The Neo4j Operations Manual v3.0

Table of Contents

This is the operations manual for Neo4j version 3.0, authored by the Neo4j Team.

The main parts of the manual are:

- Introduction — Introducing Neo4j Community and Enterprise Editions.
- Deployment — Instructions on how to deploy Neo4j into production environments.
- Security — Instructions on setting up Neo4j security.
- Backup — Instructions on setting up Neo4j backups.
- Monitoring — Instructions on setting up Neo4j monitoring.
- Performance — Instructions on how to go about performance tuning for Neo4j.
- Reference — Listings of all Neo4j configuration parameters.
- Tutorial — Step-by-step instructions on various scenarios for setting up Neo4j.

Who should read this?

This manual is written for:

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

Chapter 1. Introduction

This chapter introduces Neo4j capabilities, editions, and architecture.

1.1. Neo4j editions

There are two editions of Neo4j to choose from: *Community Edition* and *Enterprise Edition*. The nature of the required solution will help decide which edition to select.

Community Edition is a fully functional edition of Neo4j, suitable for single instance deployments. It has full support for key Neo4j features, such as ACID compliance, Cypher, and programming APIs. It is ideal for smaller workgroup or do-it-yourself projects similar to:

- learning Neo4j and just getting started
- building a solution for an internal team that can tolerate downtime for support
- building a solution available to external users, but without guarantees on uptime or availability
- building a solution which does not have high demands for scalability or concurrent access

Enterprise Edition extends the functionality of Community Edition to include key features for performance and scalability, such as a clustering architecture for high availability and online backup functionality. It is the choice for production systems with availability requirements or needs for scaling up, for example:

- the ability to scale up your solution with the clustering architecture
- 24x7 availability capabilities
- ability to support disaster recovery
- provisioning for early stage load testing
- access to professional support from Neo Technology

Which is the right Neo4j edition for a particular deployment?

As a rule of thumb:

1. Both editions offer the same, great core graph database capabilities
2. Enterprise Edition is the choice for a commercial solution, a critical or highly depended-on internal solution, and when anticipate needing scalability, redundancy, or high availability.

Table 1. Features

Edition	Enterprise	Community
Property Graph Model	X	X
Native Graph Processing & Storage	X	X
ACID	X	X
Cypher - Graph Query Language	X	X
Language Drivers	X	X
Extensible REST API	X	X
High-Performance Native API	X	X
HTTPS	X	X

Table 2. Performance & Scalability

Edition	Enterprise	Community
Enterprise Lock Manager	X	-
Clustering	X	-
Hot Backups	X	-
Advanced Monitoring	X	-

1.2. Designing for the enterprise

When designing your solution, some of your first considerations will concern your functional requirements and the type of technology choices you make to meet them. Some of those functional requirements likely will include a need to scale to many concurrent users, maintain consistent uptime, or the ability to recover from a system failure and maintain availability. These are important production related questions that help drive your technical decisions and can ultimately guide you to choose to cluster Neo4j.

This covers four major advantages of using Neo4j clustering:

1. Read Scalability
2. High Availability
3. Disaster Recovery
4. Analytics

1.2.1. Read scalability

Clustering Neo4j allows you to distribute read workload across a number of Neo4j instances. You can take two approaches to scaling your reads with Neo4j:

Distribute load balance reads to any instance in the cluster

Neo4j's clustering architecture replicates the entire database to each instance in your cluster. Therefore you are able to direct any read from your application to any instance without much concern for data locality.

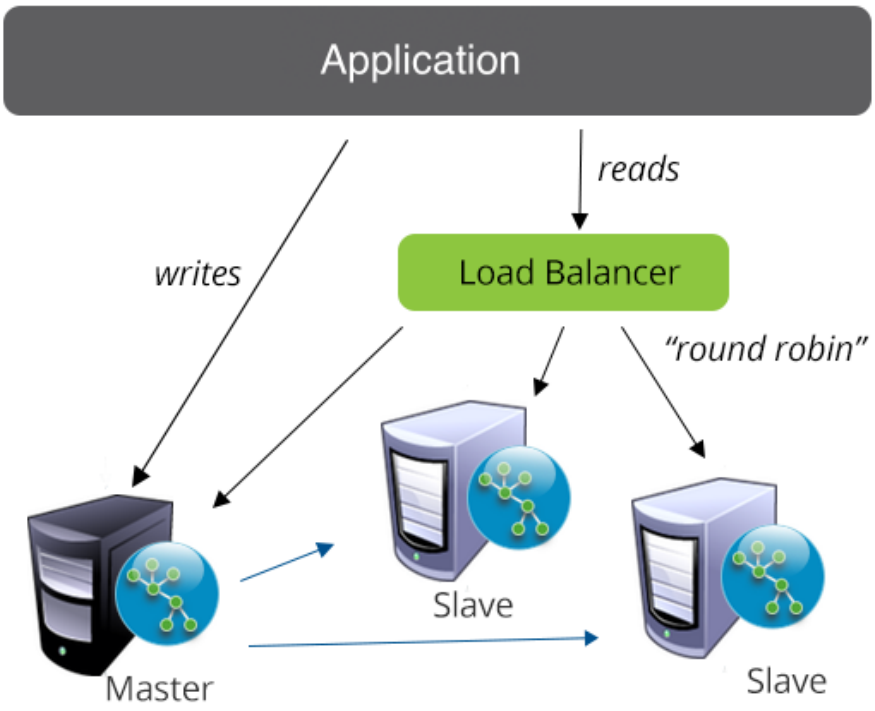

Figure 1. Distribute load balance reads to any instance in the cluster

When would you choose this method?

- You need to scale up the number of concurrent read requests
- Your data has no natural or obvious way of partitioning reads
- A significant portion of the data that needs to be read can reasonably be expected to already be in memory on any instance in the cluster.

Distribute direct reads to specific instances in the cluster

This is sometimes referred to as "cache sharding". The strategy simply allows you to take advantage of natural partitions in your data to direct reads to particular instances where the system will already have those datasets in memory. This approach is significantly beneficial when your total active dataset is much larger than can fit in memory in any particular instance.

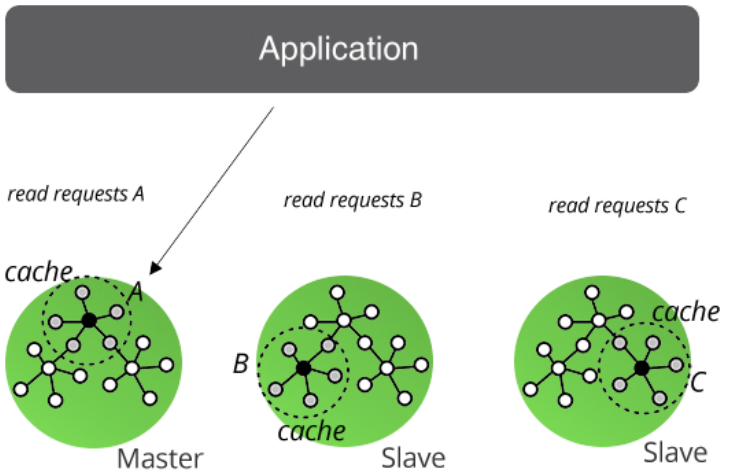

Figure 2. Cache sharding

When would you choose this method?

- Your total active data set is larger than can reasonably be expected to fit in memory in any single instance in your cluster.
- A natural or obvious partition can be identified in your dataset
- You have the application and operations ability to direct which instances are read from.

1.2.2. High availability

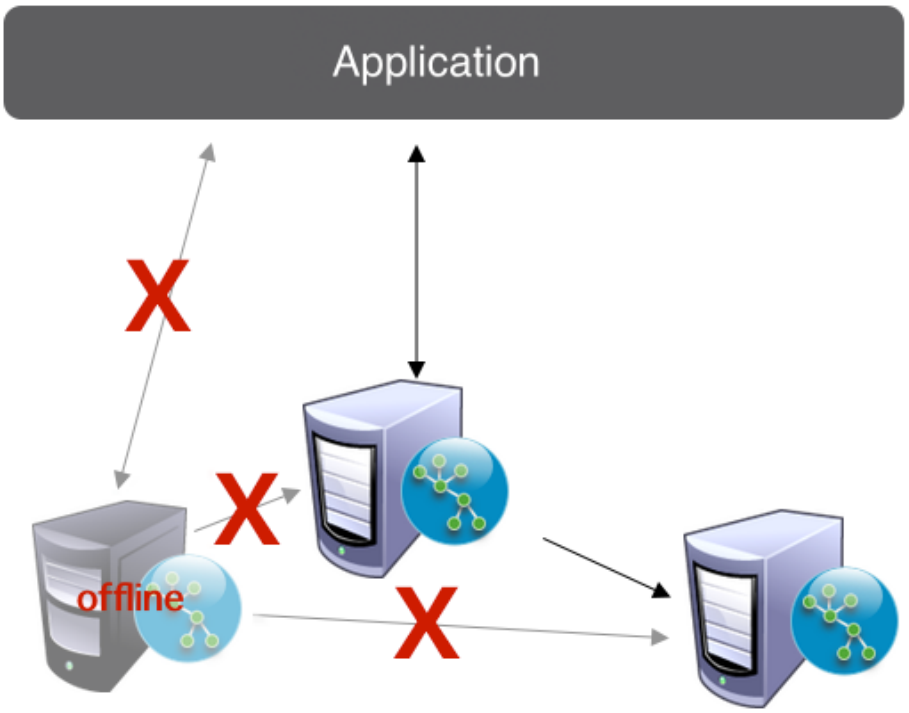

Figure 3. High availability cluster

A significant and fundamental functional requirement for any service or application is the requirements for overall availability. Very often this question is answered more by the demands of the users, the times they would be interacting with the solution, the impact downtime would have on the business or users of the system to complete their roles, or the financial impact of a system failure. These are not always customer-facing solutions and can be critical internal systems.

Availability can often be addressed with various strategies for recovery or mirroring. However, Neo4j's clustering architecture is an automated solution for ensuring Neo4j is consistently available to your application and end-users.

How do you know if you need Neo4j's clustering for high availability reasons?

- Neo4j is serving data for a critical business or consumer-facing solution that would impact the ability for the company to conduct business if the component were down.
- Global end-users with random access behavior are depending on the data stored in Neo4j.
- Business continuity must be ensured by availability of disaster recovery features.

1.2.3. Disaster recovery

Disaster recovery, in general terms, defines your ability to recover from major outages of your services. The most common example is whole-datacenter outages where many services are disrupted. In these cases a disaster recovery strategy can define a failover datacenter along with a strategy for bringing services back online.

Neo4j clustering can accommodate disaster recovery strategies that require very short-windows of downtime or low tolerances for data loss in disaster scenarios. By deploying a cluster instance to an alternate location, you have an active copy of your database up and available in your designated disaster recovery location that is consistently keeping up with the transactions against your database.

Why would you choose Clustering in support of Disaster Recovery?

1. Minimize downtime: Your application availability demands are very high and you cannot sustain significant periods of downtime.
2. Require real-time: You already employ a disaster recovery strategy for other application or service components that are near real-time.
3. Minimize data loss: You have a significantly large database that changes frequently and have low tolerance for data loss in a disaster scenario.

1.2.4. Analytics

Your application needs to access data for its purposes. It reads data, writes data, and is generally keeping your application service or end-users happy. Then comes the analytics team that wants to collect and aggregate data for their reports. Next thing you know, you have a set of long-running compute queries running against your production databases and disrupting your service or end-users' happiness.

You can't avoid servicing the needs of the analytics requests, but you can box in the impact their queries have on your service. Neo4j clustering can be used to include separate instances entirely in support of query analytics, either from end users or from BI tools. Using clustering means the data is always up to date for analytics queries as well.

When would you decide to use clustering to support analytics needs?

- You have regular BI users that consistently need to run analytics against the most recent versions of the data
- Your analytics includes queries that aggregate over large or entire sets of data
- Your analytics processes include complex compute algorithms for predictive or modeling purposes

1.3. Architecture

A Neo4j cluster is comprised of a single master instance and zero or more slave instances. All instances in the cluster have full copies of the data in their local database files. The basic cluster configuration consists of three instances:

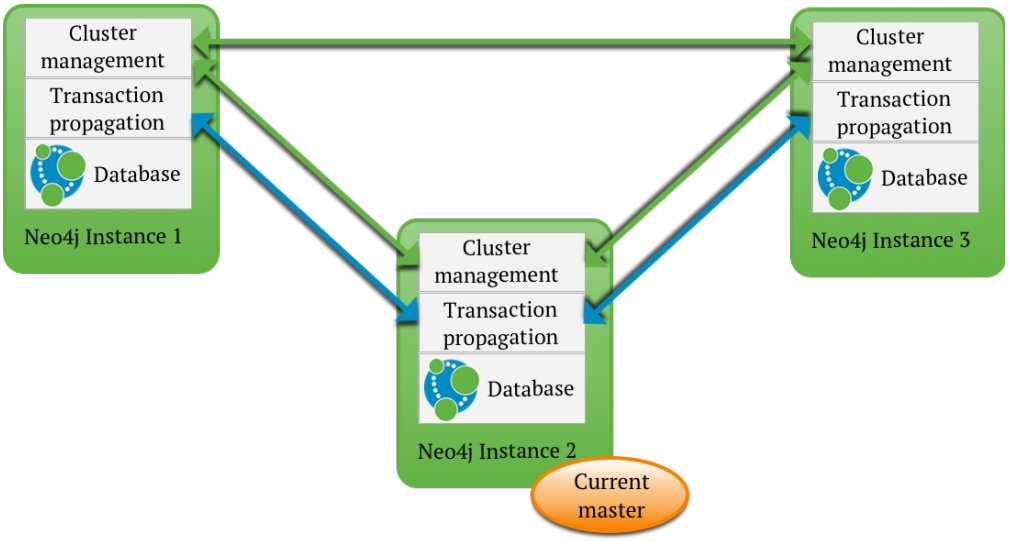

Figure 4. Neo4j cluster

Each instance contains the logic needed in order to coordinate with the other members of the cluster for data replication and election management, represented by the green arrows in the picture above. Each slave instance that is not an arbiter instance (see below) communicates with the master to keep databases up to date, as represented by the blue arrows in the picture above.

1.3.1. Arbiter instance

A special case of a slave instance is the arbiter instance. The arbiter instance comprises the full Neo4j software running in an arbiter mode, such that it participates in cluster communication, but it does not replicate a copy of the datastore.

1.3.2. Transaction propagation

Write transactions performed directly on the master will execute as though the instance were running in non-cluster mode. On success the transaction will be pushed out to a configurable number of slaves. This is done optimistically, meaning that if the push fails, the transaction will still be successful.

When performing a write transaction on a slave each write operation will be synchronized with the master. Locks will be acquired on both master and slave. When the transaction commits it will first be committed on the master and then, if successful, on the slave. To ensure consistency, a slave must be up to date with the master before performing a write operation. The automatic updating of slaves is built into the communication protocol between slave and master.

1.3.3. Failover

Whenever a Neo4j database becomes unavailable, for example caused by hardware failure or network outage, the other instances in the cluster will detect that and mark it as temporarily failed. A database instance that becomes available after an outage will automatically catch up with the cluster.

If the master goes down another member will be elected and have its role switched from slave to master after quorum (see below) has been reached within the cluster. When the new master has performed its role switch, it will broadcast its availability to all the other members of the cluster. Normally a new master is elected and started within seconds. During this time no writes can take

place

Quorum

A cluster must have quorum to elect a new master. Quorum is defined as: *more than 50% of active cluster members*. A simple rule of thumb when designing a cluster is: A cluster that must be able to tolerate **n** master instance failures requires **2n+1** instances to satisfy quorum and allow elections to take place. Therefore, the simplest valid cluster size is three instances, which allows for a single master failure.

Election Rules

1. If a master fails, or on a cold-start of the cluster, the slave with the highest committed transaction ID will be elected as the new master. This rule ensures that the slave with the most up-to-date datastore becomes the new master.
2. If a master fails and two or more slaves are tied, i.e. have the same highest committed transaction ID, the slave with the lowest ha.server_id value will be elected the new master. This is a good tie-breaker because the ha.server_id is unique within the cluster, and allows for configuring which instances can become master before others.

1.3.4. Branching

Data branching can be caused in two different ways:

- A slave falls too far behind the master and then leaves or re-joins the cluster. This type of branching is harmless.
- The master re-election happens and the old master has one or more committed transactions that the slaves did not receive before it died. This type of branching is harmful and requires action.

The database makes the best of the situation by creating a directory with the contents of the database files from before branching took place so that it can be reviewed and the situation be resolved. Data branching does not occur under normal operations.

1.3.5. Summary

All this can be summarized as:

- Write transactions can be performed on any database instance in a cluster.
- Neo4j cluster is fault tolerant and can continue to operate from any number of machines down to a single machine.
- Slaves will be automatically synchronized with the master on write operations.
- If the master fails, a new master will be elected automatically.
- The cluster automatically handles instances becoming unavailable (for example due to network issues), and also makes sure to accept them as members in the cluster when they are available again.
- Transactions are atomic, consistent and durable but eventually propagated out to other slaves.
- Updates to slaves are eventually consistent by nature but can be configured to be pushed optimistically from master during commit.
- If the master goes down, any running write transaction will be rolled back and new transactions will block or fail until a new master has become available.
- Reads are highly available and the ability to handle read load scales with more database instances in the cluster.

