/nodes.csv.gz`
- `--relationships=import/relationships.zip`

The data

You will use the same data set as in the previous example but with the headers in separate files.

movies3-header.csv

```
movieId:ID,title,year:int,:LABEL
```

movies3.csv

```
tt0133093,"The Matrix",1999,Movie
tt0234215,"The Matrix Reloaded",2003,Movie;Sequel
tt0242653,"The Matrix Revolutions",2003,Movie;Sequel
```

actors3-header.csv

```
personId:ID,name,:LABEL
```

actors3.csv

```
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
```

roles3-header.csv

```
:START_ID,role,:END_ID,:TYPE
```

roles3.csv

```
keanu,"Neo",tt0133093,ACTED_IN
keanu,"Neo",tt0234215,ACTED_IN
keanu,"Neo",tt0242653,ACTED_IN
laurence,"Morpheus",tt0133093,ACTED_IN
laurence,"Morpheus",tt0234215,ACTED_IN
laurence,"Morpheus",tt0242653,ACTED_IN
carrieanne,"Trinity",tt0133093,ACTED_IN
carrieanne,"Trinity",tt0234215,ACTED_IN
carrieanne,"Trinity",tt0242653,ACTED_IN
```

Importing the data

The call to `neo4j-admin database import` would look as follows:

The header line for a file group, whether it is the first line of a file in the group or a dedicated header file, must be the *first* line in the file group.

shell

```
bin/neo4j-admin database import full --nodes=import/movies3-header.csv,import/movies3.csv
--nodes=import/actors3-header.csv,import/actors3.csv --relationships=import/roles3
-header.csv,import/roles3.csv neo4j
```

Multiple input files

In addition to using a separate header file, you can also provide multiple node or relationship files. Files within such an input group can be specified with multiple match strings, delimited by `,`, where each matched string can be either the exact file name or a regular expression matching one or more files. Multiple matching files will be sorted according to their characters and their natural number sort order for file names containing numbers.

The data

movies4-header.csv

```
movieId:ID,title,year:int,:LABEL
```

movies4-part1.csv

```
tt0133093,"The Matrix",1999,Movie
tt0234215,"The Matrix Reloaded",2003,Movie;Sequel
```

movies4-part2.csv

```
tt0242653,"The Matrix Revolutions",2003,Movie;Sequel
```

actors4-header.csv

```
personId:ID,name,:LABEL
```

actors4-part1.csv

```
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
```

actors4-part2.csv

```
carrieanne,"Carrie-Anne Moss",Actor
```

roles4-header.csv

```
:START_ID,role,:END_ID,:TYPE
```

roles4-part1.csv

```
keanu,"Neo",tt0133093,ACTED_IN
keanu,"Neo",tt0234215,ACTED_IN
keanu,"Neo",tt0242653,ACTED_IN
laurence,"Morpheus",tt0133093,ACTED_IN
laurence,"Morpheus",tt0234215,ACTED_IN
```

roles4-part2.csv

```
laurence,"Morpheus",tt0242653,ACTED_IN
carrieanne,"Trinity",tt0133093,ACTED_IN
carrieanne,"Trinity",tt0234215,ACTED_IN
carrieanne,"Trinity",tt0242653,ACTED_IN
```

Importing the data

The call to `neo4j-admin database import` would look like this:

shell

```
bin/neo4j-admin database import full --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 neo4j
```

Regular expressions

File names can be specified using regular expressions when there are many data source files. Each file name that matches the regular expression will be included.

If using separate header files, for the import to work correctly, the header file must be the first in the file group. When using regular expressions to specify the input files, the list of files will be sorted according to the names of the files that match the expression. The matching is aware of the numbers inside the file names and will sort them accordingly, without the need for padding with zeros.

Example 128. Match order

For example, let's assume that you have the following files:

- `movies4-header.csv`
- `movies4-data1.csv`
- `movies4-data2.csv`
- `movies4-data12.csv`

If you use the regular expression `movies4.*`, the sorting will place the header file last and the import will fail. A better alternative would be to name the header file explicitly and use a regular expression that only matches the names of the data files. For example: `--nodes "import/movies4-header.csv,movies-data.*"` will accomplish this.

Importing the data using regular expressions, the call to `neo4j-admin database import` can be simplified to:

shell

```
bin/neo4j-admin database import full --nodes="import/movies4-header.csv,import/movies4-part.*"
--nodes="import/actors4-header.csv,import/actors4-part.*" --relationships="import/roles4
-header.csv,import/roles4-part.*" neo4j
```

The use of regular expressions should not be confused with file globbing.

The expression `.*` means: "zero or more occurrences of any character except line break". Therefore, the regular expression `movies4.*` will list all files starting with `movies4`. Conversely, with file globbing, `ls movies4.*` will list all files starting with `movies4.`.

Another important difference to pay attention to is the sorting order. The result of a regular expression matching will place the file `movies4-part2.csv` before the file `movies4-part12.csv`. If doing `ls movies4-part*` in a directory containing the above-listed files, the file `movies4-part12.csv` will be listed before the file `movies4-part2.csv`.

Using the same label for every node

If you want to use the same node label(s) for every node in your nodes file you can do this by specifying the appropriate value as an option to `neo4j-admin database import`. There is then no need to specify the `:LABEL` column in the header file and each row (node) will apply the specified labels from the command line option.

Example 129. 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 database import` would look like this:

shell

```
bin/neo4j-admin database import full --nodes=Movie=import/movies5a.csv
--nodes=Movie:Sequel=import/sequels5a.csv --nodes=Actor=import/actors5a.csv
--relationships=import/roles5a.csv
```

Using the same relationship type for every relationship

If you want to use the same relationship type for every relationship in your relationships file this can be done by specifying the appropriate value as an option to `neo4j-admin database import`.

Example 130. 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 database import` would look like this:

shell

```
bin/neo4j-admin database import full --nodes=import/movies5b.csv --nodes=import/actors5b.csv
--relationships=ACTED_IN=import/roles5b.csv neo4j
```

Properties

Nodes and relationships can have properties. The property type is specified in the CSV header row, see CSV header format.

The data

The following example creates a small graph containing one actor and one movie connected by one relationship.

There is a `roles` property on the relationship which contains an array of the characters played by the actor in a movie:

movies6.csv

```
movieId:ID,title,year:int,:LABEL
tt0099892,"Joe Versus the Volcano",1990,Movie
```

actors6.csv

```
personId:ID,name,:LABEL
meg,"Meg Ryan",Actor
```

roles6.csv

```
:START_ID,roles:string[],:END_ID,:TYPE
meg,"DeDe;Angelica Graynamore;Patricia Graynamore",tt0099892,ACTED_IN
```

Importing the data

The call to `neo4j-admin database import` would look like this:

shell

```
bin/neo4j-admin database import full --nodes=import/movies6.csv --nodes=import/actors6.csv
--relationships=import/roles6.csv neo4j
```

ID space

The import tool assumes that identifiers are unique across node files. This may not be the case for data sets that use sequential, auto-incremented, or otherwise colliding identifiers. Those data sets can define ID spaces where identifiers are unique within their respective ID space.

In cases where the node ID is unique only within files, using ID spaces is a way to ensure uniqueness across all node files. See Using ID spaces.

Each node processed by `neo4j-admin database import` must provide an ID if it is to be connected in any relationships. The node ID is used to find the start node and end node when creating a relationship.

From Neo4j v5.3, a node header can also contain multiple `ID` columns, where the relationship data references the composite value of all those columns. This also implies using `string` as `id-type`. For each `ID` column, you can specify to store its values as different node properties. However, the composite value cannot be stored as a node property.

Example 131. ID space

To define an ID space `Movie-ID` for `movieId:ID`, use the syntax `movieId:ID(Movie-ID)`.

The data

For example, if movies and people both use sequential identifiers, then you would define `Movie` and `Actor` ID spaces.

movies7.csv

```
movieId:ID(Movie-ID),title,year:int,:LABEL
1,"The Matrix",1999,Movie
2,"The Matrix Reloaded",2003,Movie;Sequel
3,"The Matrix Revolutions",2003,Movie;Sequel
```

actors7.csv

```
personId:ID(Actor-ID),name,:LABEL
1,"Keanu Reeves",Actor
2,"Laurence Fishburne",Actor
3,"Carrie-Anne Moss",Actor
```

You also need to reference the appropriate ID space in your relationships file so it knows which nodes to connect.

roles7.csv

```
:START_ID(Actor-ID),role,:END_ID(Movie-ID)
1,"Neo",1
1,"Neo",2
1,"Neo",3
2,"Morpheus",1
2,"Morpheus",2
2,"Morpheus",3
3,"Trinity",1
3,"Trinity",2
3,"Trinity",3
```

Importing the data

The call to `neo4j-admin database import` would look like this:

shell

```
bin/neo4j-admin database import full --nodes=import/movies7.csv --nodes=import/actors7.csv
--relationships=ACTED_IN=import/roles7.csv neo4j
```

Skip relationships referring to missing nodes

The import tool has no tolerance for bad entities (relationships or nodes) and will fail the import on the first bad entity. You can specify explicitly that you want it to ignore rows that contain bad entities.

There are two different types of bad input:

1. Bad relationships.
2. Bad nodes.

Relationships that refer to missing node IDs, either for `:START_ID` or `:END_ID` are considered bad relationships. Whether or not such relationships are skipped is controlled with the `--skip-bad-relationships` flag, which can have the values `true` or `false` or no value, which means `true`. The default is `false`, which means that any bad relationship is considered an error and will fail the import. For more information, see the `--skip-bad-relationships` option.

The data

In the following example, there is a missing `emil` node referenced in the roles file.

movies8a.csv

```
movieId:ID,title,year:int,:LABEL
tt0133093,"The Matrix",1999,Movie
tt0234215,"The Matrix Reloaded",2003,Movie;Sequel
tt0242653,"The Matrix Revolutions",2003,Movie;Sequel
```

actors8a.csv

```
personId:ID,name,:LABEL
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
```

roles8a.csv

```
:START_ID,role,:END_ID,:TYPE
keanu,"Neo",tt0133093,ACTED_IN
keanu,"Neo",tt0234215,ACTED_IN
keanu,"Neo",tt0242653,ACTED_IN
laurence,"Morpheus",tt0133093,ACTED_IN
laurence,"Morpheus",tt0234215,ACTED_IN
laurence,"Morpheus",tt0242653,ACTED_IN
carrieanne,"Trinity",tt0133093,ACTED_IN
carrieanne,"Trinity",tt0234215,ACTED_IN
carrieanne,"Trinity",tt0242653,ACTED_IN
emil,"Emil",tt0133093,ACTED_IN
```

Importing the data

The call to `neo4j-admin database import` would look like this:

shell

```
bin/neo4j-admin database import full --nodes=import/movies8a.csv --nodes=import/actors8a.csv
--relationships=import/roles8a.csv neo4j
```

Because there is a bad relationship in the input data, the import process will fail.

Let's see what happens if you append the `--skip-bad-relationships` flag:

shell

```
bin/neo4j-admin database import full --skip-bad-relationships --nodes=import/movies8a.csv
--nodes=import/actors8a.csv --relationships=import/roles8a.csv neo4j
```

The data files are successfully imported and the bad relationship is ignored. An entry is written to the `import.report` file.

ignore bad relationships

```
InputRelationship:
   source: roles8a.csv:11
   properties: [role, Emil]
   startNode: emil (global id space)
   endNode: tt0133093 (global id space)
   type: ACTED_IN
 referring to missing node emil
```

Skip nodes with the same ID

Nodes that specify `:ID`, which has already been specified within the ID space are considered bad nodes. Whether or not such nodes are skipped is controlled with `--skip-duplicate-nodes` flag which can have the values `true` or `false` or no value, which means `true`. The default is `false`, which means that any duplicate node is considered an error and will fail the import. For more information, see the `--skip-duplicate-nodes` option.

The data

In the following example there is a node ID, `laurence`, that is specified twice within the same ID space.

actors8b.csv

```
personId:ID,name,:LABEL
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
laurence,"Laurence Harvey",Actor
```

Importing the data

The call to `neo4j-admin database import` would look like this:

shell

```
bin/neo4j-admin database import full --database=neo4j --nodes=import/actors8b.csv neo4j
```

Because there is a bad node in the input data, the import process will fail.

Let's see what happens if you append the `--skip-duplicate-nodes` flag:

shell

```
bin/neo4j-admin database import full --skip-duplicate-nodes --nodes=import/actors8b.csv neo4j
```

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
```

Set up and use a composite database

Composite databases allow queries that access multiple graphs at once. This is a function that enables:

- **Data Federation**: the ability to access data available in distributed sources in the form of **disjoint graphs**.
- **Data Sharding**: the ability to access data available in distributed sources in the form of a **common graph partitioned on multiple databases.**

In this tutorial, you will learn how to:

- Model your data for composite database use
- Create databases for the composite
- Import data to your databases
- Configure a composite database
- Retrieve data with a single Cypher query

Model your data for composite database use

The example data in this tutorial is based on the Northwind dataset, created by Microsoft. It contains the sales data of a fictitious small company called “Northwind Traders”. The data includes customers, products, customer orders, warehouse stock, shipping, suppliers, employees, and sales territories.

For more information on how Northwind (a relational dataset) is modeled into a graph, run `:guide northwind-graph` in Neo4j Browser to play the built-in guide Northwind Graph. See the Neo4j Browser documentation.

The Northwind graph model consists of the following data:

- Node labels
  - `:Product`
  - `:Category`
  - `:Supplier`
  - `:Order`

    - `:Customer`
- Relationship types
    - `:SUPPLIES`
    - `:PART_OF`
    - `:ORDERS`
    - `:PURCHASED`

Figure 9. The Northwind data model

In this scenario, assume that data privacy constraints require customers' data to be stored in their original region. For simplicity, there are two regions: the Americas (AME) and Europe (EU). The first step is to remodel the Northwind dataset, so that customer data can be separated from the Product catalog, which has no privacy constraints. You create two graphs: one for the Product catalog, which includes `:Product`, `:Category`, `:Supplier`, `:PART_OF`, `:SUPPLIES`, and one partitioned graph in two databases for the Customer orders in EU and AME, with `:Product`, `:Order`, `:Customer`, `:PURCHASED`, and `:ORDERS`.

Figure 10. The new data model

Data Federation

This way, the Product and Customer data are in two **disjoint graphs**, with different labels and relationship types. This is called *Data Federation*.

To query across them, you have to federate the graphs, because relationships cannot span across them. This is done by using a *proxy node* modeling pattern: nodes with the `:Product` label must be present in both federated domains.

In the Product catalog graph, nodes with the `:Product` label contain all the data related to a product, while in the Customer graphs, the same label is associated to a proxy node which only contains `productID`. The `productID` property allows you to link data across the graphs in this federation.

Figure 11. 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 12. Data Sharding

In general, there are a couple of main use cases that require sharding. The most common is scalability, i.e. different shards can be deployed on different servers, splitting the load on different resources. Another reason could be data regulations: different shards can be deployed on servers, residing in different locations, and managed independently.

Create databases for the composite

For this tutorial, you need to create three databases `db0` for the Product catalog, `db1` for the EU customer data, and `db2` for the AME customers by following these steps:

1. Start the Neo4j DBMS.

```
bin/neo4j start
```

2. Check all available databases.

```
ls -al /data/databases/
```

```
total 0
drwxr-xr-x@  5 username  staff   160  9 Jun 12:53 .
drwxr-xr-x@  5 username  staff   160  9 Jun 12:53 ..
drwxr-xr-x  37 username  staff  1184  9 Jun 12:53 neo4j
-rw-r--r--   1 username  staff     0  9 Jun 12:53 store_lock
drwxr-xr-x  38 username  staff  1216  9 Jun 12:53 system
```

3. Connect to the Neo4j DBMS using `cypher-shell` with the default credentials and change the password when prompted:

```
bin/cypher-shell -u neo4j -p neo4j
```

```
Password change required
new password: *****
Connected to Neo4j 5 at neo4j://localhost:7687 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
```

> For more information about the Cypher Shell command-line interface (CLI) and how to use it, see [cypher-shell].

4. Run the command `SHOW DATABASES` to list all available databases:

```
SHOW DATABASES;
```

```
+-----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------+
| name     | type       | aliases | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage | default | home  | constituents |
+-----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------+
| "neo4j"  | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"       | ""             | TRUE     | TRUE  | []            |
| "system" | "system"   | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"       | ""             | FALSE    | FALSE | []            |
+-----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------+
2 rows available after 102 ms, consumed after another 11 ms
```

5. Run the command `CREATE DATABASE <database-name>` to create the databases:

```
CREATE DATABASE db0;
```

```
0 rows available after 137 ms, consumed after another 0 ms
```

```
CREATE DATABASE db1;
```

```
0 rows available after 141 ms, consumed after another 0 ms
```

```
CREATE DATABASE db2;
```

```
0 rows available after 135 ms, consumed after another 0 ms
```

6. Run the command `SHOW DATABASES` again to verify that the new databases have been created and are `online`:

```
SHOW DATABASES;
```

```
+-------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------+
| name     | type       | aliases | access       | address          | role      | writer |
requestedStatus | currentStatus | statusMessage | default | home   | constituents |
+-------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------+
| "db0"    | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | FALSE    | FALSE | []           |
| "db1"    | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | FALSE    | FALSE | []           |
| "db2"    | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | FALSE    | FALSE | []           |
| "neo4j"  | "standard" | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | TRUE     | TRUE  | []           |
| "system" | "system"   | []      | "read-write" | "localhost:7687" | "primary" | TRUE   | "online"
| "online"      | ""            | FALSE    | FALSE | []           |
+-------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------+
5 rows available after 8 ms, consumed after another 7 ms
```

Import data to your databases

You can use the command `LOAD CSV WITH HEADERS FROM` to import data to the databases.

Load the Product catalog in db0

1. Run the following Cypher query to change the active database to `db0`, and add the Product data:

```
:use db0;

LOAD CSV WITH HEADERS FROM "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
```

Configure a composite database

Set up a composite database with the `CREATE COMPOSITE DATABASE` Cypher command and add local database aliases as constituents to the composite database. In this example, the composite database is called `compositenw`.

1. Run the command `CREATE COMPOSITE DATABASE <composite-database-name>` to create the composite database:

```
CREATE COMPOSITE DATABASE compositenw;
```

```
0 rows available after 137 ms, consumed after another 0 ms
```

2. Run the command `CREATE ALIAS <composite-database-name>.<alias-name> FOR DATABASE <database-name>` to create the constituent database aliases:

```
CREATE ALIAS compositenw.product FOR DATABASE db0;
```

```
0 rows available after 101 ms, consumed after another 0 ms
```

```
CREATE ALIAS compositenw.customerEU FOR DATABASE db1;
```

```
0 rows available after 107 ms, consumed after another 0 ms
```

```
CREATE ALIAS compositenw.customerAME FOR DATABASE db2;
```

```
0 rows available after 98 ms, consumed after another 0 ms
```

The constituent database aliases in this tutorial are local database aliases (pointing to databases in the same Neo4j DBMS), but they can just as well be remote database aliases (pointing to databases in another Neo4j DBMS).

3. Run the command `SHOW DATABASES` to verify that the composite database has been configured and is `online`:

```
SHOW DATABASES;
```

```
+-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------+
| name          | type        | aliases                       | access       | address           | role
| writer | requestedStatus | currentStatus | statusMessage | default | home  | constituents
|
+-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------+
| "db0"         | "standard"  | ["compositenw.product"]       | "read-write" | "localhost:7687" |
"primary" | TRUE   | "online"        | "online"      | ""            | FALSE   | FALSE | []
|
| "db1"         | "standard"  | ["compositenw.customerEU"]    | "read-write" | "localhost:7687" |
"primary" | TRUE   | "online"        | "online"      | ""            | FALSE   | FALSE | []
|
| "db2"         | "standard"  | ["compositenw.customerAME"] | "read-write" | "localhost:7687" |
"primary" | TRUE   | "online"        | "online"      | ""            | FALSE   | FALSE | []
|
| "compositenw" | "composite" | []                            | "read-only"  | "localhost:7687" |
"primary" | FALSE  | "online"        | "online"      | ""            | FALSE   | FALSE |
["compositenw.customerAME", "compositenw.customerEU", "compositenw.product"] |
| "neo4j"       | "standard"  | []                            | "read-write" | "localhost:7687" |
"primary" | TRUE   | "online"        | "online"      | ""            | TRUE    | TRUE  | []
|
| "system"      | "system"    | []                            | "read-write" | "localhost:7687" |
"primary" | TRUE   | "online"        | "online"      | ""            | FALSE   | FALSE | []
|
+-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------+
6 rows available after 242 ms, consumed after another 18 ms
```

4. Run the command `SHOW ALIASES FOR DATABASES` to verify that the database aliases have been configured:

```
SHOW ALIASES FOR DATABASES;
```

```
+-------------------------------------------------------------------+
| name                        | database  | location | url  | user |
+-------------------------------------------------------------------+
| "compositenw.product"       | "db0"     | "local"  | null | null |
| "compositenw.customerEU"    | "db1"     | "local"  | null | null |
| "compositenw.customerAME"   | "db2"     | "local"  | null | null |
+-------------------------------------------------------------------+
3 rows available after 203 ms, consumed after another 16 ms
```

Retrieve data with a single Cypher query

Query a single database

When connected to a composite database you can retrieve data from a single database by using the Cypher clause USE and the name of an alias:

```
:use compositenw
```

```
USE compositenw.product
MATCH (p:Product)
RETURN p.productName AS product
LIMIT 5;
```

```
+--------------------------------+
| product                        |
+--------------------------------+
| "Chai"                         |
| "Chang"                        |
| "Aniseed Syrup"                |
| "Chef Anton's Cajun Seasoning" |
| "Chef Anton's Gumbo Mix"       |
+--------------------------------+

5 rows available after 6 ms, consumed after another 21 ms
```

Query across multiple shards

Use the composite database to query both shards and get customers whose name starts with A:

```
:use compositenw
```

```
USE compositenw.customerAME
MATCH (c:Customer)
WHERE c.customerID STARTS WITH 'A'
RETURN c.customerID AS name, c.country AS country
  UNION
USE compositenw.customerEU
MATCH (c:Customer)
WHERE c.customerID STARTS WITH 'A'
RETURN c.customerID AS name, c.country AS country
LIMIT 5;
```

```
+---------------------+
| name    | country   |
+---------------------+
| "ANATR" | "Mexico"  |
| "ANTON" | "Mexico"  |
| "ALFKI" | "Germany" |
| "AROUT" | "UK"      |
+---------------------+

4 rows available after 25 ms, consumed after another 56 ms
```

Or, using a more common composite database idiom:

```
:use compositenw
```

```
UNWIND ['compositenw.customerAME', 'compositenw.customerEU'] AS g
CALL {
  USE graph.byName(g)
  MATCH (c:Customer)
  WHERE c.customerID STARTS WITH 'A'
  RETURN c.customerID AS name, c.country AS country
}
RETURN name, country
LIMIT 5;
```

```
+---------------------+
| name    | country   |
+---------------------+
| "ANATR" | "Mexico"  |
| "ANTON" | "Mexico"  |
| "ALFKI" | "Germany" |
| "AROUT" | "UK"      |
+---------------------+

4 rows available after 61 ms, consumed after another 8 ms
```

Query across federation and shards

Here is a more complex query that uses all 3 databases to find all customers who have bought discontinued products in the Meat/Poultry category:

```
:use compositenw
```

```
CALL {
  USE compositenw.product
  MATCH (p:Product)-[:PART_OF]->(c:Category)
  WHERE p.discontinued = true
    AND c.categoryName = 'Meat/Poultry'
  RETURN COLLECT(p.productID) AS pids
}
WITH *
UNWIND [g IN graph.names() WHERE g STARTS WITH 'compositenw.customer'] AS g
CALL {
  USE graph.byName(g)
  WITH pids
  UNWIND pids as pid
  MATCH (p:Product{productID:pid})<-[:ORDERS]-(:Order)<-[:PURCHASED]-(c:Customer)
  RETURN DISTINCT c.customerID AS customer, c.country AS country
}
RETURN customer, country
LIMIT 20;
```

```
+--------------------------+
| customer | country       |
+--------------------------+
| "RICSU"  | "Switzerland" |
| "PERIC"  | "Mexico"      |
| "WARTH"  | "Finland"     |
| "WELLI"  | "Brazil"      |
| "DRACD"  | "Germany"     |
| "RATTC"  | "USA"         |
| "HUNGO"  | "Ireland"     |
| "QUEDE"  | "Brazil"      |
| "SEVES"  | "UK"          |
| "ANTON"  | "Mexico"      |
| "BERGS"  | "Sweden"      |
| "SAVEA"  | "USA"         |
| "AROUT"  | "UK"          |
| "FAMIA"  | "Brazil"      |
| "WANDK"  | "Germany"     |
| "WHITC"  | "USA"         |
| "ISLAT"  | "UK"          |
| "LONEP"  | "USA"         |
| "QUICK"  | "Germany"     |
| "HILAA"  | "Venezuela"   |
+--------------------------+