Chapter 2. Deployment

This chapter covers deploying Neo4j: capacity planning, single-instance and clustered installations, and post-installation tasks.

2.1. System requirements

This section provides an overview of the system requirements for running a Neo4j instance.

CPU

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

Minimum
: Intel Core i3

Recommended
: Intel Core i7

: IBM POWER8

Memory

More memory allows for larger graphs, but it needs to be configured properly to avoid disruptive garbage collection operations. See Memory tuning for suggestions.

Minimum
: 2GB

Recommended
: 16—32GB or more

Disk

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

Minimum
: 10GB SATA

Recommended
: SSD w/ SATA

Filesystem

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

Minimum
ext4 (or similar)

Recommended
ext4, ZFS

Software

Neo4j requires a Java Virtual Machine, *JVM*, to operate. Community Edition installers for Windows and Mac include a JVM for convenience. All other distributions, including all distributions of Neo4j Enterprise Edition, require a pre-installed JVM.

Java
OpenJDK 8 *(http://openjdk.java.net/)* or Oracle Java 8 *(http://www.oracle.com/technetwork/java/javase/downloads/index.html)*

IBM Java 8 *(http://www.ibm.com/developerworks/java/jdk/)*

Operating Systems
Linux (Ubuntu, Debian)

Windows Server 2012

Architectures
x86

OpenPOWER (POWER8)

2.2. File locations

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

Important files can be found in the following locations by default.

Package	Configuration	Data	Logs	Metrics	Import	Bin	Lib	Plugins
Linux or OS X tarball	*<neo4j-home>/conf/neo4j.conf*	*<neo4j-home>/data*	*<neo4j-home>/logs*	*<neo4j-home>/metrics*	*<neo4j-home>/import*	*<neo4j-home>/bin*	*<neo4j-home>/lib*	*<neo4j-home>/plugins*
Windows zip	*<neo4j-home>\conf\neo4j.conf*	*<neo4j-home>\data*	*<neo4j-home>\logs*	*<neo4j-home>\metrics*	*<neo4j-home>\import*	*<neo4j-home>\bin*	*<neo4j-home>\lib*	*<neo4j-home>\plugins*
Debian/Ubuntu .deb	*/etc/neo4j/neo4j.conf*	*/var/lib/neo4j/data*	*/var/log/neo4j*	*/var/lib/neo4j/metrics*	*/var/lib/neo4j/import*	*/usr/share/neo4j/bin*	*/usr/share/neo4j/lib*	*/var/lib/neo4j/plugins*
Windows desktop	*%APPDATA%\Neo4j Community Edition\neo4j.conf*	*%APPDATA%\Neo4j Community Edition*	*%APPDATA%\Neo4j Community Edition\logs*	*%APPDATA%\Neo4j Community Edition\metrics*	*%APPDATA%\Neo4j Community Edition\import*	*%ProgramFiles%\Neo4j CE 3.0\bin*	(in package)	*%ProgramFiles%\Neo4j CE 3.0\plugins*
OS X desktop	*${HOME}/Documents/Neo4j/neo4j.conf*	*${HOME}/Documents/Neo4j*	*${HOME}/Documents/Neo4j/logs*	*${HOME}/Documents/Neo4j/metrics*	*${HOME}/Documents/Neo4j/import*	(in package)	(in package)	(in package)

Please note that the data directory is internal to Neo4j and its structure subject to change between

versions without notice.

2.2.1. Log files

Filename	Description
neo4j.log	The standard log, where general information about Neo4j is written.
debug.log	Information useful when debugging problems with Neo4j.
http.log	Request log for the HTTP API.
gc.log	Garbage Collection logging provided by the JVM.
query.log	Log of executed queries that takes longer than a specified threshold. (Enterprise Edition only.)
service-error.log	Log of errors encountered when installing or running the Windows service. (Windows only.)

2.2.2. Configuration

Some of these paths are configurable with `dbms.directories.*` settings; see Configuration settings reference for details.

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

Location	Default	Environment variable	Notes
<neo4j-home>	parent of *bin*	`NEO4J_HOME`	Must be set explicitly if *bin* is not a subdirectory.
bin	directory where *neo4j* script is located	`NEO4J_BIN`	Must be set explicitly if *neo4j* script is invoked as a symlink.
conf	*<neo4j-home>/conf*	`NEO4J_CONF`	Must be set explicitly if it is not a subdirectory of *<neo4j-home>*.

2.2.3. Permissions

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

Read only

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

Read and write

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

Exectute

- all files in *bin*

2.3. Single instance

This section covers installing a single instance of Neo4j.

Neo4j runs on Linux, Windows and OS X. There are desktop installers for Community Edition available for OS X and Windows. There are also platform-specific packages and zip/tar archives of both Community and Enterprise editions.

2.3.1. Linux installation

Install Neo4j on Linux from a tarball and run as a console application or service.

Linux packages

For Debian packages, see Debian.

After installation you may have to do some platform specific configuration and performance tuning. For that, refer to Post-installation tasks.

Unix console application

1. Download the latest release from http://neo4j.com/download/.
   - Select the appropriate tar.gz distribution for your platform.
2. Extract the contents of the archive, using: `tar -xf <filename>`
   - Refer to the top-level extracted directory as: NEO4J_HOME
3. Change directory to: `$NEO4J_HOME`
   - Run: `./bin/neo4j console`
4. Stop the server by typing `Ctrl-C` in the console.

Linux service

The `neo4j` command can also be used with `start`, `stop`, `restart` or `status` instead of `console`. By using these actions, you can create a Neo4j service.

This approach to running Neo4j as a service is deprecated. We strongly advise you to run Neo4j from a package where feasible.

You can build your own `init.d` script. See for instance the Linux Standard Base specification on system initialization *(http://refspecs.linuxfoundation.org/LSB_3.1.0/LSB-Core-generic/LSB-Core-generic/tocsysinit.html)*, or one of the many samples *(https://gist.github.com/chrisvest/7673244)* and tutorials *(http://www.linux.com/learn/tutorials/442412-managing-linux-daemons-with-init-scripts)*.

2.3.2. OS X installation

Install Neo4j on OS X with a desktop installer or from a tarball. Run it as a desktop or console application, or as a service.

Mac OS X installer

1. Download the *.dmg* installer that you want from http://neo4j.com/download/.
2. Click the downloaded installer file.
3. Drag the Neo4j icon into the Applications folder.

If you install Neo4j using the Mac installer and already have an existing instance of Neo4j the installer will ensure that both the old and new versions can co-exist on your system.

Unix console application

1. Download the latest release from http://neo4j.com/download/.
   - Select the appropriate tar.gz distribution for your platform.
2. Extract the contents of the archive, using: `tar -xf <filename>`
   - Refer to the top-level extracted directory as: NEO4J_HOME
3. Change directory to: `$NEO4J_HOME`
   - Run: `./bin/neo4j console`
4. Stop the server by typing `Ctrl-C` in the console.

When Neo4j runs in console mode in the foreground logs are printed to the Terminal.

OS X service

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

2.3.3. Windows installation

Install Neo4j on Windows with a desktop installer or from a ZIP archive. Run it as a desktop or console application, or as a Windows service.

Windows installer

1. Download the version that you want from http://neo4j.com/download/.
   - Select the appropriate version and architecture for your platform.
2. Double-click the downloaded installer file.
3. Follow the prompts.

The installer will prompt to be granted Administrator privileges. Newer versions of Windows come with a SmartScreen feature that may prevent the installer from running — you can make it run anyway by clicking "More info" on the "Windows protected your PC" screen.

If you install Neo4j using the windows installer and you already have an existing instance of Neo4j the installer will select a new install directory by default. If you specify the same directory it will ask if you want to upgrade. This should proceed without issue although some users have reported a `JRE is damaged` error. If you see this error simply install Neo4j into a different location.

Windows console application

1. Download the latest release from http://neo4j.com/download/.
   - Select the appropriate Zip distribution.
2. Right-click the downloaded file, click Extract All.
3. Change directory to top-level extracted directory.
   - Run `bin\neo4j console`
4. 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 install-service` and start it with `bin\neo4j start`. Other commands available are `stop`, `restart`, `status` and `uninstall-service`.

Windows PowerShell module

The Neo4j PowerShell module allows administrators to:

- install, start and stop Neo4j Windows® Services
- and start tools, such as `Neo4j Shell` and `Neo4j Import`.

The PowerShell module is installed as part of the ZIP file *(http://neo4j.com/download/)* 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. Bottom-right on that dialog you will find an "Unblock" button. Click that. Then you should be able to import the module.

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

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

For more information see About execution policies *(https://technet.microsoft.com/en-us/library/hh847748.aspx)*.

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

How do I import the module?

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

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

This will add the module to the current session.

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

```
Invoke-Neo4j console
```

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

How do I get help about the module?

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

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

The output should be similar to the following:

```
CommandType     Name                                Version    Source
-----------     ----                                -------    ------
Function        Invoke-Neo4j                        3.0.0      Neo4j-Management
Function        Invoke-Neo4jAdmin                   3.0.0      Neo4j-Management
Function        Invoke-Neo4jBackup                  3.0.0      Neo4j-Management
Function        Invoke-Neo4jImport                  3.0.0      Neo4j-Management
Function        Invoke-Neo4jShell                   3.0.0      Neo4j-Management
```

The module also supports the standard PowerShell help commands.

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

To see examples for a command, do like this:

```
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`.

2.3.4. Debian

This article covers deploying Neo4j on Debian and Debian-based distributions like Ubuntu using the Neo4j Debian package.

Installation

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

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

Prerequisites (Ubuntu 14.04 and Debian 8 only)

Neo4j 3.0 requires the Java 8 runtime. Java 8 is not included in Ubuntu 14.04 LTS or Debian 8 (jessie) and will have to be installed manually prior to installing or upgrading to Neo4j 3.0. Debian users can find OpenJDK 8 in backports *(https://packages.debian.org/jessie-backports/openjdk-8-jdk)*.

Java 8 on Debian 8

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

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

You are now ready to install Neo4j 3.0 (which will install Java 8 automatically if it is not already installed). See Dealing with multiple installed Java versions to make sure you can start Neo4j after install.

Java 8 on Ubuntu 14.04

Users on Ubuntu 14.04 can add oracle Java 8 via webupd8. Note that when installing from webupd8 or any other PPA, you must install Java 8 manually before installing Neo4j. Otherwise there is a risk that Java 9 will be installed in, which is not compatible with Neo4j.

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

Once installed, see Dealing with multiple installed Java versions to make sure you can start Neo4j after

install.

Dealing with multiple installed Java versions

It is important that you configure your default Java version to point to Java 8, or Neo4j 3.0 will be unable to start. Do so with the `update-java-alternatives` command.

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

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

```
java-1.7.0-openjdk-amd64 1071 /usr/lib/jvm/java-1.7.0-openjdk-amd64
java-1.8.0-openjdk-amd64 1069 /usr/lib/jvm/java-1.8.0-openjdk-amd64
```

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

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

Add the repository

The Debian package is available from http://debian.neo4j.org. To use the repository follow these steps:

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

Installing

To install Neo4j Community Edition:

```
sudo apt-get install neo4j={neo4j-version-exact}
```

To install Neo4j Enterprise Edition:

```
sudo apt-get install neo4j-enterprise={neo4j-version-exact}
```

Upgrade

There are three steps involved in upgrading a Neo4j Debian/Ubuntu installation. First, the configuration files need to be migrated. Then the database must be imported. Finally, the database store format must be upgraded.

Configuration files

The configuration files have changed in 3.0. If you have not edited the configuration files, the Debian package will simply remove the files that are no longer necessary, and replace the old default files with new default files.

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

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

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

Importing the 2.3 database to 3.0

The location of databases has changed in 3.0. 2.3 databases will need to be imported to 3.0. To do this, the `neo4j-admin import` command can be used. To import a database called `graph.db` (the default database name in 2.3) use the following command:

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

This command will import the database located in `/var/lib/neo4j/data/graph.db` into 3.0 and call it `graph.db`.

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

Migrating the 2.3 database to 3.0

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

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

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

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

```
sudo service neo4j start
```

File locations

File locations for all Neo4j packages are documented *here*.

Operation

Most Neo4j configuration goes into *neo4j.conf*. Some package-specific options are set in */etc/default/neo4j*.

Environment variable	Default value	Details
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.

Environment variable	Default value	Details
NEO4J_ULIMIT_NOFILE	*60000*	Maximum number of file handles that can be opened by the Neo4j process. See this page for details.

2.3.5. Docker

This article covers running Neo4j in a Docker container.

Docker does not run natively on OS X or Windows. For running Docker on OS X *(https://docs.docker.com/engine/installation/mac/)* and Windows *(https://docs.docker.com/engine/installation/windows/)* please consult the Docker documentation.

Overview

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

- `7474` for HTTP.
- `7473` for HTTPS.
- `7687` for Bolt.

It also exposes two volumes:

- */data* to allow the database to be persisted outside its container.
- */logs* to allow access to Neo4j log files.

```
docker run \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j/data:/data \
    --volume=$HOME/neo4j/logs:/logs \
    neo4j:3.0
```

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

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

By default Neo4j requires authentication. You have to login with `neo4j/neo4j` at the first connection and set a new password.

Neo4j editions

Tags are available for both Neo4j Community and Enterprise editions. Version-specific Enterprise Edition tags have an `-enterprise` suffix, for example `neo4j:3.0.0-enterprise`. Community Edition tags have no suffix, for example `neo4j:3.0.0`. The latest Neo4j Enterprise Edition release is available as `neo4j:enterprise`.

Upgrade Neo4j on Docker

There are several changes in Neo4j 3.0 that affect the Docker image.

- Neo4j now exposes three ports: 7473, 7474 and 7687.

- The configuration has been overhauled; the environment variables that are available have been changed to reflect the new settings (see below for details).

If you are supplying your own configuration files rather than using environment variables then you will need to migrate those. See Single-instance upgrade for details.

Docker configuration

File descriptor limit

Neo4j may use a large number of file descriptors if many indexes are in use or there is a large number of simultaneous database connections.

Docker controls the number of open file descriptors in a container; the limit depends on the configuration of your system. We recommend a limit of at least 40000 for running Neo4j.

To check the limit on your system, run this command:

```
docker run neo4j:3.0 \
    bash -c 'echo Soft limit: $(ulimit -Sn); echo Hard limit: $(ulimit -Hn)'
```

To override the default configuration for a single container, use the `--ulimit` option like this:

```
docker run \
    --detach \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j/data:/data \
    --volume=$HOME/neo4j/logs:/logs \
    --ulimit=nofile=40000:40000
    neo4j:3.0
```

Neo4j configuration

The default configuration provided by this image is intended for learning about Neo4j, but must be modified to make it suitable for production use. In particular the memory assigned to Neo4j is very limited (see `NEO4J_CACHE_MEMORY` and `NEO4J_HEAP_MEMORY` below), to allow multiple containers to be run on the same server. You can read more about configuring Neo4j in the Configuration settings reference.

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:3.0
```

The following environment variables are available:

- `NEO4J_AUTH`: controls authentication, set to `none` to disable authentication or `neo4j/<password>` to override the default password (see Security for details).
- `NEO4J_dbms_memory_pagecache_size`: the size of Neo4j's native-memory cache, defaults to 512M
- `NEO4J_dbms_memory_heap_maxSize`: the size of Neo4j's heap in MB, defaults to 512
- `NEO4J_dbms_txLog_rotation_retentionPolicy`: the retention policy for logical logs, defaults to `100M size`
- `NEO4J_dbms_allowFormatMigration`: set to `true` to enable upgrades, defaults to `false` (see the Single-instance upgrade for details)

Neo4j Enterprise Edition

The following settings control features that are only available in the Enterprise Edition of Neo4j.

- `NEO4J_dbms_mode`: the database mode, defaults to `SINGLE`, set to `HA` to create a cluster
- `NEO4J_ha_serverId`: the id of the server, must be unique within a cluster
- `NEO4J_ha_host_coordination`: the address (including port) used for cluster coordination in HA mode, this must be resolvable by all cluster members
- `NEO4J_ha_host_data`: the address (including port) used for data transfer in HA mode, this must be resolvable by all cluster members
- `NEO4J_ha_initialHosts`: comma-separated list of other members of the cluster

See below for an example of how to configure HA clusters.

/conf volume

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

```
docker run \
    --detach \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j/data:/data \
    --volume=$HOME/neo4j/logs:/logs \
    --volume=$HOME/neo4j/conf:/conf \
    neo4j:3.0
```

Any configuration files in the */conf* volume will override files provided by the image. This includes values that may have been set in response to environment variables passed to the container by Docker. So if you want to change one value in a file you must ensure that the rest of the file is complete and correct.

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

```
docker run --rm\
    --volume=$HOME/neo4j/conf:/conf \
    neo4j:3.0 dump-config
```

Build a new image

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

```
FROM neo4j:3.0
```

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

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

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

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

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

Neo4j Highly Available mode

(This feature is only available in Neo4j Enterprise Edition.)