20 rows available after 51 ms, consumed after another 2 ms
```

The way this query works is by `compositenw` calling database `db0` to retrieve all discontinued products in the Meat/Poultry category. Then, using the returned product IDs, it queries both `db1` and `db2` **in parallel** and gets the customers who have purchased these products and their country.

You have just learned how to store and retrieve data from multiple databases using a single Cypher query.

+ For more details on composite databases, see [composite-databases].

Neo4j Single Sign-On (SSO) configuration

Neo4j supports SSO authentication and authorization through identity providers implementing the OpenID Connect (OIDC) standard. This page features detailed examples of how to configure Single Sign-On (SSO) for several identity providers. It also presents frequently asked questions and solutions to common problems encountered when configuring SSO.

The following configurations are crafted for a Neo4j Browser served on `http://localhost:7474/browser/` (the default URL when starting the database on `localhost`).

Therefore, when reproducing them in the identity providers, you must modify the redirect URI to include the URI serving your Neo4j Browser application. For example:

<code>http://localhost:7474/browser/?idp_id={provider}&auth_flow_step=redirect_uri</code>

SSO works in the following way:

1. The server (Neo4j DBMS) contacts the identity provider (Okta, Azure, Google, etc.) and fetches the JSON Web Keys (JWKs) from the provider.
2. The client (e.g., Bloom, Neo4j Browser, etc.) asks the user for credentials and contacts the identity

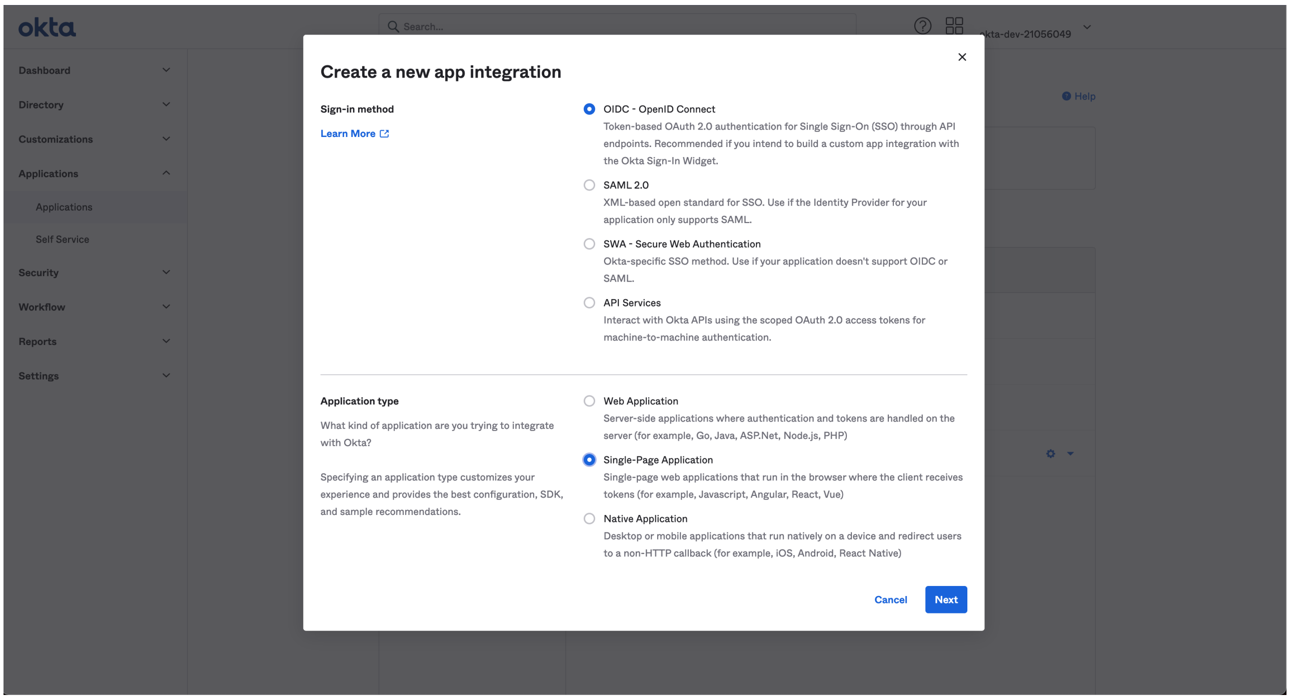
provider.

3. The identity provider responds with a JSON Web Token (JWT), a JSON file containing fields (claims) relative to the user (email, audience, groups, etc.).
4. The client provides the server with the JWT, and the server verifies its signature with the JWKs.

Okta

This example shows how to configure Okta for authentication and authorization using access tokens.

1. Configure the client with the appropriate redirect URI. You can skip the group assignments in this step:



Create a new app integration

Sign-in method

Learn More

[✓ OIDC - OpenID Connect]
Token-based OAuth 2.0 authentication for Single Sign-On (SSO) through API endpoints. Recommended if you intend to build a custom app integration with the Okta Sign-In Widget.

SAML 2.0
XML-based open standard for SSO. Use if the Identity Provider for your application only supports SAML.

SWA - Secure Web Authentication
Okta-specific SSO method. Use if your application doesn't support OIDC or SAML.

API Services
Interact with Okta APIs using the scoped OAuth 2.0 access tokens for machine-to-machine authentication.

Application type

What kind of application are you trying to integrate with Okta?

Specifying an application type customizes your experience and provides the best configuration, SDK, and sample recommendations.

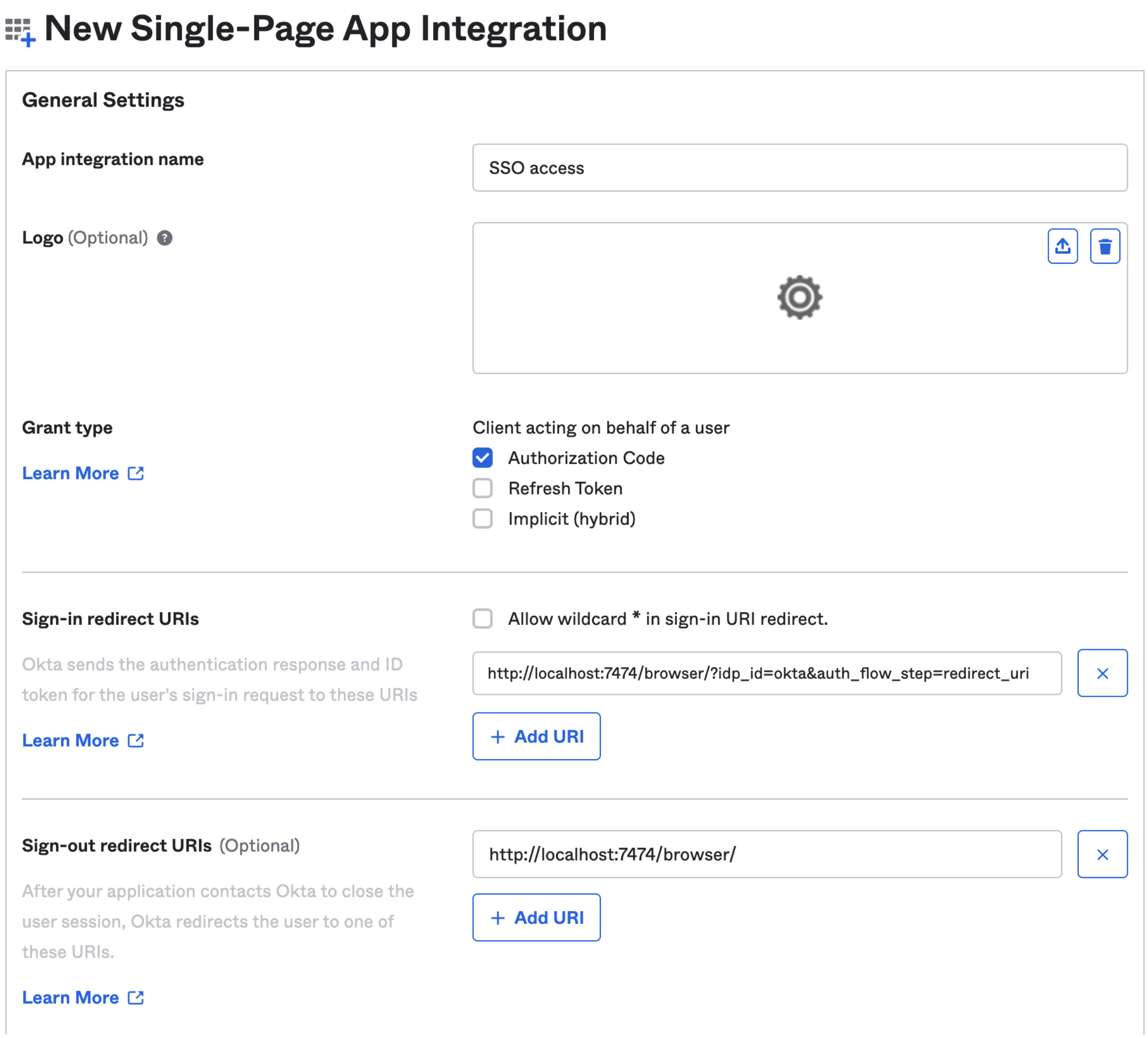
Web Application
Server-side applications where authentication and tokens are handled on the server (for example, Go, Java, ASP.Net, Node.js, PHP)

[✓ Single-Page Application]
Single-page web applications that run in the browser where the client receives tokens (for example, Javascript, Angular, React, Vue)

Native Application
Desktop or mobile applications that run natively on a device and redirect users to a non-HTTP callback (for example, iOS, Android, React Native)

Cancel Next

Figure 13. Okta OIDC client creation



New Single-Page App Integration

General Settings

App integration name: SSO access

Logo (Optional)

Grant type

Learn More

Client acting on behalf of a user
- [x] Authorization Code
- [ ] Refresh Token
- [ ] Implicit (hybrid)

Sign-in redirect URIs

Okta sends the authentication response and ID token for the user's sign-in request to these URIs

Learn More

- [ ] Allow wildcard * in sign-in URI redirect.

http://localhost:7474/browser/?idp_id=okta&auth_flow_step=redirect_uri

+ Add URI

Sign-out redirect URIs (Optional)

After your application contacts Okta to close the user session, Okta redirects the user to one of these URIs.

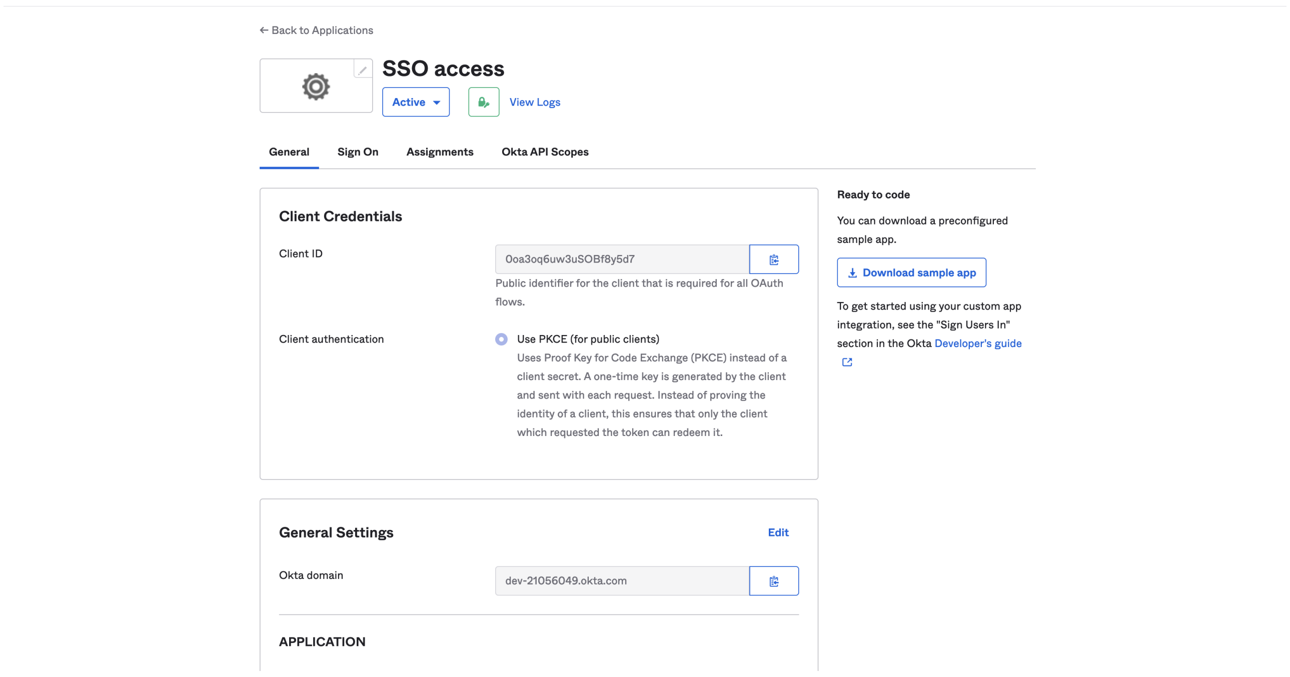
Learn More

http://localhost:7474/browser/

+ Add URI

Figure 14. Okta OIDC client configuration

2. Take note of the Client ID and the Okta domain. You will need them later when configuring the Okta parameters and the Well-known OpenID Connect endpoint in the *neo4j.conf* file:



← Back to Applications

SSO access

Active ▾ View Logs

General | Sign On | Assignments | Okta API Scopes

Client Credentials

Client ID: 0oa3oq6uw3uSOBf8y5d7

Public identifier for the client that is required for all OAuth flows.

Client authentication: ● Use PKCE (for public clients)

Uses Proof Key for Code Exchange (PKCE) instead of a client secret. A one-time key is generated by the client and sent with each request. Instead of proving the identity of a client, this ensures that only the client which requested the token can redeem it.

Ready to code

You can download a preconfigured sample app.

Download sample app

To get started using your custom app integration, see the "Sign Users In" section in the Okta Developer's guide

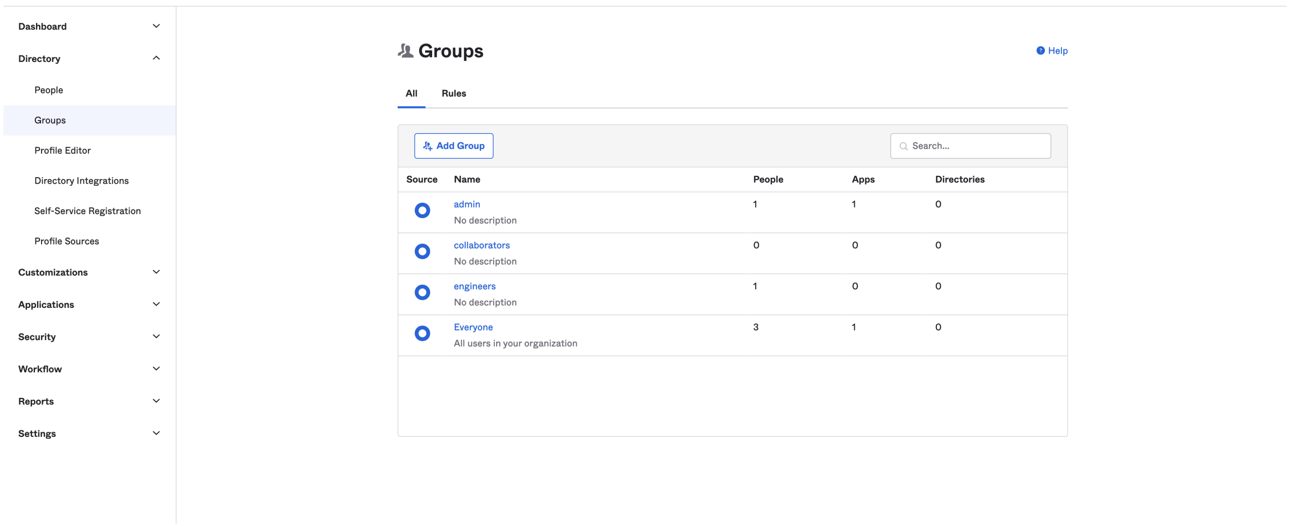
General Settings Edit

Okta domain: dev-21056049.okta.com

APPLICATION

Figure 15. Okta OIDC client configuration

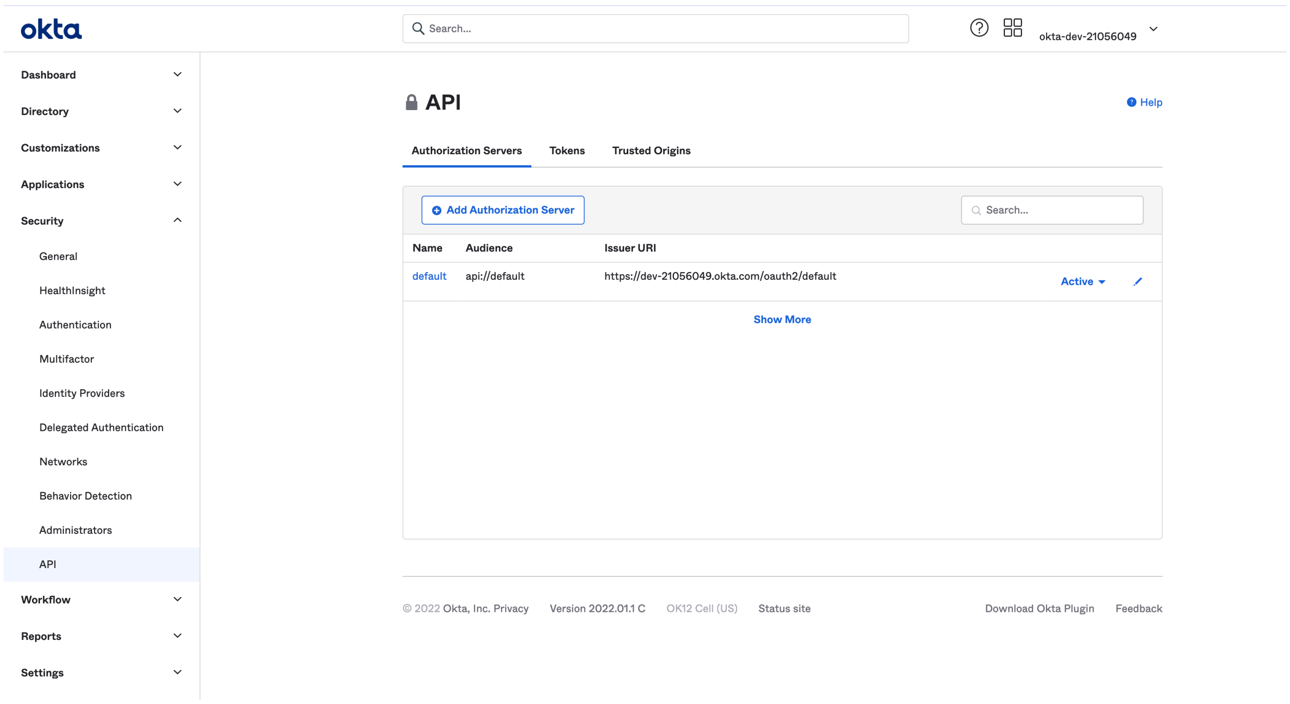
3. Create groups in Okta, assign users to them (the user can be added to a group either on user creation or editing the group), and map them in the `neo4j.conf` to native groups:



Source	Name	People	Apps	Directories
	admin No description	1	1	0
	collaborators No description	0	0	0
	engineers No description	1	0	0
	Everyone All users in your organization	3	1	0

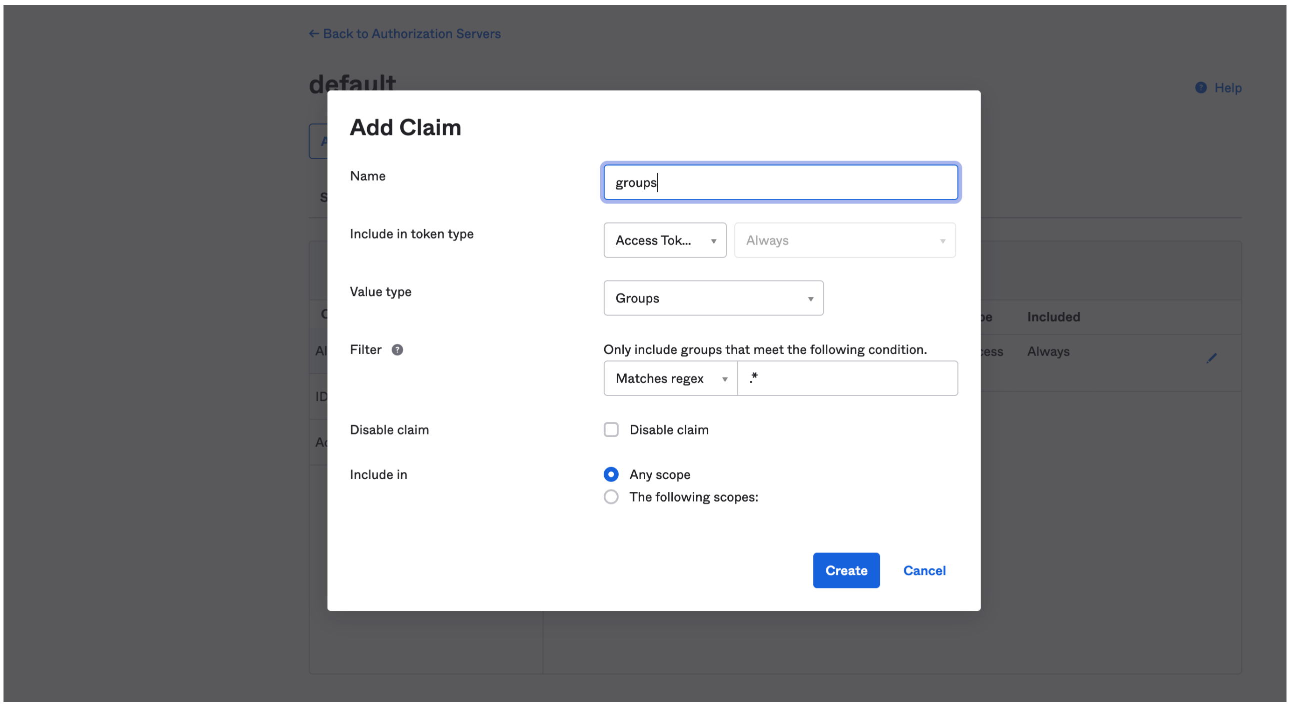
Figure 16. Okta OIDC server groups

4. Configure the default authorization server (the one that shows `api://default` as audience) to return the `groups` claim in access tokens:



Name	Audience	Issuer URI	
default	api://default	https://dev-21056049.okta.com/oauth2/default	Active

Figure 17. Okta OIDC authorization server


Add Claim

Name: groups

Include in token type: Access Tok... | Always

Value type: Groups

Filter: Only include groups that meet the following condition. Matches regex | .*

Disable claim: ☐ Disable claim

Include in: ● Any scope ○ The following scopes:

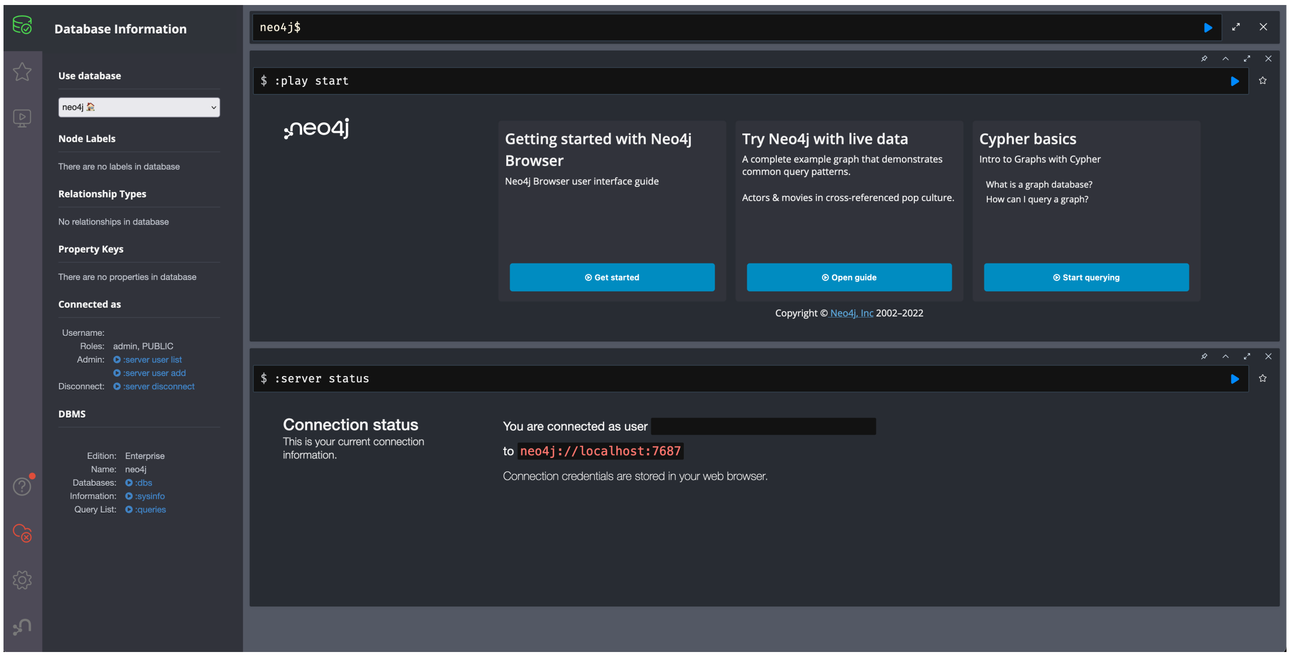
Figure 18. Okta OIDC server claims

5. Configure Neo4j to use Okta authentication by configuring the following settings in the *neo4j.conf* file:

```
dbms.security.authentication_providers=oidc-okta
dbms.security.authorization_providers=oidc-okta
dbms.security.oidc.okta.display_name=Okta
dbms.security.oidc.okta.auth_flow=pkce
dbms.security.oidc.okta.well_known_discovery_uri=https://dev-21056049.okta.com/oauth2/default/.well-
known/openid-configuration
dbms.security.oidc.okta.audience=api://default
dbms.security.oidc.okta.claims.username=sub
dbms.security.oidc.okta.claims.groups=groups
dbms.security.oidc.okta.params=client_id=0oa3oq6uw3uSOBf8y5d7;response_type=code;scope=openid profile
email
dbms.security.oidc.okta.authorization.group_to_role_mapping= "engineers" = admin; \
                                                             "collaborators" = reader
```

The `token_type_principal` and the `token_type_authentication` are omitted, meaning access tokens are used instead.

6. Log in with your Okta SSO credentials using the email of an `engineer` role user that results in an `admin` role in the database:



Database Information

Use database

neo4j 🏠

Node Labels

There are no labels in database

Relationship Types

No relationships in database

Property Keys

There are no properties in database

Connected as

Username:	
Roles:	admin, PUBLIC
Admin:	:server user list
	:server user add
Disconnect:	:server disconnect

DBMS

Edition:	Enterprise
Name:	neo4j
Databases:	:dbs
Information:	:sysinfo
Query List:	:queries