In order to run Neo4j in HA mode under Docker you need to wire up the containers in the cluster so that they can talk to each other. Each container must have a network route to each of the others and the `NEO4J_ha_host_coordination`, `NEO4J_ha_host_data` and `NEO4J_ha_initialHosts` environment variables must be set accordingly (see above).

Within a single Docker host, this can be achieved as follows.

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

docker run --name=instance1 --detach --publish=7474:7474 --publish=7687:7687 --net=cluster --hostname
=instance1 \
    --volume=$HOME/neo4j/logs1:/logs \
    --env=NEO4J_dbms_mode=HA --env=NEO4J_ha_serverId=1 \
    --env=NEO4J_ha_host_coordination=instance1:5001 --env=NEO4J_ha_host_data=instance1:6001 \
    --env=NEO4J_ha_initialHosts=instance1:5001,instance2:5001,instance3:5001 \
    neo4j:3.0-enterprise

docker run --name=instance2 --detach --publish 7475:7474 --publish=7688:7687 --net=cluster --hostname
=instance2 \
    --volume=$HOME/neo4j/logs2:/logs \
    --env=NEO4J_dbms_mode=HA --env=NEO4J_ha_serverId=2 \
    --env=NEO4J_ha_host_coordination=instance2:5001 --env=NEO4J_ha_host_data=instance2:6001 \
    --env=NEO4J_ha_initialHosts=instance1:5001,instance2:5001,instance3:5001 \
    neo4j:3.0-enterprise

docker run --name=instance3 --detach --publish 7476:7474 --publish=7689:7687 --net=cluster --hostname
=instance3 \
    --volume=$HOME/neo4j/logs3:/logs \
    --env=NEO4J_dbms_mode=HA --env=NEO4J_ha_serverId=3 \
    --env=NEO4J_ha_host_coordination=instance3:5001 --env=NEO4J_ha_host_data=instance3:6001 \
    --env=NEO4J_ha_initialHosts=instance1:5001,instance2:5001,instance3:5001 \
    neo4j:3.0-enterprise
```

See the Set up a Neo4j cluster for more details of Neo4j Highly Available mode.

User-defined procedures

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

```
docker run --publish 7474:7474 --publish=7687:7687 --volume=$HOME/neo4j/plugins:/plugins neo4j:3.0
```

See Developer Manual Procedures *(http://neo4j.com/docs/developer-manual/3.0/procedures/)* for more details on procedures.

Neo4j shell

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

```
docker exec --interactive --tty <container> bin/neo4j-shell
```

Encryption

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

```
docker run --publish 7473:7473 --publish=7687:7687 --volume $HOME/neo4j/ssl:/ssl neo4j:3.0
```

2.4. Highly Available cluster

This section covers installing a Neo4j Highly Available cluster in Neo4j Enterprise Edition.

2.4.1. Setup and configuration

Neo4j can be configured in cluster mode to accommodate differing requirements for load, fault tolerance and available hardware. Refer to design considerations for a discussion on different design options.

Follow these steps in order to configure a Neo4j cluster:

1. Download and install the Neo4j Enterprise Edition on each of the servers to be included in the cluster.
2. If applicable, decide which server(s) that are to be configured as arbiter instance(s).
3. Edit the Neo4j configuration file on each of the servers to accommodate the design decisions.
4. Follow installation instructions for a single instance installation.
5. Modify the configuration files on each server as outlined in the section below. There are many parameters that can be modified to achieve a certain behavior. However, the only ones mandatory for an initial cluster are: `dbms.mode`, `ha.server_id` and `ha.initial_hosts`.

Important configuration settings

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

Please note that IP addresses or hostnames should be explicitly configured for the machines participating in the cluster. In the absence of a specified IP address, Neo4j will attempt to find a valid interface for binding. This is not recommended practice.

dbms.mode

`dbms.mode` configures the operating mode of the database.

For cluster mode it is set to: `dbms.mode=HA`

ha.server_id

`ha.server_id` is the cluster identifier for each instance. It must be a positive integer and must be unique among all Neo4j instances in the cluster.

For example, `ha.server_id=1`.

ha.host.coordination

`ha.host.coordination` is an address/port setting that specifies where the Neo4j instance will listen for cluster communication. The default port is `5001`.

For example, `ha.host.coordination=192.168.33.22:5001` will listen for cluster communications on port 5001.

ha.initial_hosts

`ha.initial_hosts` is a comma separated list of address/port pairs, which specifies how to reach other Neo4j instances in the cluster (as configured via their `ha.host.coordination` option). These hostname/ports will be used when the Neo4j instances start, to allow them to find and join the cluster. When cold starting the cluster, i.e. when no cluster is available yet, the database will be unavailable until all members listed in ha.initial_hosts are online and communicating with each other. It is good practice to configure all the instances in the cluster to have the exact same entries in ha.initial_hosts, for the cluster to come up quickly and cleanly.

Do not use any whitespace in this configuration option.

For example, `ha.initial_hosts=192.168.33.21:5001,192.168.33.22:5001,192.168.33.23:5001` will initiate a cluster containing the hosts 192.168.33.21-33, all listening on the same port, 5001.

ha.host.data

`ha.host.data` is an address/port setting that specifies where the Neo4j instance will listen for transactions from the cluster master. The default port is `6001`.

`ha.host.data` must use a different port than `ha.host.coordination`.

For example, `ha.host.data=192.168.33.22:6001` will listen for transactions from the cluster master on port 6001.

Address and port formats

The `ha.host.coordination` and `ha.host.data` configuration options are specified as `<hostname or IP address>:<port>`.

For `ha.host.data` the address must be an address assigned to one of the host's network interfaces.

For `ha.host.coordination` the address must be an address assigned to one of the host's network interfaces, or the value `0.0.0.0`, which will cause Neo4j to listen on every network interface.

Either the address or the port can be omitted, in which case the default for that part will be used. If the hostname or IP address is omitted, then the port must be preceded with a colon (eg. `:5001`).

The syntax for setting a port range is: `<hostname or IP address>:<first port>[-<second port>]`. In this case, Neo4j will test each port in sequence, and select the first that is unused. Note that this usage is not permitted when the hostname is specified as `0.0.0.0` (the "all interfaces" address).

For a hands-on tutorial for setting up a Neo4j cluster, see Set up a Neo4j cluster.

Review Reference for a list of all available configuration settings.

2.4.2. Arbiter instances

A typical deployment of Neo4j will use a cluster of three machines to provide fault tolerance and read scalability.

While having at least three instances is necessary for failover to happen in case the master becomes unavailable, it is not required for all instances to run the full Neo4j stack. Instead, something called *arbiter instances* can be deployed. They are regarded as cluster participants in that their role is to take part in master elections with the single purpose of breaking ties in the election process. That makes possible a scenario where you have a cluster of two Neo4j database instances and an additional arbiter instance and still enjoy tolerance of a single failure of either of the three instances.

Arbiter instances are configured in *neo4j.conf* using the same settings as standard Neo4j cluster members. The instance is configured to be an arbiter by setting the dbms.mode option to `ARBITER`. Settings that are not cluster specific are of course ignored, so you can easily start up an arbiter instance in place of a properly configured Neo4j instance.

To start the arbiter instance, run `neo4j` as normal:

```
neo4j_home$ ./bin/neo4j start
```

You can stop, install and remove it as a service and ask for its status in exactly the same way as for other Neo4j instances.

2.4.3. Endpoints for status information

Introduction

A common use case for Neo4j HA clusters is to direct all write requests to the master while using slaves for read operations, distributing the read load across the cluster and and gain failover capabilities for your deployment. The most common way to achieve this is to place a load balancer in front of the HA cluster, an example being shown with HA Proxy. As you can see in that guide, it makes

use of a HTTP endpoint to discover which instance is the master and direct write load to it. In this section, we will deal with this HTTP endpoint and explain its semantics.

The endpoints

Each HA instance comes with 3 endpoints regarding its HA status. They are complimentary but each may be used depending on your load balancing needs and your production setup. Those are:

- /db/manage/server/ha/master
- /db/manage/server/ha/slave
- /db/manage/server/ha/available

The /master and /slave endpoints can be used to direct write and non-write traffic respectively to specific instances. This is the optimal way to take advantage of Neo4j's scaling characteristics. The /available endpoint exists for the general case of directing arbitrary request types to instances that are available for transaction processing.

To use the endpoints, perform an HTTP GET operation on either and the following will be returned:

Table 3. HA HTTP endpoint responses

Endpoint	Instance State	Returned Code	Body text
`/db/manage/server/ha/master`	Master	`200 OK`	`true`
	`Slave`	404 Not Found	`false`
	`Unknown`	404 Not Found	`UNKNOWN`
`/db/manage/server/ha/slave`	Master	`404 Not Found`	`false`
	`Slave`	200 OK	`true`
	`Unknown`	404 Not Found	`UNKNOWN`
`/db/manage/server/ha/available`	Master	`200 OK`	`master`
	`Slave`	200 OK	`slave`
	`Unknown`	404 Not Found	`UNKNOWN`

Examples

From the command line, a common way to ask those endpoints is to use curl. With no arguments, curl will do an HTTP GET on the URI provided and will output the body text, if any. If you also want to get the response code, just add the -v flag for verbose output. Here are some examples:

- Requesting master endpoint on a running master with verbose output

```
#> curl -v localhost:7474/db/manage/server/ha/master
* About to connect() to localhost port 7474 (#0)
*   Trying ::1...
* connected
* Connected to localhost (::1) port 7474 (#0)
> GET /db/manage/server/ha/master HTTP/1.1
> User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
> Host: localhost:7474
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/plain
< Access-Control-Allow-Origin: *
< Transfer-Encoding: chunked
< Server: Jetty(6.1.25)
<
* Connection #0 to host localhost left intact
true* Closing connection #0
```

- Requesting slave endpoint on a running master without verbose output:

```
#> curl localhost:7474/db/manage/server/ha/slave
false
```

- Finally, requesting the master endpoint on a slave with verbose output

```
#> curl -v localhost:7475/db/manage/server/ha/master
* About to connect() to localhost port 7475 (#0)
*   Trying ::1...
* connected
* Connected to localhost (::1) port 7475 (#0)
> GET /db/manage/server/ha/master HTTP/1.1
> User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
> Host: localhost:7475
> Accept: */*
>
< HTTP/1.1 404 Not Found
< Content-Type: text/plain
< Access-Control-Allow-Origin: *
< Transfer-Encoding: chunked
< Server: Jetty(6.1.25)
<
* Connection #0 to host localhost left intact
false* Closing connection #0
```

Unknown status

The `UNKNOWN` status exists to describe when a Neo4j instance is neither master nor slave. For example, the instance could be transitioning between states (master to slave in a recovery scenario or slave being promoted to master in the event of failure), or the instance could be an arbiter instance. If the `UNKNOWN` status is returned, the client should not treat the instance as a master or a slave and should instead pick another instance in the cluster to use, wait for the instance to transit from the `UNKNOWN` state, or undertake restorative action via systems admin.

If the Neo4j server has Basic Security enabled, the HA status endpoints will also require authentication credentials. For some load balancers and proxy servers, providing this with the request is not an option. For those situations, consider disabling authentication of the HA status endpoints by setting `dbms.security.ha_status_auth_enabled=false` in the `neo4j.conf` configuration file.

2.4.4. HAProxy for load balancing

Configure HAProxy for load balancing in a Neo4j Highly Available cluster.

In the Neo4j HA architecture, the cluster is typically fronted by a load balancer. In this section we will explore how to set up HAProxy to perform load balancing across the HA cluster.

For this tutorial we will assume a Linux environment with HAProxy already installed. See http://www.haproxy.org/ for downloads and installation instructions.

Configuring HAProxy for the Bolt Protocol

In a typical HA deployment, HAProxy will be configured with two open ports, one for routing write operations to the master and one for load balancing read operations over slaves. Each application will have two driver instances, one connected to the master port for performing writes and one connected to the slave port for performing reads.

First we set up the mode and timeouts. The settings below will kill the connection if a server or a client is idle for longer than two hours. Long-running queries may take longer time, but this can be taken care of by enabling HAProxy's TCP heartbeat feature.

```
defaults
    mode        tcp

    timeout connect 30s

    timeout client 2h
    timeout server 2h
```

Set up where drivers wanting to perform writes will connect:

```
frontend neo4j-write
    bind *:7680
    default_backend current-master
```

Now we set up the backend that points to the current master instance.

```
backend current-master
    option  httpchk HEAD /db/manage/server/ha/master HTTP/1.0

    server db01 10.0.1.10:7687 check port 7474
    server db02 10.0.1.11:7687 check port 7474
    server db03 10.0.1.12:7687 check port 7474
```

In the example above `httpchk` is configured in the way you would do it if authentication has been disabled for Neo4j. By default however, authentication is enabled and you will need to pass in an authentication header. This would be along the lines of `option  httpchk HEAD /db/manage/server/ha/master HTTP/1.0\r\nAuthorization:\ Basic\ bmVvNGo6bmVvNGo=` where the last part has to be replaced with a base64 encoded value for your username and password.

Configure where drivers wanting to perform reads will connect:

```
frontend neo4j-read
    bind *:7681
    default_backend slaves
```

Finally, configure a backend that points to slaves in a round-robin fashion:

```
backend slaves
    balance roundrobin
    option  httpchk HEAD /db/manage/server/ha/slave HTTP/1.0

    server db01 10.0.1.10:7687 check port 7474
    server db02 10.0.1.11:7687 check port 7474
    server db03 10.0.1.12:7687 check port 7474
```

Note that the servers in the `slave` backend are configured the same way as in the `current-master` backend.

Then by putting all the above configurations into one file, we get a basic workable HAProxy configuration to perform load balancing for applications using the Bolt Protocol.

By default, encryption is enabled between servers and drivers. With encryption turned on, the HAProxy configuration constructed above needs no change to work directly in TLS/SSL passthrough layout for HAProxy. However depending on the driver authentication strategy adopted, some special requirements might apply to the server certificates.

For drivers using trust-on-first-use authentication strategy, each driver would register the HAProxy port it connects to with the first certificate received from the cluster. Then for all subsequent connections, the driver would only establish connections with the server whose certificate is the same as the one registered. Therefore, in order to make it possible for a driver to establish connections with all instances in the cluster, this mode requires all the instances in the cluster sharing the same certificate.

If drivers are configured to run in trusted-certificate mode, then the certificate known to the drivers should be a root certificate to all the certificates installed on the servers in the cluster. Alternatively, for the drivers such as Java driver who supports registering multiple certificates as trusted certificates, the drivers also work well with a cluster if server certificates used in the cluster are all registered as trusted certificates.

To use HAProxy with other encryption layout, please refer to their full documentation at their website.

Configuring HAProxy for the HTTP API

HAProxy can be configured in many ways. The full documentation is available at their website.

For this example, we will configure HAProxy to load balance requests to three HA servers. Simply write the following configuration to */etc/haproxy/haproxy.cfg:*

```
global
    daemon
    maxconn 256

defaults
    mode http
    timeout connect 5000ms
    timeout client 50000ms
    timeout server 50000ms

frontend http-in
    bind *:80
    default_backend neo4j

backend neo4j
    option httpchk GET /db/manage/server/ha/available
    server s1 10.0.1.10:7474 maxconn 32
    server s2 10.0.1.11:7474 maxconn 32
    server s3 10.0.1.12:7474 maxconn 32

listen admin
    bind *:8080
    stats enable
```

HAProxy can now be started by running:

```
/usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg
```

You can connect to http://<ha-proxy-ip>:8080/haproxy?stats to view the status dashboard. This dashboard can be moved to run on port 80, and authentication can also be added. See the HAProxy documentation for details on this.

Optimizing for reads and writes

Neo4j provides a catalogue of *health check URLs* (see Endpoints for status information) that HAProxy (or any load balancer for that matter) can use to distinguish machines using HTTP response codes. In the example above we used the /available endpoint, which directs requests to machines that are generally available for transaction processing (they are alive!).

However, it is possible to have requests directed to slaves only, or to the master only. If you are able to distinguish in your application between requests that write, and requests that only read, then you can take advantage of two (logical) load balancers: one that sends all your writes to the master, and one that sends all your read-only requests to a slave. In HAProxy you build logical load balancers by adding multiple backends.

The trade-off here is that while Neo4j allows slaves to proxy writes for you, this indirection unnecessarily ties up resources on the slave and adds latency to your write requests. Conversely, you don't particularly want read traffic to tie up resources on the master; Neo4j allows you to scale out for reads, but writes are still constrained to a single instance. If possible, that instance should exclusively do writes to ensure maximum write performance.

The following example excludes the master from the set of machines using the /slave endpoint.

```
global
    daemon
    maxconn 256

defaults
    mode http
    timeout connect 5000ms
    timeout client 50000ms
    timeout server 50000ms

frontend http-in
    bind *:80
    default_backend neo4j-slaves

backend neo4j-slaves
    option httpchk GET /db/manage/server/ha/slave
    server s1 10.0.1.10:7474 maxconn 32 check
    server s2 10.0.1.11:7474 maxconn 32 check
    server s3 10.0.1.12:7474 maxconn 32 check

listen admin
    bind *:8080
    stats enable
```

In practice, writing to a slave is uncommon. While writing to slaves has the benefit of ensuring that data is persisted in two places (the slave and the master), it comes at a cost. The cost is that the slave must immediately become consistent with the master by applying any missing transactions and then synchronously apply the new transaction with the master. This is a more expensive operation than writing to the master and having the master push changes to one or more slaves.

Cache-based sharding with HAProxy

Neo4j HA enables what is called cache-based sharding. If the dataset is too big to fit into the cache of any single machine, then by applying a consistent routing algorithm to requests, the caches on each machine will actually cache different parts of the graph. A typical routing key could be user ID.

In this example, the user ID is a query parameter in the URL being requested. This will route the same user to the same machine for each request.

```
global
    daemon
    maxconn 256

defaults
    mode http
    timeout connect 5000ms
    timeout client 50000ms
    timeout server 50000ms

frontend http-in
    bind *:80
    default_backend neo4j-slaves

backend neo4j-slaves
    balance url_param user_id
    server s1 10.0.1.10:7474 maxconn 32
    server s2 10.0.1.11:7474 maxconn 32
    server s3 10.0.1.12:7474 maxconn 32

listen admin
    bind *:8080
    stats enable
```

Naturally the health check and query parameter-based routing can be combined to only route requests to slaves by user ID. Other load balancing algorithms are also available, such as routing by source IP (`source`), the URI (`uri`) or HTTP headers(`hdr()`).

2.5. Post-installation tasks

This section covers tasks to perform after installation and when running Neo4j for the first time.

Once Neo4j is installed the following tasks may need to be performed to complete the setup.

2.5.1. Waiting for Neo4j to start

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

The details of how to poll depend:

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

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

Here is an example of polling written in Bash using the HTTP endpoint, with encryption and authentication disabled.

```
end="$((SECONDS+10))"
while true; do
    [[ "200" = "$(curl --silent --write-out %{http_code} --output /dev/null http://localhost:7474)" ]] &&
break
    [[ "${SECONDS}" -ge "${end}" ]] && exit 1
    sleep 1
done
```

2.5.2. Setting the number of open files

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

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

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

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

If you opted to run the neo4j service as a different user, change the first field in step 2 accordingly.

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

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

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

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

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

   ```
   session    required   pam_limits.so
   ```

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

   After the above procedure, the neo4j user will have a limit of 40 000 simultaneous open files. If you continue experiencing exceptions on `Too many open files` or `Could not stat() directory`, you may have to raise the limit further.

2.5.3. Setup for remote debugging

In order to configure the Neo4j server for remote debugging sessions, the Java debugging parameters need to be passed to the Java process through the configuration. They live in the *conf/neo4j-wrapper.conf* file.

In order to specify the parameters, add a line for the additional Java arguments like this:

```
dbms.jvm.additional=-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=5005
```

This configuration will start a Neo4j server ready for remote debugging attachement at localhost and port `5005`. Use these parameters to attach to the process from Eclipse, IntelliJ or your remote debugger of choice after starting the server.

Usage Data Collector

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

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

The data collected is clearly stated here. If any future versions of this system collect additional data, we will clearly announce those changes.

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

Technical Information

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

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

- The JVM heap size.
- The number of nodes, relationships, labels and properties in the database.

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

How to disable UDC

UDC is easily turned off by disabling it in the database configuration, in *neo4j.conf* for Neo4j server or in the configuration passed to the database in embedded mode. See UDC Configuration in the configuration section for details.

2.5.4. Configuring Neo4j connectors

Three different Neo4j connectors are configured by default:

- One Bolt connector with the default name of `bolt`.
- One HTTP connector with the default name of `http`.
- One HTTPS connector with the default name of `https`.

The following shows the default configuration for connectors:

```
# Bolt connector
dbms.connector.bolt.type=BOLT
dbms.connector.bolt.enabled=true
dbms.connector.bolt.tls_level=OPTIONAL
# To have Bolt accept non-local connections, uncomment this line
# dbms.connector.bolt.address=0.0.0.0:7687

# HTTP Connector
dbms.connector.http.type=HTTP
dbms.connector.http.enabled=true
#dbms.connector.http.encryption=NONE
# To have HTTP accept non-local connections, uncomment this line
#dbms.connector.http.address=0.0.0.0:7474

# HTTPS Connector
dbms.connector.https.type=HTTP
dbms.connector.https.enabled=true
dbms.connector.https.encryption=TLS
dbms.connector.https.address=localhost:7473
```

The sections below describe the connectors and how they can be modified.

Bolt

Bolt connectors are ports that accept connections via the Bolt database protocol. This is the protocol used by the official Neo4j drivers. There must be at least one Bolt driver.

Neo4j can also be configured for multiple Bolt connectors which allows for separate remote and local connections that may have different encryption requirements. Each connector has a unique name to identify it, denoted `<bolt-connector-name>` below. For example, a connector intended for external use may be named "bolt-public". The name of the Bolt driver in the default configuration is `bolt`.

Table 4. Configuration options for Bolt connectors. `<bolt-connector-name>` is a placeholder for a unique name for the connector.

Name	Description	Valid values	Default value
dbms.connector.<bolt-connector-name>.address	Address the connector should bind to.	`<host>:<port>`	`localhost:7687`
dbms.connector.<bolt-connector-name>.enabled	Enable this connector.	`true` or `false`	`false`
dbms.connector.<bolt-connector-name>.tls_level	Encryption level to require this connector to use.	`REQUIRED`, `OPTIONAL`, or `DISABLED`	`OPTIONAL`
dbms.connector.<bolt-connector-name>.type	Connector type.	`BOLT` or `HTTP`	The value `BOLT` is mandatory to configure the Bolt connector.

HTTP

HTTP connectors expose Neo4j's HTTP endpoints. HTTPS connectors are configured by setting a connector to require encryption. There must be exactly one HTTP connector and zero or one HTTPS connectors.

Each connector has a unique name to identify it, denoted `<http-connector-name>` below. For example, a connector intended for external use may be named "http-public". The name of the HTTP driver in the default configuration is `http`, and the name of the HTTPS driver in the default configuration is `https`.

Table 5. Configuration options for HTTP connectors. `<http-connector-name>` is a placeholder for a unique name for the connector.

Name	Description	Valid values	Default value
dbms.connector.<http-connector-name>.address	Address the connector should bind to.	`<host>:<port>`	`localhost:7474`
dbms.connector.<http-connector-name>.enabled	Enable this connector.	`true` or `false`	`false`
dbms.connector.<http-connector-name>.encryption	Enable TLS for this connector.	`NONE` or `TLS`	`NONE`
dbms.connector.<http-connector-name>.type	Connector type.	`BOLT` or `HTTP`	The value `HTTP` is mandatory to configure the HTTP connector.

2.5.5. Transaction timeouts

The *execution guard* is a feature that terminates transactions whose execution time has exceeded the configured timeout.

We now define how to configure this feature.

Enable the execution guard

The execution guard is enabled as follows: `unsupported.dbms.executiontime_limit.enabled=true`. This will cause all transactions to have a default timeout — set by `dbms.transaction.timeout` — and any transaction executing for longer than the configured time interval will be terminated.

When the execution guard is disabled, all timeout settings will be ignored, which means no transactions will be terminated.

Set the transaction timeout

The `dbms.transaction.timeout` setting defines the maximum time interval of a transaction within

which it should be completed. Provided the execution guard is enabled, all transactions whose execution exceeds this time interval will be terminated.

2.6. Upgrade

This section covers upgrading Neo4j from an earlier version.

2.6.1. Single-instance upgrade

This section describes upgrading a *single* Neo4j instance. To upgrade a *Neo4j HA cluster (Neo4j Enterprise Edition)*, a very specific procedure must be followed. Please see Neo4j cluster upgrade.

Throughout this instruction, the files used to store the Neo4j data are referred to as `database files`. These files is are found in the directory specified by `dbms.directories.data` in *neo4j.conf*.

Disk space requirements

An upgrade requires substantial free disk space, as it makes an entire copy of the database. The upgraded database may also require larger data files overall.

It is recommended to make available an extra 50% disk space on top of the existing database files. Determine the pre-upgrade database size by summing up the sizes of the `NEO4J_HOME/data/databases/graph.db/*store.db*` files. Then add 50% to this number.

In addition to this, don't forget to reserve the disk space needed for the pre-upgrade backup.

Supported upgrade paths

Before upgrading to a new major or minor release, the database must first be upgraded to the latest version within the relevant release. The latest version is available at this page: http://neo4j.com/download/other-releases. The following Neo4j upgrade paths are supported:

- 2.0.latest 3.0.8
- 2.1.latest 3.0.8
- 2.2.latest 3.0.8
- 2.3.latest 3.0.8
- 3.0.any 3.0.8

Upgrade instructions

Upgrade from 2.x

1. Cleanly shut down the database if it is running.
2. Make a backup copy of the database files. If using the online backup tool available with Neo4j Enterprise Edition, ensure that backups have completed successfully.
3. Install Neo4j 3.0.8.
4. Review the settings in the configuration files of the previous installation and transfer any custom settings to the 3.0.8 installation. Since many settings have been changed between Neo4j 2.x and 3.0.8, it is advisable to use the `config-migrator` to migrate the config files for you. The `config-migrator` can be found in the *tools* directory, and can be invoked with a command like: `java -jar config-migrator.jar path/to/neo4j2.3 path/to/neo4j3.0`. Take note of any warnings printed, and

manually review the edited config files produced.

5. Import your data from the old installation using `neo4j-admin import --mode=database --database=<database-name> --from=<source-directory>`.
6. If the database is not called *graph.db*, set `dbms.active_database` in *neo4j.conf* to the name of the database.
7. Set `dbms.allow_format_migration=true` in *neo4j.conf* of the 3.0.8 installation. Neo4j will fail to start without this configuration.
8. Start up Neo4j 3.0.8.
9. The database upgrade will take place during startup.
10. Information about the upgrade and a progress indicator are logged into *debug.log*.
11. When upgrade has finished, the `dbms.allow_format_migration` should be set to `false` or be removed.
12. It is good practice to make a full backup immediately after the upgrade.

Cypher compatibility

The Cypher language may evolve between Neo4j versions. For backward compatibility, Neo4j provides directives which allow explicitly selecting a previous Cypher language version. This is possible to do globally or for individual statements, as described in the Neo4j Developer Manual *(http://neo4j.com/docs/developer-manual/3.0)*.

Upgrade from 3.x

1. Cleanly shut down the database if it is running.
2. Make a backup copy of the database files. If using the online backup tool available with Neo4j Enterprise Edition, ensure that backups have completed successfully.
3. Install Neo4j 3.0.8.
4. Review the settings in the configuration files of the previous installation and transfer any custom settings to the 3.0.8 installation.
5. Wen using the default *data* directory, copy it from the old installation to the new. If databases are stored in a custom location, configure dbms.directories.data for the new installation to point to this custom location.
6. If the database is not called *graph.db*, set `dbms.active_database` in *neo4j.conf* to the name of the database.
7. Set `dbms.allow_format_migration=true` in *neo4j.conf* of the 3.0.8 installation. Neo4j will fail to start without this configuration.
8. Start up Neo4j 3.0.8.
9. The database upgrade will take place during startup.
10. Information about the upgrade and a progress indicator are logged into *debug.log*.
11. When upgrade has finished, the `dbms.allow_format_migration` should be set to `false` or be removed.
12. It is good practice to make a full backup immediately after the upgrade.

2.6.2. Neo4j cluster upgrade

This section covers upgrading a Neo4j Highly Available cluster.

Upgrading a Neo4j HA cluster to Neo4j 3.0.8 requires following a specific process in order to ensure that the cluster remains consistent, and that all cluster instances are able to join and participate in the cluster following their upgrade. Neo4j 3.0.8 does not support rolling upgrades.

Back up the Neo4j database

- Before starting any upgrade procedure, it is *very important* to make a full backup of your database.
- For detailed instructions on backing up your Neo4j database, refer to the backup guide.

Shut down the cluster

- Shut down the slave instances one by one.
- Shut down the master last.

Upgrade the master

1. Install Neo4j 3.0.8 on the master, keeping the database files untouched.
2. Disable HA in the configuration, by setting `dbms.mode=SINGLE` in *neo4j.conf*.
3. Upgrade as described for a single instance of Neo4j
4. When upgrade has finished, shut down Neo4j again.
5. Re-enable HA in the configuration by setting `dbms.mode=HA` in *neo4j.conf*.
6. Make a full backup of the Neo4j database. Please note that backups from before the upgrade are no longer valid for update via the incremental online backup. Therefore it is important to perform a *full backup*, using an empty target directory, at this point.

Upgrade the slaves

On each slave:

1. Remove all database files.
2. Install Neo4j 3.0.8.
3. Review the settings in the configuration files in the previous installation, and transfer any custom settings to the 3.0.8 installation. Be aware of settings that have changed name between versions.
4. If the database is not called *graph.db*, set `dbms.active_database` in *neo4j.conf* to the name of the database.
5. If applicable, copy the security configuration from the master, since this is not propagated automatically.

At this point it is an alternative to manually copy database files from the master to the slaves. Doing so will avoid the need to sync from the master when starting. This can save considerable time when upgrading large databases.

Restart the cluster

1. Start the master instance.
2. Start the slaves, one by one. Once a slave has joined the cluster, it will sync the database from the master instance.

Chapter 3. Security

This chapter covers securing Neo4j.

3.1. Securing Neo4j server

3.1.1. Secure the port and remote client connection accepts

By default, the Neo4j Server is bundled with a Web server that binds to host `localhost` on port `7474`, answering only requests from the local machine.

This is configured in *neo4j.conf*:

```
# Let the webserver only listen on the specified IP. Default is localhost (only
# accept local connections). Uncomment to allow any connection.
dbms.connector.http.type=HTTP
dbms.connector.http.enabled=true
#dbms.connector.http.address=0.0.0.0:7474
```

If you want the server to listen to external hosts, configure the Web server in *neo4j.conf* by setting the property dbms.connector.http.address=0.0.0.0:7474 which will cause the server to bind to all available network interfaces. Note that firewalls et cetera have to be configured accordingly as well.

3.1.2. Server authentication and authorization

Neo4j requires clients to supply authentication credentials when accessing the REST API. Without valid credentials, access to the database will be forbidden.

The authentication and authorization data is stored under *data/dbms/auth*. If necessary, this file can be copied over to other neo4j instances to ensure they share the same username/password.

When accessing Neo4j over unsecured networks, make sure HTTPS is configured and used for access (see HTTPS support).

If necessary, authentication may be disabled. This will allow any client to access the database without supplying authentication credentials.

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

Disabling authentication is not recommended, and should only be done if the operator has a good understanding of their network security, including protection against cross-site scripting (XSS) *(http://en.wikipedia.org/wiki/Cross-site_scripting)* attacks via web browsers. Developers should not disable authentication if they have a local installation using the default listening ports.

3.1.3. HTTPS support

This section introduces HTTPS and certificates in Neo4j server.

The Neo4j server includes built in support for SSL encrypted communication over HTTPS. The first time the server starts, it automatically generates a self-signed SSL certificate and a private key.

Because the certificate is self signed, it is not safe to rely on for production use. Instead, you should provide your own key and certificate for the server to use.

Using auto-generation of self-signed SSL certificates will not work if the Neo4j server has been configured with multiple connectors that bind to different IP addresses. If you need to use multiple IP addresses, please configure certificates manually and use multi-host or wildcard certificates instead.

To provide your own key and certificate, put the files *neo4j.key* and *neo4j.cert* in the *certificates* directory. Note that the files must be named exactly *neo4j.key* and *neo4j.cert*. The location of the directory can be configured by setting dbms.directories.certificates in *neo4j.conf*.

```
# Certificates location (auto generated if the file does not exist)
dbms.directories.certificates=certificates
```

Note that the key should be unencrypted. Make sure you set correct permissions on the private key, so that only the Neo4j server user can read/write it.

Neo4j also supports chained SSL certificates. This requires to have all certificates in PEM format combined in one file and the private key needs to be in DER format.

You can set what port the HTTPS connector should bind to in the same configuration file, as well as turn HTTPS on or off:

```
dbms.connector.https.type=HTTP
dbms.connector.https.enabled=true
dbms.connector.https.encryption=TLS
dbms.connector.https.address=localhost:7473
```

3.1.4. Server authorization rules

This section discusses authorization rules for restricting access to Neo4j server.

Administrators may require more fine-grained security policies in addition to the basic authorization and/or IP-level restrictions on the Web server. Neo4j server supports administrators in allowing or disallowing access the specific aspects of the database based on credentials that users or applications provide.

To facilitate domain-specific authorization policies in Neo4j Server, security rules can be implemented and registered with the server. This makes scenarios like user and role based security and authentication against external lookup services possible. See `org.neo4j.server.rest.security.SecurityRule` in the javadocs downloadable from Maven Central (org.neo4j.app:neo4j-server) *(http://search.maven.org/#search%7Cgav%7C1%7Cg%3A%22org.neo4j.app%22%20AND%20a%3A%22neo4j-server%22)*.

The use of Server Authorization Rules may interact unexpectedly with the built-in authentication and authorization (see Server authentication and authorization), if enabled.

Enforcing server authorization rules

In this example, a (dummy) failing security rule is registered to deny access to all URIs to the server by listing the rules class in *neo4j.conf*:

```
org.neo4j.server.rest.security_rules=my.rules.PermanentlyFailingSecurityRule
```

with the rule source code of:

```
public class PermanentlyFailingSecurityRule implements SecurityRule
{

    public static final String REALM = "WallyWorld"; // as per RFC2617 :-)

    @Override
    public boolean isAuthorized( HttpServletRequest request )
    {
        return false; // always fails - a production implementation performs
                      // deployment-specific authorization logic here
    }

    @Override
    public String forUriPath()
    {
        return "/*";
    }

    @Override
    public String wwwAuthenticateHeader()
    {
        return SecurityFilter.basicAuthenticationResponse(REALM);
    }
}
```

With this rule registered, any access to the server will be denied. In a production-quality implementation the rule will likely lookup credentials/claims in a 3rd-party directory service (e.g. LDAP) or in a local database of authorized users.

Example request

- **POST** http://localhost:7474/db/data/node
- **Accept:** application/json; charset=UTF-8

Example response

- **401:** Unauthorized
- **WWW-Authenticate:** Basic realm="WallyWorld"

Using wildcards to target security rules

In this example, a security rule is registered to deny access to all URIs to the server by listing the rule(s) class(es) in *neo4j.conf*. In this case, the rule is registered using a wildcard URI path (where `*` characters can be used to signify any part of the path). For example `/users*` means the rule will be bound to any resources under the `/users` root path. Similarly `/users*type*` will bind the rule to resources matching URIs like `/users/fred/type/premium`.