```
neo4j$
```

```
$ :play start
```

Getting started with Neo4j Browser
Neo4j Browser user interface guide

Try Neo4j with live data
A complete example graph that demonstrates common query patterns.
Actors & movies in cross-referenced pop culture.

Cypher basics
Intro to Graphs with Cypher
What is a graph database?
How can I query a graph?

Copyright © Neo4j, Inc 2002–2022

```
$ :server status
```

Connection status
This is your current connection information.

You are connected as user [redacted]
to `neo4j://localhost:7687`
Connection credentials are stored in your web browser.

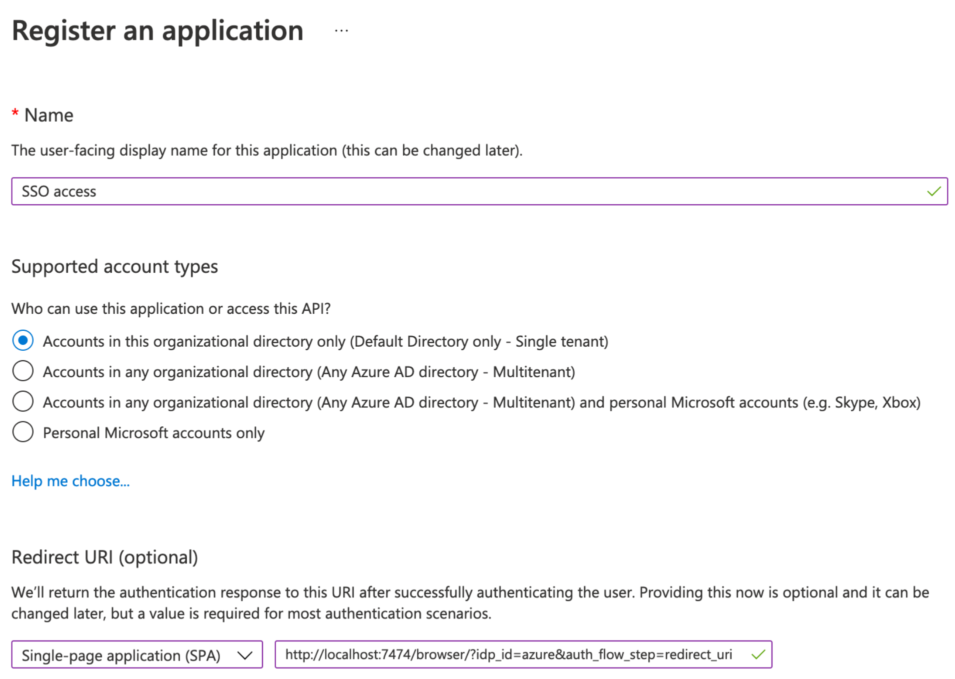
Figure 19. Okta OIDC successful login

Azure Active Directory (AAD)

This example shows how to configure AAD for authentication and authorization using ID tokens.

Register the application

1. Log in to the Azure portal.
2. Navigate to **Azure Active Directory** > **Overview**.
3. From the **Add** dropdown menu, select **App registration** and fill in the following information to create your SSO application:


Register an application

* Name

The user-facing display name for this application (this can be changed later).

SSO access

Supported account types

Who can use this application or access this API?

- [✓ Accounts in this organizational directory only (Default Directory only - Single tenant)]
- Accounts in any organizational directory (Any Azure AD directory - Multitenant)
- Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)
- Personal Microsoft accounts only

Help me choose...

Redirect URI (optional)

We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.

Single-page application (SPA) | http://localhost:7474/browser/?idp_id=azure&auth_flow_step=redirect_uri

Figure 20. Azure OIDC client creation

The redirect URI `http://localhost:7474/browser/?idp_id=azure&auth_flow_step=redirect_uri` is the URI that will accept returned token responses after successful authentication.

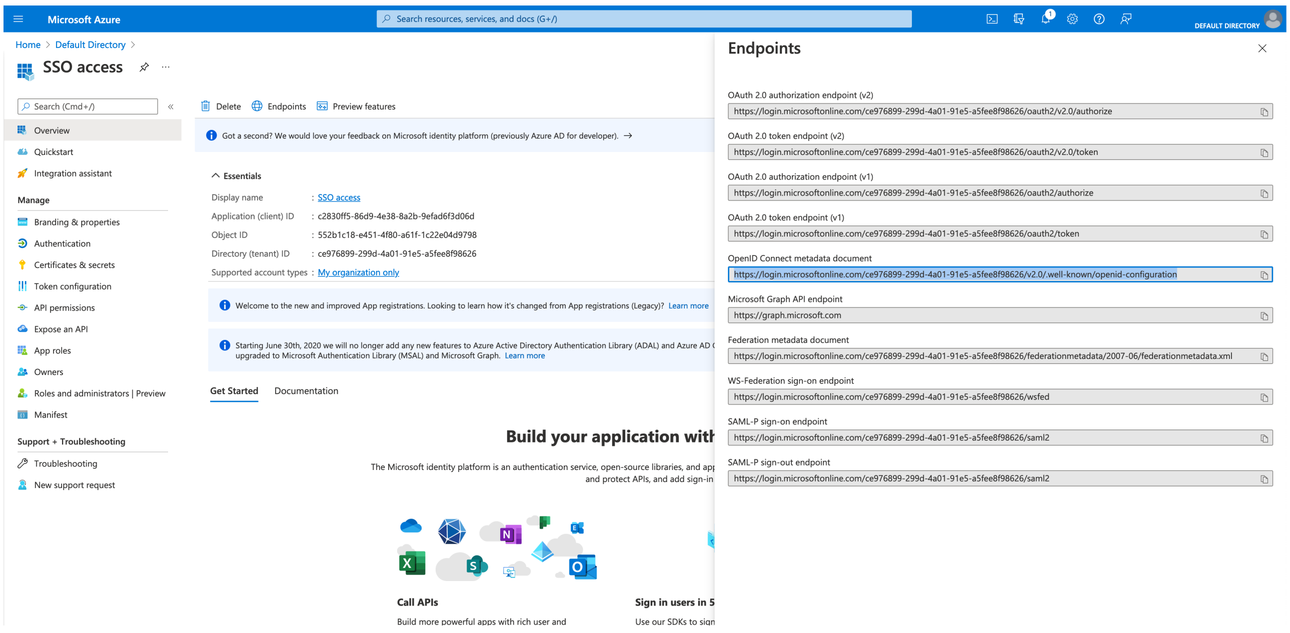
4. Click **Register**.

Configure Neo4j

1. After the successful app creation, on the app's **Overview** page, find the Application (client) ID value. Use it to configure the following properties in the *neo4j.conf* file.

```
dbms.security.oidc.azure.audience=c2830ff5-86d9-4e38-8a2b-9efad6f3d06d
dbms.security.oidc.azure.params=client_id=c2830ff5-86d9-4e38-8a2b-
9efad6f3d06d;response_type=code;scope=openid profile email
```

2. Navigate to **Endpoints**, to find the OpenID Connect metadata document. Use it to configure the `well_known_discovery_uri` in the *neo4j.conf* file.



Got a second? We would love your feedback on Microsoft identity platform (previously Azure AD for developer). →

Essentials

Display name : SSO access
Application (client) ID : c2830ff5-86d9-4e38-8a2b-9efad6f3d06d
Object ID : 552b1c18-e451-4f80-a61f-1c22e04d9798
Directory (tenant) ID : ce976899-299d-4a01-91e5-a5fee8f98626
Supported account types : My organization only

Welcome to the new and improved App registrations. Looking to learn how it's changed from App registrations (Legacy)? Learn more

Starting June 30th, 2020 we will no longer add any new features to Azure Active Directory Authentication Library (ADAL) and Azure AD
upgraded to Microsoft Authentication Library (MSAL) and Microsoft Graph. Learn more

Build your application with

The Microsoft identity platform is an authentication service, open-source libraries, and app
and protect APIs, and add sign-in

Build more powerful apps with rich user and

Use our SDKs to sign

Endpoints

OAuth 2.0 authorization endpoint (v2)
https://login.microsoftonline.com/ce976899-299d-4a01-91e5-a5fee8f98626/oauth2/v2.0/authorize

OAuth 2.0 token endpoint (v2)
https://login.microsoftonline.com/ce976899-299d-4a01-91e5-a5fee8f98626/oauth2/v2.0/token

OAuth 2.0 authorization endpoint (v1)
https://login.microsoftonline.com/ce976899-299d-4a01-91e5-a5fee8f98626/oauth2/authorize

OAuth 2.0 token endpoint (v1)
https://login.microsoftonline.com/ce976899-299d-4a01-91e5-a5fee8f98626/oauth2/token

OpenID Connect metadata document
https://login.microsoftonline.com/ce976899-299d-4a01-91e5-a5fee8f98626/v2.0/.well-known/openid-configuration

Microsoft Graph API endpoint
https://graph.microsoft.com

Federation metadata document
https://login.microsoftonline.com/ce976899-299d-4a01-91e5-a5fee8f98626/federationmetadata/2007-06/federationmetadata.xml

WS-Federation sign-on endpoint
https://login.microsoftonline.com/ce976899-299d-4a01-91e5-a5fee8f98626/wsfed

SAML-P sign-on endpoint
https://login.microsoftonline.com/ce976899-299d-4a01-91e5-a5fee8f98626/saml2

SAML-P sign-out endpoint
https://login.microsoftonline.com/ce976899-299d-4a01-91e5-a5fee8f98626/saml2

Figure 21. Azure OIDC client config

```
dbms.security.oidc.azure.well_known_discovery_uri=https://login.microsoftonline.com/ce976899-299d-
4a01-91e5-a5fee8f98626/v2.0/.well-known/openid-configuration
```

3. Configure Neo4j to use Azure authentication by configuring the following settings in the *neo4j.conf* file:

```
dbms.security.authentication_providers=oidc-azure
dbms.security.authorization_providers=oidc-azure
dbms.security.oidc.azure.display_name=Azure
dbms.security.oidc.azure.auth_flow=pkce
dbms.security.oidc.azure.config=token_type_principal=id_token;token_type_authentication=id_token
```

4. Configure which JWT claim should be used for usernames. Possible values are `sub`, `email`, or `preferred_username`.

> `sub` is the only claim guaranteed to be unique and stable. Other claims, such as `email` or `preferred_username`, may change over time and should **not** be used for authentication. Neo4j may assign permissions to a user based on this username value in a hybrid authorization configuration. Thus, changing the username claim from `sub` is not recommended. For details, see Microsoft documentation as well as the OpenId spec.