```
org.neo4j.server.rest.security_rules=my.rules.PermanentlyFailingSecurityRuleWithWildcardPath
```

with the rule source code of:

```
public String forUriPath()
{
    return "/protected/*";
}
```

With this rule registered, any access to URIs under `/protected/` will be denied by the server. Using wildcards allows flexible targeting of security rules to arbitrary parts of the server's API, including any unmanaged extensions or managed plugins that have been registered.

Example request

- **GET** http://localhost:7474/protected/tree/starts/here/dummy/more/stuff
- **Accept:** application/json

Example response

- **401:** Unauthorized
- **WWW-Authenticate:** Basic realm="WallyWorld"

Using complex wildcards to target security rules

In this example, a security rule is registered to deny access to all URIs matching a complex pattern. The config looks like this:

```
org.neo4j.server.rest.security_rules=my.rules.PermanentlyFailingSecurityRuleWithComplexWildcardPath
```

with the rule source code of:

```
public class PermanentlyFailingSecurityRuleWithComplexWildcardPath implements SecurityRule
{

    public static final String REALM = "WallyWorld"; // as per RFC2617 :-)

    @Override
    public boolean isAuthorized( HttpServletRequest request )
    {
        return false;
    }

    @Override
    public String forUriPath()
    {
        return "/protected/*/something/else/*/final/bit";
    }

    @Override
    public String wwwAuthenticateHeader()
    {
        return SecurityFilter.basicAuthenticationResponse(REALM);
    }
}
```

Example request

- **GET** http://localhost:7474/protected/wildcard_replacement/x/y/z/something/else/more_wildcard_replacement/a/b/c/final/bit/more/stuff
- **Accept:** application/json

Example response

- **401:** Unauthorized
- **WWW-Authenticate:** Basic realm="WallyWorld"

3.1.5. Using a proxy

This section discusses placing a proxy in front of Neo4j server for additional access control.

Although the Neo4j server has a number of security features built-in (see the above chapters), for sensitive deployments it is often sensible to front against the outside world it with a proxy like Apache `mod_proxy` [1: http://httpd.apache.org/docs/2.2/mod/mod_proxy.html].

This provides a number of advantages:

- Control access to the Neo4j server to specific IP addresses, URL patterns and IP ranges. This can be used to make for instance only the `/db/data` namespace accessible to non-local clients, while the `/db/admin` URLs only respond to a specific IP address.

  ```
  <Proxy *>
    Order Deny,Allow
    Deny from all
    Allow from 192.168.0
  </Proxy>
  ```

  While it is possible to develop plugins using Neo4j's `SecurityRule` (see above), operations professionals would often prefer to configure proxy servers such as Apache. However, it should be noted that in cases where both approaches are being used, they will work harmoniously provided that the behavior is consistent across proxy server and `SecurityRule` plugins.

- Run Neo4j Server as a non-root user on a Linux/Unix system on a port < 1000 (e.g. port 80) using

  ```
  ProxyPass /neo4jdb/data http://localhost:7474/db/data
  ProxyPassReverse /neo4jdb/data http://localhost:7474/db/data
  ```

- Simple load balancing in a clustered environment to load-balance read load using the Apache `mod_proxy_balancer` [2: http://httpd.apache.org/docs/2.2/mod/mod_proxy_balancer.html] plugin

  ```
  <Proxy balancer://mycluster>
  BalancerMember http://192.168.1.50:80
  BalancerMember http://192.168.1.51:80
  </Proxy>
  ProxyPass /test balancer://mycluster
  ```

3.1.6. LOAD CSV

This section discusses file access behavior when using the Neo4j Cypher clause `LOAD CSV` and how to configure it.

The Cypher `LOAD CSV` clause can be used to import CSV files over the network or from the local file system. When reading from the file system the `file:///` URL that is used is resolved relative to the directory configured by dbms.directories.import. The default value is `import`. This is a security measure which prevents the database from accessing files outside of the standard import directory.

To remove this security measure and allow access to any file on the system, set dbms.directories.import to be empty.

The related dbms.security.allow_csv_import_from_file_urls setting can be set to `false` to completely disable access to the file system for `LOAD CSV`.

To review all security-related configuration settings see the Configuration settings reference.

Chapter 4. Import

This chapter covers importing data into Neo4j.

The import tool is used to create a new Neo4j database from data in CSV files.

This chapter explains how to use the tool and format the input data. For in-depth examples of using the import tool, see Use the Import tool.

These are some things you will need to keep in mind when creating your input files:

- Fields are comma separated by default but a different delimiter can be specified.
- All files must use the same delimiter.
- Multiple data sources can be used for both nodes and relationships.
- A data source can optionally be provided using multiple files.
- A header which provides information on the data fields must be on the first row of each data source.
- Fields without corresponding information in the header will not be read.
- UTF-8 encoding is used.

Indexes are not created during the import. Instead, you will need to add indexes afterwards (see Developer Manual Indexes *(http://neo4j.com/docs/developer-manual/3.0/introduction/#graphdb-neo4j-schema-indexes)*).

Data cannot be imported into an existing database using this tool. If you want to load small to medium sized CSV files use `LOAD CSV` (see Developer Manual `LOAD CSV` *(http://neo4j.com/docs/developer-manual/3.0/cypher/#query-load-csv)*).

4.1. CSV file header format

This section explains the header format of CSV files when using the Neo4j import tool.

The header row of each data source specifies how the fields should be interpreted. The same delimiter is used for the header row as for the rest of the data.

The header contains information for each field, with the format: `<name>:<field_type>`. The `<name>` is used as the property key for values, and ignored in other cases. The following `<field_type>` settings can be used for both nodes and relationships:

Property value

Use one of `int`, `long`, `float`, `double`, `boolean`, `byte`, `short`, `char`, `string` to designate the data type. 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`.

IGNORE

Ignore this field completely.

See below for the specifics of node and relationship data source headers.

4.1.1. Nodes

The following field types do additionally apply to node data sources:

ID
: Each node must have a unique id which is used during the import. The ids are used to find the correct nodes when creating relationships. Note that the id has to be unique across all nodes in the import, even nodes with different labels.

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`.

4.1.2. Relationships

For relationship data sources, there are three mandatory fields:

TYPE
: The relationship type to use for the relationship.

START_ID
: The id of the start node of the relationship to create.

END_ID
: The id of the end node of the relationship to create.

4.1.3. ID spaces

The import tool assumes that node identifiers are unique across node files. If this is not the case then we can define an `id space`. Id spaces are defined in the `ID` field of node files.

For example, to specify the `Person` id space we would use the field type `ID(Person)` in our persons node file. We also need to reference that id space in our relationships file i.e. `START_ID(Person)` or `END_ID(Person)`.

4.2. Command line usage

This section covers how to use the Neo4j import tool from the command line.

4.2.1. Linux

Under Unix/Linux/OSX, the command is named `neo4j-import`. Depending on the installation type, the tool is either available globally, or used by executing `./bin/neo4j-import` from inside the installation directory.

4.2.2. Windows

Under Windows, used by executing `bin\neo4j-import` from inside the installation directory.

For help with running the import tool under Windows, see the reference in Windows.

4.2.3. Options

--into <store-dir>

Database directory to import into. Must not contain existing database.

--nodes[:Label1:Label2] "<file1>,<file2>,..."

Node CSV header and data. Multiple files will be logically seen as one big file from the perspective of the importer. The first line must contain the header. Multiple data sources like these can be specified in one import, where each data source has its own header. Note that file groups must be enclosed in quotation marks.

--relationships[:RELATIONSHIP_TYPE] "<file1>,<file2>,..."

Relationship CSV header and data. Multiple files will be logically seen as one big file from the perspective of the importer. The first line must contain the header. Multiple data sources like these can be specified in one import, where each data source has its own header. Note that file groups must be enclosed in quotation marks.

--delimiter <delimiter-character>

Delimiter character, or *TAB*, between values in CSV data. The default option is `,`.

--array-delimiter <array-delimiter-character>

Delimiter character, or *TAB*, between array elements within a value in CSV data. The default option is `;`.

--quote <quotation-character>

Character to treat as quotation character for values in CSV data. The default option is `"`. Quotes inside quotes escaped like `"""Go away"", he said."` and `"\"Go away\", he said."` are supported. If you have set `'` to be used as the quotation character, you could write the previous example like this instead: `'"Go away", he said.'`

--multiline-fields <true/false>

Whether or not fields from input source can span multiple lines, i.e. contain newline characters. Default value: false

--input-encoding <character set>

Character set that input data is encoded in. Provided value must be one out of the available character sets in the JVM, as provided by `Charset#availableCharsets()`. If no input encoding is provided, the default character set of the JVM will be used.

--ignore-empty-strings <true/false>

Whether or not empty string fields ("") from input source are ignored, i.e. treated as null. Default value: false

--id-type <id-type>

One out of [STRING, INTEGER, ACTUAL] and specifies how ids in node/relationship input files are treated. STRING: arbitrary strings for identifying nodes. INTEGER: arbitrary integer values for identifying nodes. ACTUAL: (advanced) actual node ids. Default value: STRING

--processors <max processor count>

(advanced) Max number of processors used by the importer. Defaults to the number of available processors reported by the JVM. There is a certain amount of minimum threads needed so for that reason there is no lower bound for this value. For optimal performance this value shouldn't be greater than the number of available processors.

--stacktrace <true/false>

Enable printing of error stack traces.

--bad-tolerance <max number of bad entries>

Number of bad entries before the import is considered failed. This tolerance threshold is about relationships referring to missing nodes. Format errors in input data are still treated as errors. Default value: 1000

--skip-bad-relationships <true/false>

Whether or not to skip importing relationships that refers to missing node ids, i.e. either start or end node id/group referring to node that wasn't specified by the node input data. Skipped nodes will be logged, containing at most number of entities specified by bad-tolerance. Default value: true

--skip-duplicate-nodes <true/false>

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 number of entities specified by bad-tolerance. Default value: false

--ignore-extra-columns <true/false>

Whether or not to ignore extra columns in the data not specified by the header. Skipped columns will be logged, containing at most number of entities specified by bad-tolerance. Default value: false

--db-config <path/to/neo4j.conf>

(advanced) File specifying database-specific configuration. For more information consult manual about available configuration options for a neo4j configuration file. Only configuration affecting store at time of creation will be read. Examples of supported config are:

- dbms.relationship_grouping_threshold
- `unsupported.dbms.block_size.strings`
- `unsupported.dbms.block_size.array_properties`

4.2.4. Verbose error information

In some cases if an unexpected error occurs it might be useful to supply the command line option `--stacktrace` to the import (and rerun the import to actually see the additional information). This will have the error printed with additional debug information, useful for both developers and issue reporting.

4.2.5. Output and statistics

While an import is running through its different stages, some statistics and figures are printed in the console. The general interpretation of that output is to look at the horizontal line, which is divided up into sections, each section representing one type of work going on in parallel with the other sections. The wider a section is, the more time is spent there relative to the other sections, the widest being the bottleneck, also marked with `*`. If a section has a double line, instead of just a single line, it means that multiple threads are executing the work in that section. To the far right a number is displayed telling how many entities (nodes or relationships) have been processed by that stage.

As an example:

```
[*>:20,25 MB/s-----------------|PREPARE(3)====================|RELATIONSHIP(2)===============]
16M
```

Would be interpreted as:

- `>` data being read, and perhaps parsed, at `20,25 MB/s`, data that is being passed on to ...
- `PREPARE` preparing the data for ...

- `RELATIONSHIP` creating actual relationship records and ...
- `v` writing the relationships to the store. This step is not visible in this example, because it is so cheap compared to the other sections.

Observing the section sizes can give hints about where performance can be improved. In the example above, the bottleneck is the data read section (marked with `>`), which might indicate that the disk is being slow, or is poorly handling simultaneous read and write operations (since the last section often revolves around writing to disk).

Chapter 5. Backup

This chapter covers how to perform and restore backups of a Neo4j database.

The backup features are available in the Neo4j Enterprise Edition.

5.1. Introducing backups

Backing up your Neo4j database to remote or offline storage is a fundamental part of operational hygiene. Neo4j supports both full and incremental backups. The backup procedure is the same for a stand-alone database and for an Highly Available cluster.

Backups are performed over the network, from a running Neo4j server and into a local copy of the database store. The backup is run using the `neo4j-backup` tool which is available in the Neo4j Enterprise edition.

5.1.1. Enabling backups

Two parameters must be configured in order to perform backups.

- `dbms.backup.enabled=true` will enable backups; this is the default value.
- `dbms.backup.address=<hostname or IP address>:6362` configures the interface and port that the backup service listens on. The value of the parameter defaults to the loopback interface and port 6362. It can also be configured to listen on all interfaces by setting `dbms.backup.address=0.0.0.0:6362`.

5.1.2. Storage considerations

For any backup it is important that the data is stored separately from the production system where there are no common dependencies. It is advisable to keep the backup on stable storage outside of the cluster servers, on different (network attached) storage, and preferably off site (for example to the cloud, a different availability zone within the same cloud, or a separate cloud). Since backups are kept for a long time, the longevity of archival storage should be considered as part of backup planning.

See Configuration settings reference for detailed documentation on available configuration options.

5.2. Perform a backup

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

5.2.1. Backup commands

Backups are performed using the `neo4j-backup` tool.

Backuo a database

```
# Performing a full backup: create a blank directory and run the backup tool
mkdir /mnt/backup/neo4j-backup
./bin/neo4j-backup -host 192.168.1.34 -to /mnt/backup/neo4j-backup

# Performing an incremental backup: just specify the location of your previous backup
./bin/neo4j-backup -host 192.168.1.34 -to /mnt/backup/neo4j-backup

# Performing an incremental backup where the service is listening on a non-default port
./bin/neo4j-backup -host 192.168.1.34 -port 9999 -to /mnt/backup/neo4j-backup
```

Now if you do a directory listing of `/mnt/backup/neo4j-backup` you will see that you have a complete set of Neo4j database files in that location.

5.2.2. Incremental backups

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

The incremental backup may fail for a number of reasons:

- If the existing directory doesn't contain a valid backup.
- If the existing directory contains a backup of a different database store.
- If the existing directory contains a backup from a previous database version.

Note that when copying the outstanding transactions, the server needs access to the transaction logs. These logs are kept by Neo4j and automatically removed after a period of time, based on the parameter dbms.tx_log.rotation.retention_policy. If the required transaction logs have already been removed, the backup tool will do a full backup instead.

5.3. Restore a backup

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

5.3.1. Restore a single database

The Neo4j backups are fully functional databases. To restore a backup the database must be shut down. Replace the files in the data directory with the backup. Then start the database.

5.3.2. Restore an HA cluster

To restore from backup in a clustered environment, follow these steps:

1. Shut down all database instances in the cluster.
2. Restore the backup to the individual database folders.
3. Start the database instances.

Chapter 6. Monitoring

This chapter covers how to use Neo4j monitoring facilities to log and display various metrics.

The monitoring features are available in the Neo4j Enterprise Edition.

Neo4j can be configured to report metrics in two different ways:

- Export metrics to csv files.
- Send metrics to Graphite or any monitoring tool based on the Graphite protocol.

6.1. Enabling metrics logging

Neo4j can expose metrics for the following parts of the database:

```
# Setting for enabling all supported metrics.
metrics.enabled=true

# Setting for enabling all Neo4j specific metrics.
metrics.neo4j.enabled=true

# Setting for exposing metrics about transactions; number of transactions started, committed, etc.
metrics.neo4j.tx.enabled=true

# Setting for exposing metrics about the Neo4j page cache; page faults, evictions, flushes and exceptions,
etc.
metrics.neo4j.pagecache.enabled=true

# Setting for exposing metrics about approximately entities are in the database; nodes, relationships,
properties, etc.
metrics.neo4j.counts.enabled=true

# Setting for exposing metrics about the network usage of the HA cluster component.
metrics.neo4j.network.enabled=true
```

6.1.1. Graphite

Add the following settings to *neo4j.conf* in order to enable integration with Graphite:

```
# Enable the Graphite integration. Default is 'false'.
metrics.graphite.enabled=true
# The IP and port of the Graphite server on the format <hostname or IP address>:<port number>.
# The default port number for Graphite is 2003.
metrics.graphite.server=localhost:2003
# How often to send data. Default is 3 minutes.
metrics.graphite.interval=3m
# Prefix for Neo4j metrics on Graphite server.
metrics.prefix=Neo4j_1
```

Start Neo4j and connect to Graphite via a web browser in order to monitor your Neo4j metrics.

6.1.2. 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 'false'.
metrics.csv.enabled=true
# Directory path for output files.
# Default is a "metrics" directory under NEO4J_HOME.
#metrics.csv.path='/local/file/system/path'
# How often to store data. Default is 3 minutes.
metrics.csv.interval=3m
```

The csv exporter does not automatically rotate the output files. When enabling the csv exporter, it is recommended to configure a job to periodically archive the files.