```
dbms.security.oidc.azure.claims.username=sub
```

Map Azure groups to Neo4j roles

Decide whether you want to use Azure AD Groups directly or Azure App Roles.

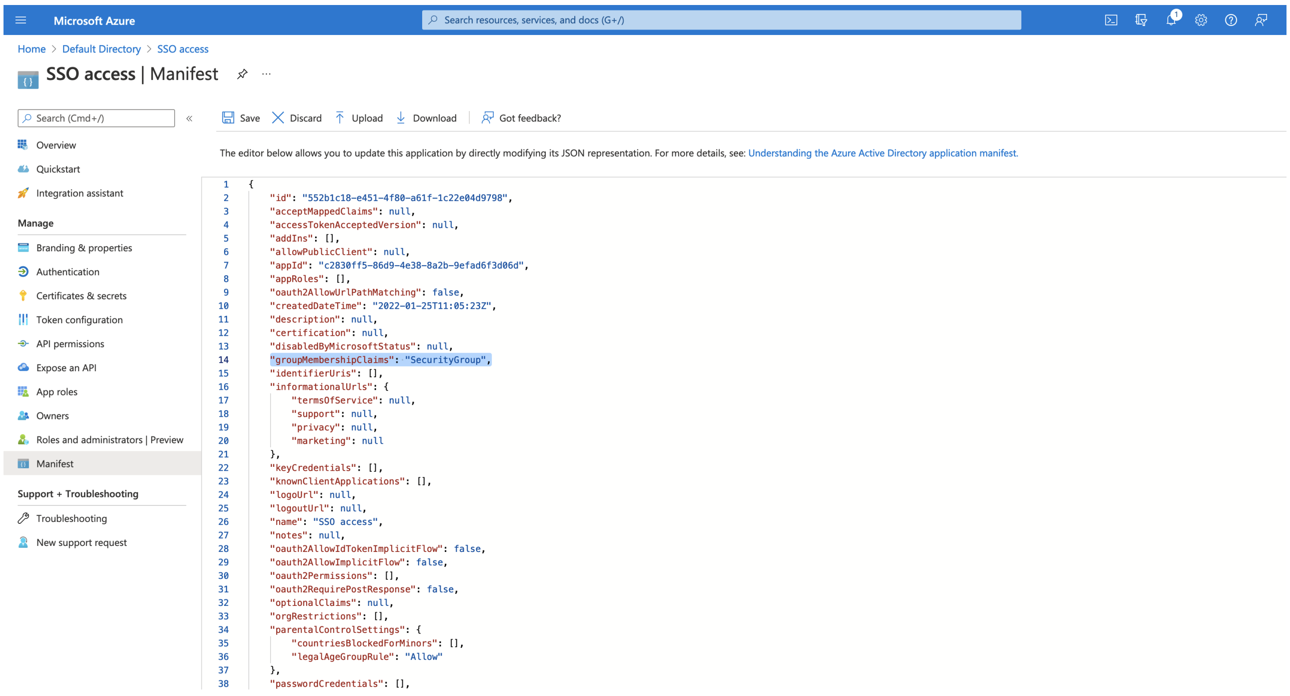
Using AD Groups directly might be convenient if you already have users assigned to relevant AD Groups and want to perform Group-to-Role mapping in Neo4j settings.

Azure App Roles allow a layer of separation between Neo4j roles and AD Groups. When App Roles are used, only the roles relevant to Neo4j are sent in the JWT token. This prevents leaking permissions between applications. JWT tokens also have a limitation of 200 roles per token per user, which can be avoided by sending only the relevant App Roles.

Details about Azure App Roles can be found in the Microsoft documentation.

Using Azure AD Groups directly

1. Configure the server to return the AD Group Object IDs in the JWT identity tokens. To do this, set `groupMembershipClaims` to `SecurityGroup` in the Manifest of the registered application:

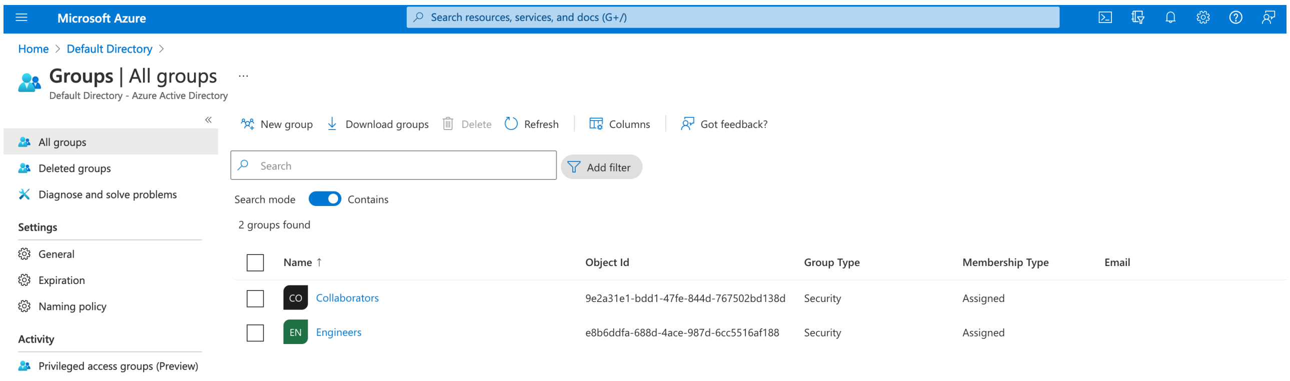


The editor below allows you to update this application by directly modifying its JSON representation. For more details, see: Understanding the Azure Active Directory application manifest.

```
{
    "id": "552b1c18-e451-4f80-a61f-1c22e04d9798",
    "acceptMappedClaims": null,
    "accessTokenAcceptedVersion": null,
    "addIns": [],
    "allowPublicClient": null,
    "appId": "c2830ff5-86d9-4e38-8a2b-9efad6f3d06d",
    "appRoles": [],
    "oauth2AllowUrlPathMatching": false,
    "createdDateTime": "2022-01-25T11:05:23Z",
    "description": null,
    "certification": null,
    "disabledByMicrosoftStatus": null,
    "groupMembershipClaims": "SecurityGroup",
    "identifierUris": [],
    "informationalUrls": {
        "termsOfService": null,
        "support": null,
        "privacy": null,
        "marketing": null
    },
    "keyCredentials": [],
    "knownClientApplications": [],
    "logoUrl": null,
    "logoutUrl": null,
    "name": "SSO access",
    "notes": null,
    "oauth2AllowIdTokenImplicitFlow": false,
    "oauth2AllowImplicitFlow": false,
    "oauth2Permissions": [],
    "oauth2RequirePostResponse": false,
    "optionalClaims": null,
    "orgRestrictions": [],
    "parentalControlSettings": {
        "countriesBlockedForMinors": [],
        "legalAgeGroupRule": "Allow"
    },
    "passwordCredentials": [],
```

Figure 22. Azure OIDC server claims

2. Create groups in the Azure AD console and assign users to them. Take note of the Object Id column. In the next step, you must map these to user roles in the Neo4j settings.



2 groups found

Name ↑	Object Id	Group Type	Membership Type	Email
CO Collaborators	9e2a31e1-bdd1-47fe-844d-767502bd138d	Security	Assigned	
EN Engineers	e8b6ddfa-688d-4ace-987d-6cc5516af188	Security	Assigned	

Figure 23. Azure OIDC server groups

3. Configure a mapping from Azure Ad Group Object IDs to Neo4j roles. For details, see Map the Identity Provider Groups to the Neo4j Roles.

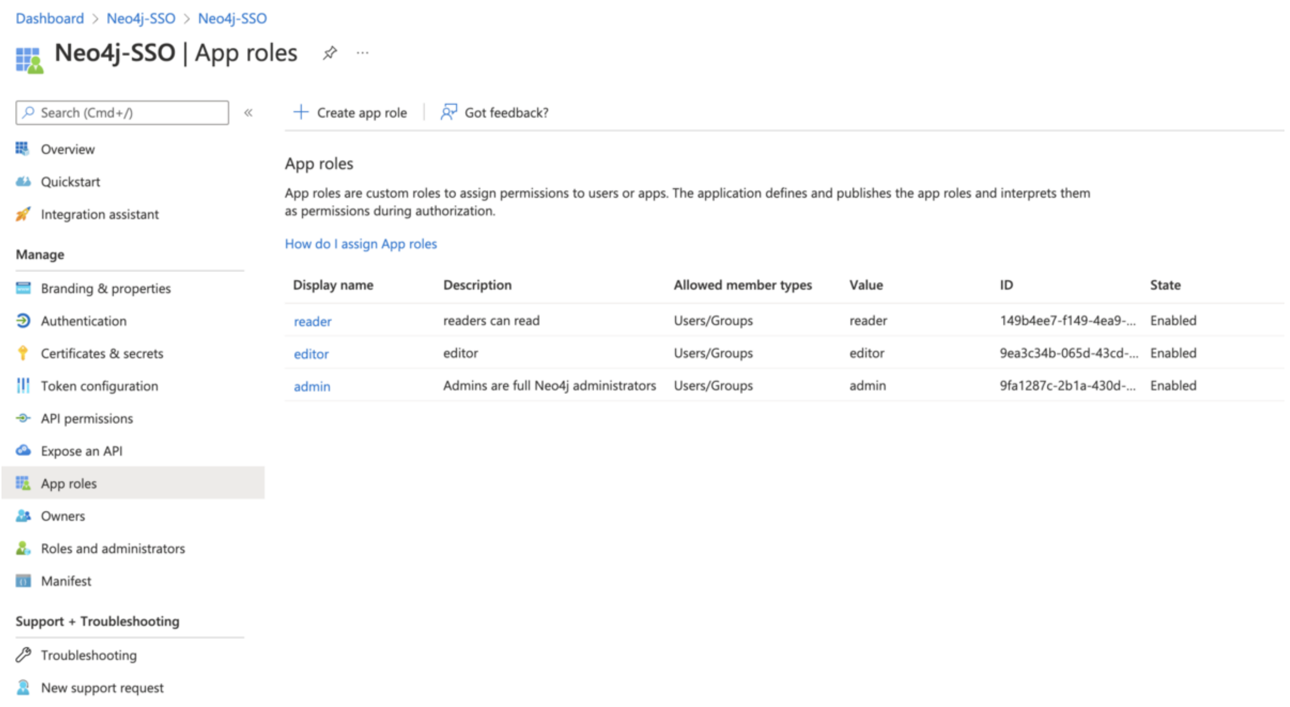
```
dbms.security.oidc.azure.authorization.group_to_role_mapping= "e8b6ddfa-688d-4ace-987d-6cc5516af188" =
admin; \
                                                              "9e2a31e1-bdd1-47fe-844d-767502bd138d" =
reader
```

4. Configure Neo4j to use the `groups` field from the JWT token.

```
dbms.security.oidc.azure.claims.groups=groups
```

Using Azure App Roles

1. On the app’s home page, navigate to **App roles** and add the Neo4j roles to the Azure active directory.


Dashboard > Neo4j-SSO > Neo4j-SSO

Neo4j-SSO | App roles

App roles

App roles are custom roles to assign permissions to users or apps. The application defines and publishes the app roles and interprets them as permissions during authorization.

How do I assign App roles

Display name	Description	Allowed member types	Value	ID	State
reader	readers can read	Users/Groups	reader	149b4ee7-f149-4ea9-...	Enabled
editor	editor	Users/Groups	editor	9ea3c34b-065d-43cd-...	Enabled
admin	Admins are full Neo4j administrators	Users/Groups	admin	9fa1287c-2b1a-430d-...	Enabled

Figure 24. Azure OIDC app roles config

2. The **Value** column in the App roles config must either correspond to Neo4j Roles or be mapped in the *neo4j.conf* file. For details, see Map the Identity Provider Groups to the Neo4j Roles.