6.2. Available metrics

Table 6. Database CheckPointing Metrics

Name	Description
neo4j.check_point.events	The total number of check point events executed so far
neo4j.check_point.total_time	The total time spent in check pointing so far
neo4j.check_point.check_point_duration	The duration of the check point event

Table 7. Database Data Metrics

Name	Description
neo4j.ids_in_use.relationship_type	The total number of different relationship types stored in the database
neo4j.ids_in_use.property	The total number of different property names used in the database
neo4j.ids_in_use.relationship	The total number of relationships stored in the database
neo4j.ids_in_use.node	The total number of nodes stored in the database

Table 8. Database PageCache Metrics

Name	Description
neo4j.page_cache.eviction_exceptions	The total number of exceptions seen during the eviction process in the page cache
neo4j.page_cache.flushes	The total number of flushes executed by the page cache
neo4j.page_cache.unpins	The total number of page unpins executed by the page cache
neo4j.page_cache.pins	The total number of page pins executed by the page cache
neo4j.page_cache.evictions	The total number of page evictions executed by the page cache
neo4j.page_cache.page_faults	The total number of page faults happened in the page cache

Table 9. Database Transaction Metrics

Name	Description
neo4j.transaction.started	The total number of started transactions
neo4j.transaction.peak_concurrent	The highest peak of concurrent transactions ever seen on this machine

Name	Description
neo4j.transaction.active	The number of currently active transactions
neo4j.transaction.active_read	The number of currently active read transactions
neo4j.transaction.active_write	The number of currently active write transactions
neo4j.transaction.committed	The total number of committed transactions
neo4j.transaction.committed_read	The total number of committed read transactions
neo4j.transaction.committed_write	The total number of committed write transactions
neo4j.transaction.rollbacks	The total number of rolled back transactions
neo4j.transaction.rollbacks_read	The total number of rolled back read transactions
neo4j.transaction.rollbacks_write	The total number of rolled back write transactions
neo4j.transaction.terminated	The total number of terminated transactions
neo4j.transaction.terminated_read	The total number of terminated read transactions
neo4j.transaction.terminated_write	The total number of terminated write transactions
neo4j.transaction.last_committed_tx_id	The ID of the last committed transaction
neo4j.transaction.last_closed_tx_id	The ID of the last closed transaction

Table 10. Cypher Metrics

Name	Description
neo4j.cypher.replan_events	The total number of times Cypher has decided to re-plan a query

Table 11. Database LogRotation Metrics

Name	Description
neo4j.log_rotation.events	The total number of transaction log rotations executed so far
neo4j.log_rotation.total_time	The total time spent in rotating transaction logs so far
neo4j.log_rotation.log_rotation_duration	The duration of the log rotation event

Table 12. Network Metrics

Name	Description
neo4j.network.slave_network_tx_writes	The amount of bytes transmitted on the network containing the transaction data from a slave to the master in order to be committed
neo4j.network.master_network_store_writes	The amount of bytes transmitted on the network while copying stores from a machines to another
neo4j.network.master_network_tx_writes	The amount of bytes transmitted on the network containing the transaction data from a master to the slaves in order to propagate committed transactions

Table 13. Cluster Metrics

Name	Description
`neo4j.cluster.slave_pull_updates`	The total number of update pulls executed by this instance
`neo4j.cluster.slave_pull_update_up_to_tx`	The highest transaction id that has been pulled in the last pull updates by this instance
`neo4j.cluster.is_master`	Whether or not this instance is the master in the cluster
`neo4j.cluster.is_available`	Whether or not this instance is available in the cluster

6.2.1. Java Virtual Machine Metrics

These metrics are environment dependent and they may vary on different hardware and with JVM configurations. Typically these metrics will show information about garbage collections (for example the number of events and time spent collecting), memory pools and buffers, and finally the number of active threads running.

Chapter 7. Performance

This chapter describes factors that affect operational performance and how to tune Neo4j for optimal throughput.

7.1. Memory tuning

This section covers how to configure memory for a Neo4j instance. The various memory requirements and trade-offs are explained as well as the characteristics of garbage collection.

Neo4j will automatically configure default values for memory-related configuration parameters that are not explicitly defined within its configuration on startup. In doing so, it will assume that all of the RAM on the machine is available for running Neo4j.

There are three types of memory to consider: OS Memory, Page Cache and Heap Space.

Please notice that the OS memory is not explicitly configurable, but is "what is left" when done specifying page cache and heap space. If configuring page cache and heap space equal to or greater than the available RAM, or if not leaving enough head room for the OS, the OS will start swapping to disk, which will heavily affect performance. Therefore, follow this checklist:

1. Plan OS memory sizing
2. Plan page cache sizing
3. Plan heap sizing
4. Do the sanity check:

```
Actual OS allocation = available RAM - (page cache + heap size)
```

Make sure that your system is configured such that it will never need to swap.

7.1.1. OS memory sizing

Some memory must be reserved for all activities on the server that are not Neo4j related. In addition, leave enough memory for the operating system file buffer cache to fit the contents of the `index` and `schema` directories, since it will impact index lookup performance if the indexes cannot fit in memory. 1G is a good starting point for when Neo4j is the only server running on that machine.

```
OS Memory = 1GB + (size of graph.db/index) + (size of graph.db/schema)
```

7.1.2. Page cache sizing

The page cache is used to cache the Neo4j data as stored on disk. Ensuring that all, or at least most, of the graph data from disk is cached into memory will help avoid costly disk access and result in optimal performance. You can determine the total memory needed for the page cache by summing up the sizes of the `NEO4J_HOME/data/databases/graph.db/*store.db*` files and adding 20% for growth.

The parameter for specifyig the page cache is: `dbms.memory.pagecache.size`. This specifies how much memory Neo4j is allowed to use for this cache.

If this is not explicitly defined on startup, Neo4j will look at how much available memory the machine

has, subtract the JVM max heap allocation from that, and then use 50% of what is left for the page cache. This is considered the default configuration.

The following are two possible methods for estimating the page cache size:

1. For an existing Neo4j database, sum up the size of all the `*store.db*` files in your store file directory, to figure out how big a page cache you need to fit all your data. Add another 20% for growth. For instance, on a posix system you can look at the total of running `$ du -hc *store.db*` in the *data/databases/graph.db* directory.
2. For a new Neo4j database, it is useful to run an import with a fraction (e.g. 1/100th) of the data and then multiply the resulting store-size by that fraction (x 100). Add another 20% for growth. For example: import 1/100th of the data and sum up the sizes of the resulting database files. Then multiply by 120 for a total estimate of the database size, including 20% for growth.

Parameter	Possible values	Effect
`dbms.memory.pagecache.size`	The maximum amount of memory to use for the page cache, either in bytes, or greater byte-like units, such as `100m` for 100 mega-bytes, or `4g` for 4 giga-bytes.	The amount of memory to use for mapping the store files, in a unit of bytes. This will automatically be rounded down to the nearest whole page. This value cannot be zero. For extremely small and memory constrained deployments, it is recommended to still reserve at least a couple of megabytes for the page cache.
`unsupported.dbms.report_configuration`	`true` or `false`	If set to `true` the current configuration settings will be written to the default system output, mostly the console or the logfiles.

7.1.3. Heap sizing

The size of the available heap memory is an important aspect for the performance of Neo4j.

Generally speaking, it is beneficial to configure a large enough heap space to sustain concurrent operations. For many setups, a heap size between 8G and 16G is large enough to run Neo4j reliably.

The heap memory size is determined by the parameters in NEO4J_HOME/conf/neo4j-wrapper.conf, namely `dbms.memory.heap.initial_size` and `dbms.memory.heap.max_size` providing the heap size in Megabytes, e.g. 16000. It is recommended to set these two parameters to the same value to avoid unwanted full garbage collection pauses.

7.1.4. Tuning of the garbage collector

The heap is separated into an *old generation* and a *young generation*. New objects are allocated in the young generation, and then later moved to the old generation, if they stay live (in use) for long enough. When a generation fills up, the garbage collector performs a collection, during which all other threads in the process are paused. The young generation is quick to collect since the pause time correlates with the *live set* of objects, and is independent of the size of the young generation. In the old generation, pause times roughly correlates with the size of the heap. For this reason, the heap should ideally be sized and tuned such that transaction and query state never makes it to the old generation.

The heap size is configured with the `dbms.memory.heap.max_size` (in MBs) setting in the *neo4j-wrapper.conf* file. The initial size of the heap is specified by the `dbms.memory.heap.initial_size` setting, or with the `-Xms???m` flag, or chosen heuristically by the JVM itself if left unspecified. The JVM will automatically grow the heap as needed, up to the maximum size. The growing of the heap requires a full garbage collection cycle. It is recommended to set the initial heap size and the maximum heap size to the same value. This way the pause that happens when the garbage collector grows the heap can be avoided.

The ratio of the size between the old generation and the new generation of the heap is controlled by the `-XX:NewRatio=N` flag. `N` is typically between 2 and 8 by default. A ratio of 2 means that the old generation size, divided by the new generation size, is equal to 2. In other words, two thirds of the heap memory will be dedicated to the old generation. A ratio of 3 will dedicate three quarters of the heap to the old generation, and a ratio of 1 will keep the two generations about the same size. A ratio of 1 is quite aggressive, but may be necessary if your transactions changes a lot of data. Having a large new generation can also be important if you run Cypher queries that need to keep a lot of data resident, for example when sorting big result sets.

If the new generation is too small, short-lived objects may be moved to the old generation too soon. This is called premature promotion and will slow the database down by increasing the frequency of old generation garbage collection cycles. If the new generation is too big, the garbage collector may decide that the old generation does not have enough space to fit all the objects it expects to promote from the new to the old generation. This turns new generation garbage collection cycles into old generation garbage collection cycles, again slowing the database down. Running more concurrent threads means that more allocations can take place in a given span of time, in turn increasing the pressure on the new generation in particular.

The *Compressed OOPs* feature in the JVM allows object references to be compressed to use only 32 bits. The feature saves a lot of memory, but is not enabled for heaps larger than 32 GB. Gains from increasing the heap size beyond 32 GB can therefore be small or even negative, unless the increase is significant (64 GB or above).

Neo4j has a number of long-lived objects, that stay around in the old generation, effectively for the lifetime of the Java process. To process them efficiently, and without adversely affecting the garbage collection pause time, we recommend using a concurrent garbage collector.

How to tune the specific garbage collection algorithm depends on both the JVM version and the workload. It is recommended to test the garbage collection settings under realistic load for days or weeks. Problems like heap fragmentation can take a long time to surface.

To gain good performance, these are the things to look into first:

- Make sure the JVM is not spending too much time performing garbage collection. The goal is to have a large enough heap to make sure that heavy/peak load will not result in so called GC-trashing. Performance can drop as much as two orders of magnitude when GC-trashing happens. Having too large heap may also hurt performance so you may have to try some different heap sizes.
- Use a concurrent garbage collector. We find that -XX:+UseG1GC works well in most use-cases.
  - The Neo4j JVM needs enough heap memory for the transaction state and query processing, plus some head-room for the garbage collector. Because the heap memory needs are so workload dependent, it is common to see configurations from 1 GB, up to 32 GBs of heap memory.
- Start the JVM with the -server flag and a good sized heap.
  - The operating system on a dedicated server can usually make do with 1 to 2 GBs of memory, but the more physical memory the machine has, the more memory the operating system will need.

Edit the following properties:

Table 14. neo4j-wrapper.conf JVM tuning properties

Property Name	Meaning
`dbms.memory.heap.initial_size`	initial heap size (in MB)
`dbms.memory.heap.max_size`	maximum heap size (in MB)

Property Name	Meaning
dbms.jvm.additional	additional literal JVM parameter

7.2. Transaction logs

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

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

By default, log switches happen when log sizes surpass 250 MB. This can be configured using the parameter dbms.tx_log.rotation.size.

There are several different means of controlling the amount of transaction logs that is kept, using the parameter dbms.tx_log.rotation.retention_policy. The format in which this is configured is:

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

For example:

```
# Will keep logical logs indefinitely
dbms.tx_log.rotation.retention_policy=true

# Will keep only the most recent non-empty log
dbms.tx_log.rotation.retention_policy=false

# Will keep logical logs which contains any transaction committed within 30 days
dbms.tx_log.rotation.retention_policy=30 days

# Will keep logical logs which contains any of the most recent 500 000 transactions
dbms.tx_log.rotation.retention_policy=500k txs
```

Full list:

Type	Description	Example
files	Number of most recent logical log files to keep	"10 files"
size	Max disk size to allow log files to occupy	"300M size" or "1G size"
txs	Number of latest transactions to keep Keep	"250k txs" or "5M txs"
hours	Keep logs which contains any transaction committed within N hours from current time	"10 hours"
days	Keep logs which contains any transaction committed within N days from current time	"50 days"

7.3. Compressed property value storage

This section explains Neo4j property value compression and disk usage.

Neo4j can in many cases compress and inline the storage of property values, such as short arrays and strings, with the purpose of saving disk space and possibly an I/O operation.

Compressed storage of short arrays

Neo4j will try to store your primitive arrays in a compressed way. To do that, it employs a "bit-shaving" algorithm that tries to reduce the number of bits required for storing the members of the array. In particular:

1. For each member of the array, it determines the position of leftmost set bit.
2. Determines the largest such position among all members of the array.
3. It reduces all members to that number of bits.
4. Stores those values, prefixed by a small header.

That means that when even a single negative value is included in the array then the original size of the primitives will be used.

There is a possibility that the result can be inlined in the property record if:

- It is less than 24 bytes after compression.
- It has less than 64 members.

For example, an array `long[] {0L, 1L, 2L, 4L}` will be inlined, as the largest entry (`4`) will require 3 bits to store so the whole array will be stored in 4 × 3 = *12 bits*. The array `long[] {-1L, 1L, 2L, 4L}` however will require the whole 64 bits for the `-1` entry so it needs 64 × 4 = *32 bytes* and it will end up in the dynamic store.

Compressed storage of short strings

Neo4j will try to classify your strings in a short string class and if it manages that it will treat it accordingly. In that case, it will be stored without indirection in the property store, inlining it instead in the property record, meaning that the dynamic string store will not be involved in storing that value, leading to reduced disk footprint. Additionally, when no string record is needed to store the property, it can be read and written in a single lookup, leading to performance improvements and less disk space required.

The various classes for short strings are:

- Numerical, consisting of digits 0..9 and the punctuation space, period, dash, plus, comma and apostrophe.
- Date, consisting of digits 0..9 and the punctuation space dash, colon, slash, plus and comma.
- Hex (lower case), consisting of digits 0..9 and lower case letters a..f
- Hex (upper case), consisting of digits 0..9 and upper case letters a..f
- Upper case, consisting of upper case letters A..Z, and the punctuation space, underscore, period, dash, colon and slash.
- Lower case, like upper but with lower case letters a..z instead of upper case
- E-mail, consisting of lower case letters a..z and the punctuation comma, underscore, period, dash, plus and the at sign (@).
- URI, consisting of lower case letters a..z, digits 0..9 and most punctuation available.
- Alpha-numerical, consisting of both upper and lower case letters a..zA..z, digits 0..9 and punctuation space and underscore.
- Alpha-symbolical, consisting of both upper and lower case letters a..zA..Z and the punctuation

space, underscore, period, dash, colon, slash, plus, comma, apostrophe, at sign, pipe and semicolon.

- European, consisting of most accented european characters and digits plus punctuation space, dash, underscore and period — like latin1 but with less punctuation.
- Latin 1.
- UTF-8.

In addition to the string's contents, the number of characters also determines if the string can be inlined or not. Each class has its own character count limits, which are

Table 15. Character count limits

String class	Character count limit
Numerical, Date and Hex	54
Uppercase, Lowercase and E-mail	43
URI, Alphanumerical and Alphasymbolical	36
European	31
Latin1	27
UTF-8	14

That means that the largest inline-able string is 54 characters long and must be of the Numerical class and also that all Strings of size 14 or less will always be inlined.

Also note that the above limits are for the default 41 byte PropertyRecord layout — if that parameter is changed via editing the source and recompiling, the above have to be recalculated.

7.4. Linux file system tuning

This section covers Neo4j I/O behavior and how to optimize for operations on disk.

Databases often produce many small and random reads when querying data, and few sequential writes when committing changes.

By default, most Linux distributions schedule IO requests using the Completely Fair Queuing (CFQ) algorithm, which provides a good balance between throughput and latency. The particular IO workload of a database, however, is better served by the Deadline scheduler. The Deadline scheduler gives preference to *read* requests, and processes them as soon as possible. This tends to decrease the latency of reads, while the latency of writes goes up. Since the writes are usually sequential, their lingering in the IO queue increases the change of overlapping or adjacent write requests being merged together. This effectively reduces the number of writes that are sent to the drive.

On Linux, the IO scheduler for a drive, in this case `sda`, can be changed at runtime like this:

```
$ echo 'deadline' > /sys/block/sda/queue/scheduler
$ cat              /sys/block/sda/queue/scheduler
noop [deadline] cfq
```

Another recommended practice is to disable file and directory access time updates. This way, the file system won't have to issue writes that update this meta-data, thus improving write performance. This can be accomplished by setting the `noatime,nodiratime` mount options in *fstab*, or when issuing the disk mount command.

7.5. Disks, RAM and other tips

This section provides an overview of performance considerations for disk and RAM when running Neo4j.

As with any persistence solution, performance depends a lot on the persistence media used. Better disks equals better 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. Today a typical mechanical drive has an average seek time of about 5ms. This can cause a query or traversal to be very slow when the amount of RAM assigned to the page cache is too small. A new, good SATA enabled SSD has an average seek time of less than 100 microseconds, meaning those scenarios will execute at least 50 times faster. However, this is still tens or hundreds of times slower than accessing RAM.

To avoid hitting disk you need more RAM. On a standard mechanical drive you can handle graphs with a few tens of millions of primitives (nodes, relationships and properties) with 2-3 GBs of RAM. A server with 8-16 GBs of RAM can handle graphs with hundreds of millions of primitives, and a good server with 16-64 GBs can handle billions of primitives. However, if you invest in a good SSD you will be able to handle much larger graphs on less RAM.

Use tools like `dstat` or `vmstat` to gather information when your application is running. If the swap or paging numbers are high, that is a sign that the Lucene indexes don't quite fit in memory. In this case, queries that do index lookups will have high latencies.

When Neo4j starts up, its page cache is empty and needs to warm up. 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.

Neo4j also flushes its page cache in the background, so it is not uncommon to see a steady trickle of blocks being written to the drive during steady-state. This background flushing only produces a small amount of IO wait, however. If the IO wait times are high during steady-state, it may be a sign that Neo4j is bottle-necked on the random IO performance of the drive. The best drives for running Neo4j are fast SSDs that can take lots of random IOPS.

Appendix A: Reference

This appendix contains a complete reference of Neo4j configuration settings. They can be set in neo4j.conf.

A.1. Configuration settings reference

Table 16. Settings used by the server configuration

Name	Description
browser.allow_outgoing_connections	Configure the policy for outgoing Neo4j Browser connections.
browser.credential_timeout	Configure the Neo4j Browser to time out logged in users after this idle period.
browser.remote_content_hostname_whitelist	Whitelist of hosts for the Neo4j Browser to be allowed to fetch content from.
browser.retain_connection_credentials	Configure the Neo4j Browser to store or not store user credentials.
cypher.default_language_version	Set this to specify the default parser (language version).
cypher.forbid_exhaustive_shortestpath	This setting is associated with performance optimization.
cypher.hints_error	Set this to specify the behavior when Cypher planner or runtime hints cannot be fulfilled.
cypher.min_replan_interval	The minimum lifetime of a query plan before a query is considered for replanning.
cypher.planner	Set this to specify the default planner for the default language version.
cypher.statistics_divergence_threshold	The threshold when a plan is considered stale.
dbms.active_database	Name of the database to load.
dbms.allow_format_migration	Whether to allow a store upgrade in case the current version of the database starts against an older store version.
dbms.backup.address	Listening server for online backups.
dbms.backup.enabled	Enable support for running online backups.
dbms.checkpoint.interval.time	Configures the time interval between check-points.
dbms.checkpoint.interval.tx	Configures the transaction interval between check-points.
dbms.checkpoint.iops.limit	Limit the number of IOs the background checkpoint process will consume per second.
dbms.directories.certificates	Directory for storing certificates to be used by Neo4j for TLS connections.
dbms.directories.data	Path of the data directory.
dbms.directories.import	Sets the root directory for file URLs used with the Cypher `LOAD CSV` clause.
dbms.directories.lib	Path of the lib directory.
dbms.directories.logs	Path of the logs directory.
dbms.directories.metrics	The target location of the CSV files: a path to a directory wherein a CSV file per reported field will be written.
dbms.directories.plugins	Location of the database plugin directory.
dbms.directories.run	Path of the run directory.

Name	Description
dbms.index_sampling.background_enabled	Enable or disable background index sampling.
dbms.index_sampling.sample_size_limit	Index sampling chunk size limit.
dbms.index_sampling.update_percentage	Percentage of index updates of total index size required before sampling of a given index is triggered.
dbms.index_searcher_cache_size	The maximum number of open Lucene index searchers.
dbms.logs.debug.level	Debug log level threshold.
dbms.logs.debug.rotation.delay	Minimum time interval after last rotation of the debug log before it may be rotated again.
dbms.logs.debug.rotation.keep_number	Maximum number of history files for the debug log.
dbms.logs.debug.rotation.size	Threshold for rotation of the debug log.
dbms.logs.gc.enabled	Enable GC Logging.
dbms.logs.gc.options	GC Logging Options.
dbms.logs.gc.rotation.keep_number	Number of GC logs to keep.
dbms.logs.gc.rotation.size	Size of each GC log that is kept.
dbms.logs.http.enabled	Enable HTTP request logging.
dbms.logs.http.rotation.keep_number	Number of HTTP logs to keep.
dbms.logs.http.rotation.size	Size of each HTTP log that is kept.
dbms.logs.query.enabled	Log executed queries that take longer than the configured threshold, dbms.logs.query.threshold.
dbms.logs.query.parameter_logging_enabled	Log parameters for executed queries that took longer than the configured threshold.
dbms.logs.query.rotation.keep_number	Maximum number of history files for the query log.
dbms.logs.query.rotation.size	The file size in bytes at which the query log will auto-rotate.
dbms.logs.query.threshold	If the execution of query takes more time than this threshold, the query is logged - provided query logging is enabled.
dbms.memory.pagecache.size	The amount of memory to use for mapping the store files, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g').
dbms.memory.pagecache.swapper	Specify which page swapper to use for doing paged IO.
dbms.mode	Configure the operating mode of the database — 'SINGLE' for stand-alone operation, 'HA' for operating as a member in a cluster or 'ARBITER' for an HA-only cluster member with no database.
dbms.query_cache_size	The number of Cypher query execution plans that are cached.
dbms.read_only	Only allow read operations from this Neo4j instance.
dbms.record_format	Database record format.
dbms.relationship_grouping_threshold	Relationship count threshold for considering a node to be dense.
dbms.security.allow_csv_import_from_file_urls	Determines if Cypher will allow using file URLs when loading data using `LOAD CSV`.
dbms.security.auth_enabled	Enable auth requirement to access Neo4j.
dbms.security.ha_status_auth_enabled	Require authorization for access to the HA status endpoints.
dbms.security.http_authorization_classes	Comma-seperated list of custom security rules for Neo4j to use.

Name	Description
dbms.shell.enabled	Enable a remote shell server which Neo4j Shell clients can log in to.
dbms.shell.host	Remote host for shell.
dbms.shell.port	The port the shell will listen on.
dbms.shell.read_only	Read only mode.
dbms.shell.rmi_name	The name of the shell.
dbms.threads.worker_count	Number of Neo4j worker threads, your OS might enforce a lower limit than the maximum value specified here.
dbms.transaction.timeout	The maximum time interval of a transaction within which it should be completed.
dbms.transaction_timeout	Timeout for idle transactions in the REST endpoint.
dbms.tx_log.rotation.retention_policy	Make Neo4j keep the logical transaction logs for being able to backup the database.
dbms.tx_log.rotation.size	Specifies at which file size the logical log will auto-rotate.
dbms.udc.enabled	Enable the UDC extension.
dbms.unmanaged_extension_classes	Comma-separated list of <classname>=<mount point> for unmanaged extensions.
ha.allow_init_cluster	Whether to allow this instance to create a cluster if unable to join.
ha.branched_data_copying_strategy	Strategy for how to order handling of branched data on slaves and copying of the store from the master.
ha.branched_data_policy	Policy for how to handle branched data.
ha.broadcast_timeout	Timeout for broadcasting values in cluster.
ha.configuration_timeout	Timeout for waiting for configuration from an existing cluster member during cluster join.
ha.data_chunk_size	Max size of the data chunks that flows between master and slaves in HA.
ha.default_timeout	Default timeout used for clustering timeouts.
ha.election_timeout	Timeout for waiting for other members to finish a role election.
ha.heartbeat_interval	How often heartbeat messages should be sent.
ha.heartbeat_timeout	How long to wait for heartbeats from other instances before marking them as suspects for failure.
ha.host.coordination	Host and port to bind the cluster management communication.
ha.host.data	Hostname and port to bind the HA server.
ha.initial_hosts	A comma-separated list of other members of the cluster to join.
ha.internal_role_switch_timeout	Timeout for waiting for internal conditions during state switch, like for transactions to complete, before switching to master or slave.
ha.join_timeout	Timeout for joining a cluster.
ha.learn_timeout	Timeout for learning values.
ha.leave_timeout	Timeout for waiting for cluster leave to finish.
ha.max_acceptors	Maximum number of servers to involve when agreeing to membership changes.
ha.max_channels_per_slave	Maximum number of connections a slave can have to the master.

Name	Description
ha.paxos_timeout	Default value for all Paxos timeouts.
ha.phase1_timeout	Timeout for Paxos phase 1.
ha.phase2_timeout	Timeout for Paxos phase 2.
ha.pull_batch_size	Size of batches of transactions applied on slaves when pulling from master.
ha.pull_interval	Interval of pulling updates from master.
ha.role_switch_timeout	Timeout for request threads waiting for instance to become master or slave.
ha.server_id	Id for a cluster instance.
ha.slave_lock_timeout	Timeout for taking remote (write) locks on slaves.
ha.slave_only	Whether this instance should only participate as slave in cluster.
ha.slave_read_timeout	How long a slave will wait for response from master before giving up.
ha.tx_push_factor	The amount of slaves the master will ask to replicate a committed transaction.
ha.tx_push_strategy	Push strategy of a transaction to a slave during commit.
metrics.bolt.messages.enabled	Enable reporting metrics about Bolt Protocol message processing.
metrics.csv.enabled	Set to `true` to enable exporting metrics to CSV files.
metrics.csv.interval	The reporting interval for the CSV files.
metrics.cypher.replanning.enabled	Enable reporting metrics about number of occurred replanning events.
metrics.enabled	The default enablement value for all the supported metrics.
metrics.graphite.enabled	Set to `true` to enable exporting metrics to Graphite.
metrics.graphite.interval	The reporting interval for Graphite.
metrics.graphite.server	The hostname or IP address of the Graphite server.
metrics.jvm.buffers.enabled	Enable reporting metrics about the buffer pools.
metrics.jvm.gc.enabled	Enable reporting metrics about the duration of garbage collections.
metrics.jvm.memory.enabled	Enable reporting metrics about the memory usage.
metrics.jvm.threads.enabled	Enable reporting metrics about the current number of threads running.
metrics.neo4j.checkpointing.enabled	Enable reporting metrics about Neo4j check pointing.
metrics.neo4j.cluster.enabled	Enable reporting metrics about HA cluster info.
metrics.neo4j.counts.enabled	Enable reporting metrics about approximately how many entities are in the database.
metrics.neo4j.enabled	The default enablement value for all Neo4j specific support metrics.
metrics.neo4j.logrotation.enabled	Enable reporting metrics about the Neo4j log rotation.
metrics.neo4j.network.enabled	Enable reporting metrics about the network usage.
metrics.neo4j.pagecache.enabled	Enable reporting metrics about the Neo4j page cache.
metrics.neo4j.server.enabled	Enable reporting metrics about Server threading info.
metrics.neo4j.tx.enabled	Enable reporting metrics about transactions.

Name	Description
metrics.prefix	A common prefix for the reported metrics field names.
tools.consistency_checker.check_graph	Perform checks between nodes, relationships, properties, types and tokens.
tools.consistency_checker.check_indexes	Perform checks on indexes.
tools.consistency_checker.check_label_scan_store	Perform checks on the label scan store.
tools.consistency_checker.check_property_owners	Perform optional additional checking on property ownership.

Table 17. Deprecated settings

Name	Description
dbms.index_sampling.buffer_size	Size of buffer used by index sampling.

Table 18. browser.allow_outgoing_connections

Description	Configure the policy for outgoing Neo4j Browser connections.
Valid values	browser.allow_outgoing_connections is a boolean
Default value	`true`

Table 19. browser.credential_timeout

Description	Configure the Neo4j Browser to time out logged in users after this idle period. Setting this to 0 indicates no limit.
Valid values	browser.credential_timeout is a duration (valid units are `ms`, `s`, `m`; default unit is `ms`)
Default value	`0`

Table 20. browser.remote_content_hostname_whitelist

Description	Whitelist of hosts for the Neo4j Browser to be allowed to fetch content from.
Valid values	browser.remote_content_hostname_whitelist is a string
Default value	`http://guides.neo4j.com,https://guides.neo4j.com,http://localhost,https://localhost`

Table 21. browser.retain_connection_credentials

Description	Configure the Neo4j Browser to store or not store user credentials.
Valid values	browser.retain_connection_credentials is a boolean
Default value	`true`

Table 22. cypher.default_language_version

Description	Set this to specify the default parser (language version).
Valid values	cypher.default_language_version is one of `2.3`, `3.0`, `default`
Default value	`default`

Table 23. cypher.forbid_exhaustive_shortestpath

Description	This setting is associated with performance optimization. Set this to `true` in situations where it is preferable to have any queries using the 'shortestPath' function terminate as soon as possible with no answer, rather than potentially running for a long time attempting to find an answer (even if there is no path to be found). For most queries, the 'shortestPath' algorithm will return the correct answer very quickly. However there are some cases where it is possible that the fast bidirectional breadth-first search algorithm will find no results even if they exist. This can happen when the predicates in the `WHERE` clause applied to 'shortestPath' cannot be applied to each step of the traversal, and can only be applied to the entire path. When the query planner detects these special cases, it will plan to perform an exhaustive depth-first search if the fast algorithm finds no paths. However, the exhaustive search may be orders of magnitude slower than the fast algorithm. If it is critical that queries terminate as soon as possible, it is recommended that this option be set to `true`, which means that Neo4j will never consider using the exhaustive search for shortestPath queries. However, please note that if no paths are found, an error will be thrown at run time, which will need to be handled by the application.
Valid values	cypher.forbid_exhaustive_shortestpath is a boolean
Default value	`false`

Table 24. cypher.hints_error

Description	Set this to specify the behavior when Cypher planner or runtime hints cannot be fulfilled. If true, then non-conformance will result in an error, otherwise only a warning is generated.
Valid values	cypher.hints_error is a boolean
Default value	`false`

Table 25. cypher.min_replan_interval

Description	The minimum lifetime of a query plan before a query is considered for replanning.
Valid values	cypher.min_replan_interval is a duration (valid units are `ms`, `s`, `m`; default unit is `ms`)
Default value	`10000`

Table 26. cypher.planner

Description	Set this to specify the default planner for the default language version.
Valid values	cypher.planner is one of `COST`, `RULE`, `default`
Default value	`default`

Table 27. cypher.statistics_divergence_threshold

Description	The threshold when a plan is considered stale. If any of the underlying statistics used to create the plan has changed more than this value, the plan is considered stale and will be replanned. A value of 0 means always replan, and 1 means never replan.
Valid values	cypher.statistics_divergence_threshold is a double which is minimum `0.0`, and is maximum `1.0`
Default value	`0.75`

Table 28. dbms.active_database

Description	Name of the database to load.
Valid values	dbms.active_database is a string
Default value	`graph.db`

Table 29. dbms.allow_format_migration

Description	Whether to allow a store upgrade in case the current version of the database starts against an older store version. Setting this to `true` does not guarantee successful upgrade, it just allows an upgrade to be performed.
Valid values	dbms.allow_format_migration is a boolean
Default value	`false`

Table 30. dbms.backup.address

Description	Listening server for online backups.
Valid values	dbms.backup.address is a hostname and port
Default value	`127.0.0.1:6362-6372`

Table 31. dbms.backup.enabled

Description	Enable support for running online backups.
Valid values	dbms.backup.enabled is a boolean
Default value	`true`

Table 32. dbms.checkpoint.interval.time

Description	Configures the time interval between check-points. The database will not check-point more often than this (unless check pointing is triggered by a different event), but might check-point less often than this interval, if performing a check-point takes longer time than the configured interval. A check-point is a point in the transaction logs, from which recovery would start from. Longer check-point intervals typically means that recovery will take longer to complete in case of a crash. On the other hand, a longer check-point interval can also reduce the I/O load that the database places on the system, as each check-point implies a flushing and forcing of all the store files.
Valid values	dbms.checkpoint.interval.time is a duration (valid units are `ms`, `s`, `m`; default unit is `ms`)
Default value	`300000`