```
dbms.security.oidc.azure.authorization.group_to_role_mapping= "managers" = admin; \
                                                              "engineers" = reader
```

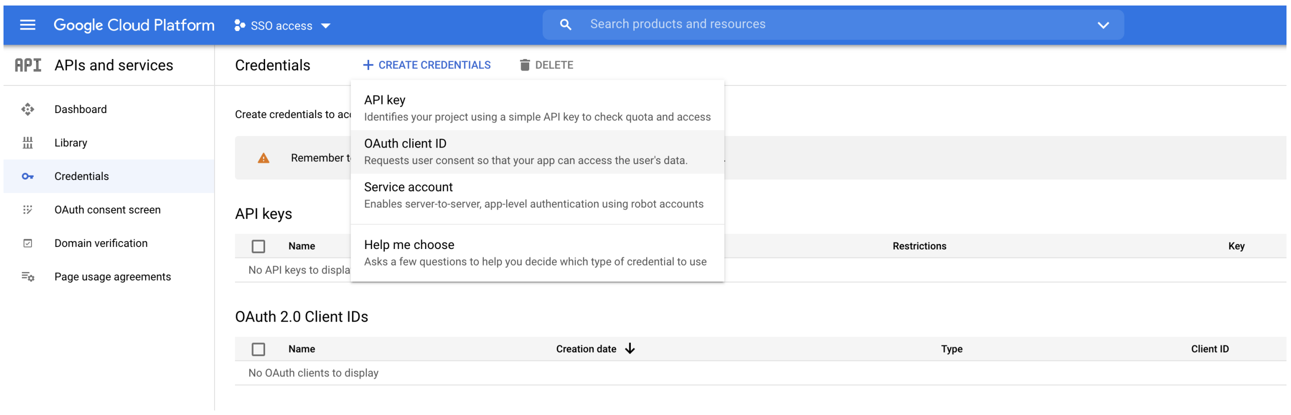
3. Configure Neo4j to use the `roles` field from the JWT token.

```
dbms.security.oidc.azure.claims.groups=roles
```

Google

This example shows how to use Google OpenID Connect for authentication using ID tokens in conjunction with native authorization.

1. Configure the client and the redirect URI:



Create credentials to ac

Remember t

API key
Identifies your project using a simple API key to check quota and access

OAuth client ID
Requests user consent so that your app can access the user's data.

Service account
Enables server-to-server, app-level authentication using robot accounts

Help me choose
Asks a few questions to help you decide which type of credential to use

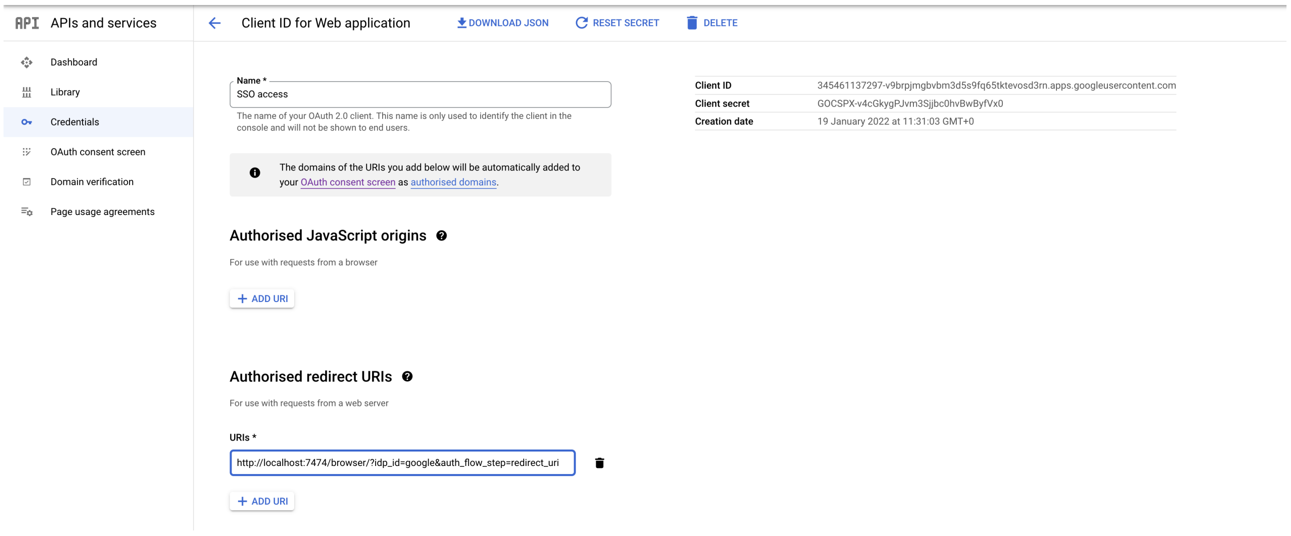
API keys

Name	Restrictions	Key
No API keys to displa		

OAuth 2.0 Client IDs

Name	Creation date	Type	Client ID
No OAuth clients to display			

Figure 25. Google OIDC client creation



Name *
SSO access

The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.

The domains of the URIs you add below will be automatically added to your OAuth consent screen as authorised domains.

Authorised JavaScript origins

For use with requests from a browser

Authorised redirect URIs

For use with requests from a web server

URIs *
http://localhost:7474/browser/?idp_id=google&auth_flow_step=redirect_uri

Client ID	345461137297-v9brpjmgbvbm3d5s9fq65tktevosd3rn.apps.googleusercontent.com
Client secret	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
Creation date	19 January 2022 at 11:31:03 GMT+0

Figure 26. Google OIDC client configuration

SSO authorization does not work with Google, as the JWT returned by Google does not contain information about the groups that a user belongs to, and cannot be configured to. Therefore, it is recommended to use native (or another flavor) authorization by creating a native version of the user in Neo4j.

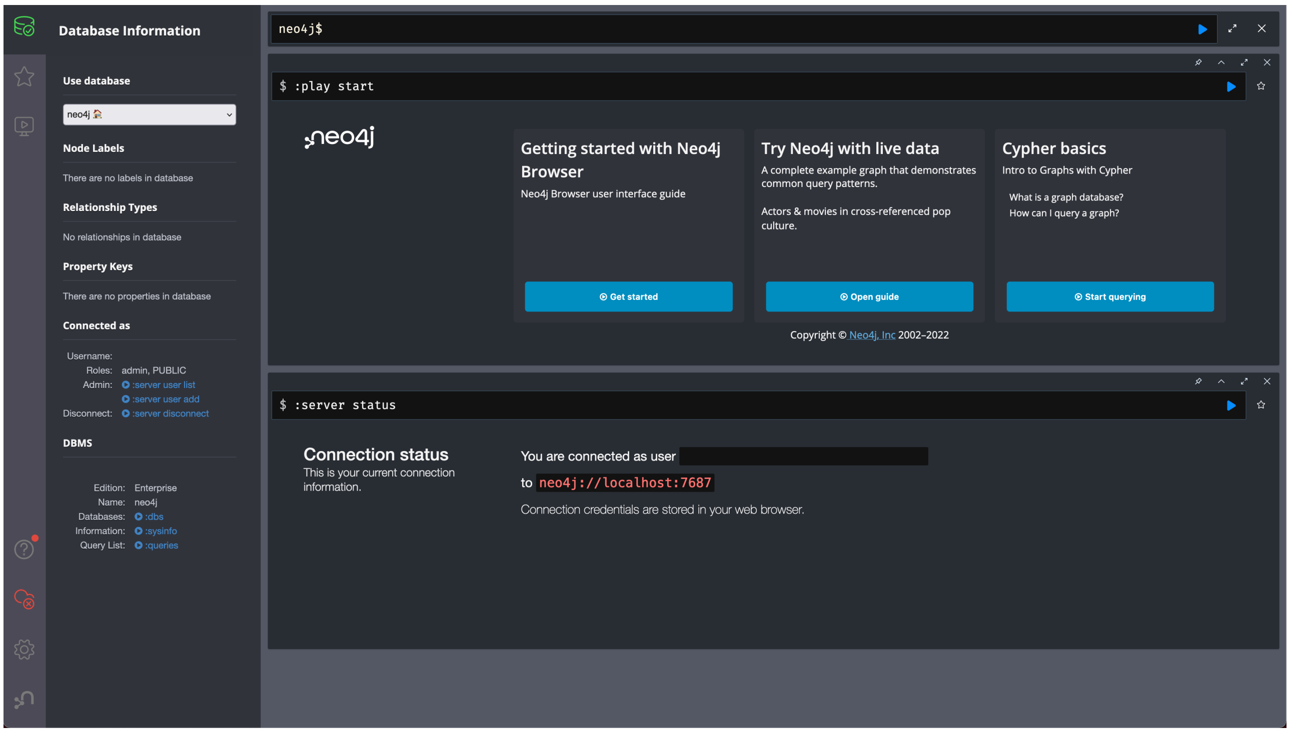
2. The role assigned to the email used to log in with SSO, in this case, `alice@neo4j-test.com`, must have `GRANT ROLE` permissions in the database (`native` authentication temporarily enabled):

```
CREATE USER `alice@neo4j-test.com` SET PASSWORD 'pass';
GRANT ROLE admin to `alice@neo4j-test.com`;
```

3. Configure Neo4j to use Google authentication by configuring the following settings in the *neo4j.conf* file:

```
dbms.security.authentication_providers=oidc-google
dbms.security.authorization_providers=native
dbms.security.oidc.google.display_name=Google
dbms.security.oidc.google.auth_flow=pkce
dbms.security.oidc.google.well_known_discovery_uri=https://accounts.google.com/.well-known/openid-
configuration
dbms.security.oidc.google.audience=345461137297-
v9brpjmgbvbm3d5s9fq65tktevosd3rn.apps.googleusercontent.com
dbms.security.oidc.google.claims.username=email
dbms.security.oidc.google.params=client_id=345461137297-
v9brpjmgbvbm3d5s9fq65tktevosd3rn.apps.googleusercontent.com;response_type=code;scope=openid profile
email
dbms.security.oidc.google.token_params=client_secret=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
dbms.security.oidc.google.config=token_type_principal=id_token;token_type_authentication=id_token
```

4. Log in with your Google SSO credentials using the email address and get the `admin` role when doing so:



Database Information

Use database

neo4j 🏠

Node Labels

There are no labels in database

Relationship Types

No relationships in database

Property Keys

There are no properties in database

Connected as

Username:
Roles: admin, PUBLIC
Admin: :server user list
:server user add
Disconnect: :server disconnect

DBMS

Edition: Enterprise
Name: neo4j
Databases: :dbs
Information: :sysinfo
Query List: :queries

```
neo4j$
```

```
$ :play start
```

Getting started with Neo4j Browser
Neo4j Browser user interface guide

Try Neo4j with live data
A complete example graph that demonstrates common query patterns.
Actors & movies in cross-referenced pop culture.

Cypher basics
Intro to Graphs with Cypher
What is a graph database?
How can I query a graph?

Copyright © Neo4j, Inc 2002–2022

```
$ :server status
```

Connection status
This is your current connection information.

You are connected as user [redacted]
to `neo4j://localhost:7687`
Connection credentials are stored in your web browser.

Figure 27. Azure OIDC successful login

The native authentication is disabled to prevent someone from logging in to *alice@neo4j-test.com* with the set password.

FAQ

When should `pkce` be used as auth flow?

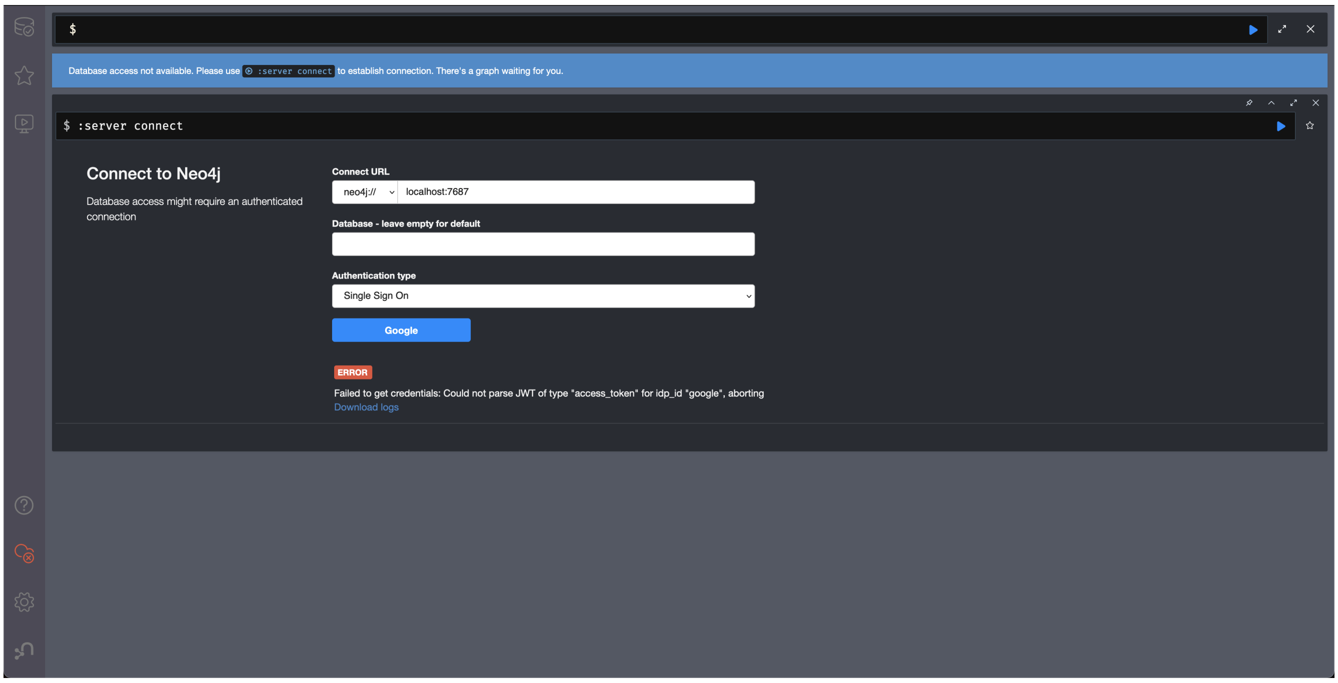
Assuming the client (Neo4j Browser or Bloom) can be accessed through the public internet, always use `pkce` auth-flow rather than `implicit` because the latter requires the client's secret to be available to the public client. In general, if both flows are available, it is recommended to opt for `pkce` because it is more secure than `implicit`.

Is Google authentication secure if it has a client secret listed in the config?

Yes. Google uses the pkce flow, but identity providers sometimes also use a client secret to ensure the client asking for a token is the one using it (pkce does not guarantee that). The client secret does not add any additional security as it is public but the `pkce` flow provides sufficient security.

Could not parse JWT of type "access_token"

When getting the message `Failed to get credentials: Could not parse JWT of type "access_token"` on Browser, it probably means the provider only accepts ID tokens.



```
$
```

Database access not available. Please use `:server connect` to establish connection. There's a graph waiting for you.

```
$ :server connect
```

Connect to Neo4j

Database access might require an authenticated connection

Connect URL: neo4j:// localhost:7687

Database - leave empty for default

Authentication type: Single Sign On

ERROR

Failed to get credentials: Could not parse JWT of type "access_token" for idp_id "google", aborting

Download logs

Figure 28. Failed to parse JWT of type access_token

Change to ID tokens in your *neo4j.conf*:

```
dbms.security.oidc.{{provider}}.config=token_type_principal=id_token;token_type_authentication=id_token
```

When should identity tokens vs. access tokens be used?

It is generally safer to use access tokens when possible due to being shorter-lived. If authorization permissions change on the identity provider, Neo4j will fail authorization. Neo4j Browser will try to reconnect and reflect the changed permissions faster than if ID tokens were used.

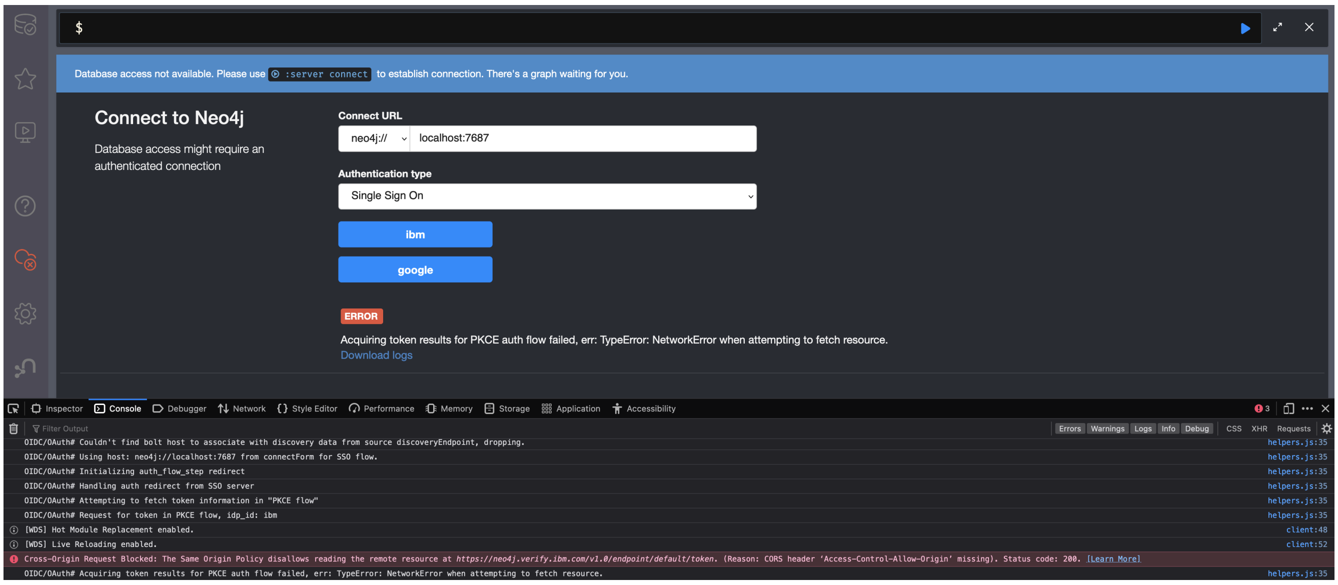
Debug logging of JWT claims

While setting up an OIDC integration, it is sometimes necessary to perform troubleshooting. In these cases, it can be useful to view the claims contained in the JWT supplied by the identity provider. To enable the logging of these claims at `DEBUG` level in the security log, set dbms.security.logs.oidc.jwt_claims_at_debug_level_enabled to `true` and the security log level to `DEBUG`.

Make sure to set dbms.security.logs.oidc.jwt_claims_at_debug_level_enabled back to `false` for production environments to avoid unwanted logging of potentially sensitive information. Also, bear in mind that the set of claims provided by an identity provider in the JWT can change over time.

How to debug further problems with the configuration

Apart from the logs available in *logs/debug.log* and *logs/security.log* in the Neo4j path, you can also use the web-development console in your web browser when doing the SSO authentication flow with Bloom or Neo4j Browser. This could reveal potential problems, such as the one presented below with an example identity provider and the Cross-Origin Request policy:

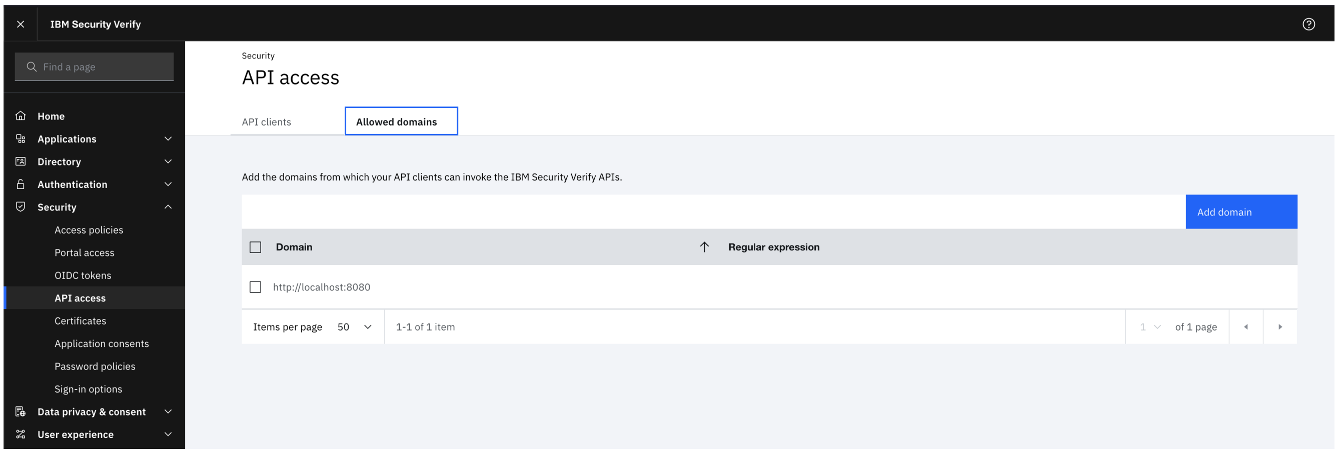


Acquiring token results for PKCE auth flow failed, err: TypeError: NetworkError when attempting to fetch resource.

```
OIDC/OAuth# Couldn't find bolt host to associate with discovery data from source discoveryEndpoint, dropping.      helpers.js:35
OIDC/OAuth# Using host: neo4j://localhost:7687 from connectForm for SSO flow.                                       helpers.js:35
OIDC/OAuth# Initializing auth_flow_step redirect                                                                   helpers.js:35
OIDC/OAuth# Handling auth redirect from SSO server                                                                 helpers.js:35
OIDC/OAuth# Attempting to fetch token information in "PKCE flow"                                                   helpers.js:35
OIDC/OAuth# Request for token in PKCE flow, idp_id: ibm                                                            helpers.js:35
[WDS] Hot Module Replacement enabled.                                                                                  client:48
[WDS] Live Reloading enabled.                                                                                          client:52
Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at https://neo4j.verify.ibm.com/v1.0/endpoint/default/token. (Reason: CORS header ‘Access-Control-Allow-Origin’ missing). Status code: 200. [Learn More]
OIDC/OAuth# Acquiring token results for PKCE auth flow failed, err: TypeError: NetworkError when attempting to fetch resource.   helpers.js:35
```

Figure 29. CORS error

The solution involves adding the redirect domain to the list of allowed domains in the provider (in this case, `localhost:8080`):



Add the domains from which your API clients can invoke the IBM Security Verify APIs.

Domain	Regular expression
http://localhost:8080	

Items per page 50 | 1-1 of 1 item | 1 of 1 page

Figure 30. CORS error solution allowing the redirect domain on the provider

Administering immutable privileges

This tutorial describes methods for administering immutable privileges, which are useful assets for restricting the actions of users who themselves are able to administer privileges. They offer a way to prevent such users from simply removing any restrictions by using their privilege management privileges.

In other words, having privilege management privileges is not sufficient to add or remove immutable privileges. The only way immutable privileges can be added or removed is when auth is disabled.

There are two specific scenarios which equate to auth being disabled. One is when `dbms.security.auth_enabled` is set to `false`, and the other is from the context of the Security Init file. Both of these are described in this tutorial.

Administering immutable privileges via the Security Init file

The Security Init file is a file containing a list of Cypher commands which are run when the system database is created for the first time. Putting immutable privilege commands in this file will ensure that they are in place before the DBMS is accessed for the first time.

> The following steps must be performed before the first time the Neo4j DBMS is started. If the DBMS has already been initialized, you will need to make changes by disabling auth. The steps to do this are described in sequence.

1. Create a file named `systemdb-init.cypher` and edit it to contain the immutable privileges which you require.
   In this instance, you would like to prevent **all** users (via the `PUBLIC` role) from being able to perform Database Management actions.
   Make sure that even users with the rights to alter their own privileges cannot remove this `DENY`.

   ```
   DENY IMMUTABLE DATABASE MANAGEMENT ON DBMS TO PUBLIC
   ```

2. Specify the path of the Security Init file by setting the `internal.dbms.init_file` config settings to the full path of the file created in step 1.
3. Start the DBMS.
4. Observe that the immutable privileges are now in place:

```
SHOW PRIVILEGES WHERE IMMUTABLE
```

Table 512. Result

access	action	resource	graph	segment	role	immutable
"DENIED"	"database_management"	"database"	"*"	"database"	"PUBLIC"	true

This privilege can now be considered to be an immutable part of the DBMS. The only way to subsequently remove it would be to disable auth.

Administering immutable privileges by disabling auth Enterprise edition

🔥	This should only be performed when you have other means of preventing access to the Neo4j DBMS.

When auth is disabled, immutable privileges can be added and removed in the same way as regular privileges. To do so, follow these steps:

1. Change the config setting `dbms.security.auth_enabled` to `false`.
2. Restart the Neo4j DBMS.
3. Create or remove immutable privileges in the same way as regular privileges, using the keyword `IMMUTABLE`.
4. Change the config setting `dbms.security.auth_enabled` to `true`.
5. Restart the Neo4j DBMS.
6. Observe that the following immutable privileges are now in place:

```
SHOW PRIVILEGES WHERE IMMUTABLE
```

Table 513. Result

access	action	resource	graph	segment	role	immutable
"DENIED"	"database_management"	"database"	"*"	"database"	"PUBLIC"	true

Privileges like this one can now be considered to be an immutable part of the DBMS. The only way to subsequently remove it would be to disable auth.

License