Table 33. dbms.checkpoint.interval.tx

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

Table 34. dbms.checkpoint.iops.limit

Description	Limit the number of IOs the background checkpoint process will consume per second. This setting is advisory, is ignored in Neo4j Community Edition, and is followed to best effort in Enterprise Edition. An IO is in this case a 8 KiB (mostly sequential) write. Limiting the write IO in this way will leave more bandwidth in the IO subsystem to service random-read IOs, which is important for the response time of queries when the database cannot fit entirely in memory. The only drawback of this setting is that longer checkpoint times may lead to slightly longer recovery times in case of a database or system crash. A lower number means lower IO pressure, and consequently longer checkpoint times. The configuration can also be commented out to remove the limitation entirely, and let the checkpointer flush data as fast as the hardware will go. Set this to -1 to disable the IOPS limit.
Valid values	dbms.checkpoint.iops.limit is an integer
Default value	`1000`

Table 35. dbms.directories.certificates

Description	Directory for storing certificates to be used by Neo4j for TLS connections.
Valid values	A filesystem path; relative paths are resolved against the installation root, *<neo4j-home>*
Default value	`certificates`

Table 36. dbms.directories.data

Description	Path of the data directory. You must not configure more than one Neo4j installation to use the same data directory.
Valid values	A filesystem path; relative paths are resolved against the installation root, *<neo4j-home>*
Default value	`data`

Table 37. dbms.directories.import

Description	Sets the root directory for file URLs used with the Cypher `LOAD CSV` clause. This must be set to a single directory, restricting access to only those files within that directory and its subdirectories.
Valid values	A filesystem path; relative paths are resolved against the installation root, *<neo4j-home>*

Table 38. dbms.directories.lib

Description	Path of the lib directory.
Valid values	A filesystem path; relative paths are resolved against the installation root, *<neo4j-home>*
Default value	`lib`

Table 39. dbms.directories.logs

Description	Path of the logs directory.
Valid values	A filesystem path; relative paths are resolved against the installation root, *<neo4j-home>*
Default value	`logs`

Table 40. dbms.directories.metrics

Description	The target location of the CSV files: a path to a directory wherein a CSV file per reported field will be written.
Valid values	A filesystem path; relative paths are resolved against the installation root, *<neo4j-home>*
Default value	`metrics`

Table 41. dbms.directories.plugins

Description	Location of the database plugin directory. Compiled Java JAR files that contain database procedures will be loaded if they are placed in this directory.
Valid values	A filesystem path; relative paths are resolved against the installation root, *<neo4j-home>*
Default value	`plugins`

Table 42. dbms.directories.run

Description	Path of the run directory. This directory holds Neo4j's runtime state, such as a pidfile when it is running in the background. The pidfile is created when starting neo4j and removed when stopping it. It may be placed on an in-memory filesystem such as tmpfs.
Valid values	A filesystem path; relative paths are resolved against the installation root, *<neo4j-home>*
Default value	`run`

Table 43. dbms.index_sampling.background_enabled

Description	Enable or disable background index sampling.
Valid values	dbms.index_sampling.background_enabled is a boolean
Default value	`true`

Table 44. dbms.index_sampling.buffer_size

Description	Size of buffer used by index sampling. This configuration setting is no longer applicable as from Neo4j 3.0.3. Please use dbms.index_sampling.sample_size_limit instead.

Valid values	dbms.index_sampling.buffer_size is a byte size (valid multipliers are `k`, `m`, `g`, `K`, `M`, `G`) which is minimum `1048576`, and is maximum `2147483647`
Default value	`67108864`
Deprecated	The `dbms.index_sampling.buffer_size` configuration setting has been deprecated.

Table 45. dbms.index_sampling.sample_size_limit

Description	Index sampling chunk size limit.
Valid values	dbms.index_sampling.sample_size_limit is an integer which is minimum `1048576`, and is maximum `2147483647`
Default value	`8388608`

Table 46. dbms.index_sampling.update_percentage

Description	Percentage of index updates of total index size required before sampling of a given index is triggered.
Valid values	dbms.index_sampling.update_percentage is an integer which is minimum `0`
Default value	`5`

Table 47. dbms.index_searcher_cache_size

Description	The maximum number of open Lucene index searchers.
Valid values	dbms.index_searcher_cache_size is an integer which is minimum `1`
Default value	`2147483647`

Table 48. dbms.logs.debug.level

Description	Debug log level threshold.
Valid values	dbms.logs.debug.level is one of `DEBUG`, `INFO`, `WARN`, `ERROR`, `NONE`
Default value	`INFO`

Table 49. dbms.logs.debug.rotation.delay

Description	Minimum time interval after last rotation of the debug log before it may be rotated again.
Valid values	dbms.logs.debug.rotation.delay is a duration (valid units are `ms`, `s`, `m`; default unit is `ms`)
Default value	`300000`

Table 50. dbms.logs.debug.rotation.keep_number

Description	Maximum number of history files for the debug log.
Valid values	dbms.logs.debug.rotation.keep_number is an integer which is minimum `1`
Default value	`7`

Table 51. dbms.logs.debug.rotation.size

Description	Threshold for rotation of the debug log.
Valid values	dbms.logs.debug.rotation.size is a byte size (valid multipliers are `k`, `m`, `g`, `K`, `M`, `G`) which is minimum `0`, and is maximum `9223372036854775807`
Default value	`20971520`

Table 52. dbms.logs.gc.enabled

Description	Enable GC Logging.
Valid values	dbms.logs.gc.enabled is a boolean

Default value	`false`

Table 53. dbms.logs.gc.options

Description	GC Logging Options.
Valid values	dbms.logs.gc.options is a string
Default value	`-XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:+PrintGCApplicationStoppedTime -XX:+PrintPromotionFailure -XX:+PrintTenuringDistribution`

Table 54. dbms.logs.gc.rotation.keep_number

Description	Number of GC logs to keep.
Valid values	dbms.logs.gc.rotation.keep_number is an integer
Default value	`5`

Table 55. dbms.logs.gc.rotation.size

Description	Size of each GC log that is kept.
Valid values	dbms.logs.gc.rotation.size is a byte size (valid multipliers are `k`, `m`, `g`, `K`, `M`, `G`) which is minimum `0`, and is maximum `9223372036854775807`
Default value	`20971520`

Table 56. dbms.logs.http.enabled

Description	Enable HTTP request logging.
Valid values	dbms.logs.http.enabled is a boolean
Default value	`false`

Table 57. dbms.logs.http.rotation.keep_number

Description	Number of HTTP logs to keep.
Valid values	dbms.logs.http.rotation.keep_number is an integer
Default value	`5`

Table 58. dbms.logs.http.rotation.size

Description	Size of each HTTP log that is kept.
Valid values	dbms.logs.http.rotation.size is a byte size (valid multipliers are `k`, `m`, `g`, `K`, `M`, `G`) which is minimum `0`, and is maximum `9223372036854775807`
Default value	`20971520`

Table 59. dbms.logs.query.enabled

Description	Log executed queries that take longer than the configured threshold, dbms.logs.query.threshold. Log entries are written to the file *query.log* located in the Logs directory. For location of the Logs directory, see File locations. This feature is available in the Neo4j Enterprise Edition.
Valid values	dbms.logs.query.enabled is a boolean
Default value	`false`

Table 60. dbms.logs.query.parameter_logging_enabled

Description	Log parameters for executed queries that took longer than the configured threshold.
Valid values	dbms.logs.query.parameter_logging_enabled is a boolean
Default value	`true`

Table 61. dbms.logs.query.rotation.keep_number

Description	Maximum number of history files for the query log.
Valid values	dbms.logs.query.rotation.keep_number is an integer which is minimum `1`
Default value	`7`

Table 62. dbms.logs.query.rotation.size

Description	The file size in bytes at which the query log will auto-rotate. If set to zero then no rotation will occur. Accepts a binary suffix `k`, `m` or `g`.
Valid values	dbms.logs.query.rotation.size is a byte size (valid multipliers are `k`, `m`, `g`, `K`, `M`, `G`) which is minimum `0`, and is maximum `9223372036854775807`
Default value	`20971520`

Table 63. dbms.logs.query.threshold

Description	If the execution of query takes more time than this threshold, the query is logged - provided query logging is enabled. Defaults to 0 seconds, that is all queries are logged.
Valid values	dbms.logs.query.threshold is a duration (valid units are `ms`, `s`, `m`; default unit is `ms`)
Default value	`0`

Table 64. dbms.memory.pagecache.size

Description	The amount of memory to use for mapping the store files, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g'). If Neo4j is running on a dedicated server, then it is generally recommended to leave about 2-4 gigabytes for the operating system, give the JVM enough heap to hold all your transaction state and query context, and then leave the rest for the page cache. The default page cache memory assumes the machine is dedicated to running Neo4j, and is heuristically set to 50% of RAM minus the max Java heap size.
Valid values	dbms.memory.pagecache.size is a byte size (valid multipliers are `k`, `m`, `g`, `K`, `M`, `G`) which is minimum `245760`
Default value	`2147483648`

Table 65. dbms.memory.pagecache.swapper

Description	Specify which page swapper to use for doing paged IO. This is only used when integrating with proprietary storage technology.
Valid values	dbms.memory.pagecache.swapper is a string

Table 66. dbms.mode

Description	Configure the operating mode of the database — 'SINGLE' for stand-alone operation, 'HA' for operating as a member in a cluster or 'ARBITER' for an HA-only cluster member with no database.
Valid values	dbms.mode is a string
Default value	`SINGLE`

Table 67. dbms.query_cache_size

Description	The number of Cypher query execution plans that are cached.
Valid values	dbms.query_cache_size is an integer which is minimum `0`
Default value	`1000`

Table 68. dbms.read_only

Description	Only allow read operations from this Neo4j instance. This mode still requires write access to the directory for lock purposes.
Valid values	dbms.read_only is a boolean

Default value	`false`

Table 69. dbms.record_format

Description	Database record format. Enterprise edition only. Valid values: `standard`, `high_limit`. Default value: `standard`.
Valid values	dbms.record_format is a string
Default value	

Table 70. dbms.relationship_grouping_threshold

Description	Relationship count threshold for considering a node to be dense.
Valid values	dbms.relationship_grouping_threshold is an integer which is minimum `1`
Default value	`50`

Table 71. 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 is a boolean
Default value	`true`

Table 72. dbms.security.auth_enabled

Description	Enable auth requirement to access Neo4j.
Valid values	dbms.security.auth_enabled is a boolean
Default value	`false`

Table 73. dbms.security.ha_status_auth_enabled

Description	Require authorization for access to the HA status endpoints.
Valid values	dbms.security.ha_status_auth_enabled is a boolean
Default value	`true`

Table 74. dbms.security.http_authorization_classes

Description	Comma-seperated list of custom security rules for Neo4j to use.
Valid values	dbms.security.http_authorization_classes is a list separated by "," where items are a string
Default value	`[]`

Table 75. dbms.shell.enabled

Description	Enable a remote shell server which Neo4j Shell clients can log in to.
Valid values	dbms.shell.enabled is a boolean
Default value	`false`

Table 76. dbms.shell.host

Description	Remote host for shell. By default, the shell server listens only on the loopback interface, but you can specify the IP address of any network interface or use `0.0.0.0` for all interfaces.
Valid values	dbms.shell.host is a string which must be a valid name
Default value	`127.0.0.1`

Table 77. dbms.shell.port

Description	The port the shell will listen on.
Valid values	dbms.shell.port is an integer which must be a valid port number (is in the range `0` to `65535`)
Default value	`1337`

Table 78. dbms.shell.read_only

Description	Read only mode. Will only allow read operations.
Valid values	dbms.shell.read_only is a boolean
Default value	`false`

Table 79. dbms.shell.rmi_name

Description	The name of the shell.
Valid values	dbms.shell.rmi_name is a string which must be a valid name
Default value	`shell`

Table 80. dbms.threads.worker_count

Description	Number of Neo4j worker threads, your OS might enforce a lower limit than the maximum value specified here.
Valid values	dbms.threads.worker_count is an integer which is in the range `1` to `44738`
Default value	`2`

Table 81. dbms.transaction.timeout

Description	The maximum time interval of a transaction within which it should be completed.
Valid values	dbms.transaction.timeout is a duration (valid units are `ms`, `s`, `m`; default unit is `ms`)
Default value	`60000`

Table 82. dbms.transaction_timeout

Description	Timeout for idle transactions in the REST endpoint.
Valid values	dbms.transaction_timeout is a duration (valid units are `ms`, `s`, `m`; default unit is `ms`)
Default value	`60000`

Table 83. dbms.tx_log.rotation.retention_policy

Description	Make Neo4j keep the logical transaction logs for being able to backup the database. Can be used for specifying the threshold to prune logical logs after. For example "10 days" will prune logical logs that only contains transactions older than 10 days from the current time, or "100k txs" will keep the 100k latest transactions and prune any older transactions.
Valid values	dbms.tx_log.rotation.retention_policy is a string which must be `true`/`false` or of format '<number><optional unit> <type>' for example `100M size` for limiting logical log space on disk to 100Mb, or `200k txs` for limiting the number of transactions to keep to 200 000
Default value	`7 days`

Table 84. dbms.tx_log.rotation.size

Description	Specifies at which file size the logical log will auto-rotate. `0` means that no rotation will automatically occur based on file size.
Valid values	dbms.tx_log.rotation.size is a byte size (valid multipliers are `k`, `m`, `g`, `K`, `M`, `G`) which is minimum `1048576`
Default value	`262144000`

Table 85. dbms.udc.enabled

Description	Enable the UDC extension.
Valid values	dbms.udc.enabled is a boolean
Default value	`true`

Table 86. dbms.unmanaged_extension_classes

Description	Comma-separated list of <classname>=<mount point> for unmanaged extensions.
Valid values	dbms.unmanaged_extension_classes is a comma-seperated list of <classname>=<mount point> strings
Default value	`[]`

Table 87. ha.allow_init_cluster

Description	Whether to allow this instance to create a cluster if unable to join.
Valid values	ha.allow_init_cluster is a boolean
Default value	`true`

Table 88. ha.branched_data_copying_strategy

Description	Strategy for how to order handling of branched data on slaves and copying of the store from the master. The default is branch_then_copy, which, when combined with the keep_last or keep_none branch handling strategies results in less space used during the store copy.
Valid values	ha.branched_data_copying_strategy is one of `branch_then_copy`, `copy_then_branch`
Default value	`branch_then_copy`

Table 89. ha.branched_data_policy

Description	Policy for how to handle branched data.
Valid values	ha.branched_data_policy is one of `keep_all`, `keep_last`, `keep_none`
Default value	`keep_all`

Table 90. ha.broadcast_timeout

Description	Timeout for broadcasting values in cluster. Must consider end-to-end duration of Paxos algorithm. This value is the default value for the ha.join_timeout and ha.leave_timeout settings.
Valid values	ha.broadcast_timeout is a duration (valid units are `ms`, `s`, `m`; default unit is `ms`)
Default value	`30000`

Table 91. ha.configuration_timeout

Description	Timeout for waiting for configuration from an existing cluster member during cluster join.
Valid values	ha.configuration_timeout is a duration (valid units are `ms`, `s`, `m`; default unit is `ms`)
Default value	`1000`

Table 92. ha.data_chunk_size

Description	Max size of the data chunks that flows between master and slaves in HA. Bigger size may increase throughput, but may also be more sensitive to variations in bandwidth, whereas lower size increases tolerance for bandwidth variations.
Valid values	ha.data_chunk_size is a byte size (valid multipliers are `k`, `m`, `g`, `K`, `M`, `G`) which is minimum `1024`
Default value	`2097152`

Table 93. ha.default_timeout

Description	Default timeout used for clustering timeouts. Override specific timeout settings with proper values if necessary. This value is the default value for the ha.heartbeat_interval, ha.paxos_timeout and ha.learn_timeout settings.
Valid values	ha.default_timeout is a duration (valid units are `ms`, `s`, `m`; default unit is `ms`)
Default value	`5000`

Table 94. ha.election_timeout

Description	Timeout for waiting for other members to finish a role election. Defaults to ha.paxos_timeout.
Valid values	ha.election_timeout is a duration (valid units are `ms`, `s`, `m`; default unit is `ms`)
Default value	`5000`

Table 95. ha.heartbeat_interval

Description	How often heartbeat messages should be sent. Defaults to ha.default_timeout.
Valid values	ha.heartbeat_interval is a duration (valid units are `ms`, `s`, `m`; default unit is `ms`)
Default value	`5000`

Table 96. ha.heartbeat_timeout

Description	How long to wait for heartbeats from other instances before marking them as suspects for failure. This value reflects considerations of network latency, expected duration of garbage collection pauses and other factors that can delay message sending and processing. Larger values will result in more stable masters but also will result in longer waits before a failover in case of master failure. This value should not be set to less than twice the ha.heartbeat_interval value otherwise there is a high risk of frequent master switches and possibly branched data occurrence.
Valid values	ha.heartbeat_timeout is a duration (valid units are `ms`, `s`, `m`; default unit is `ms`)
Default value	`40000`

Table 97. ha.host.coordination

Description	Host and port to bind the cluster management communication.
Valid values	ha.host.coordination is a hostname and port
Default value	`0.0.0.0:5001-5099`

Table 98. ha.host.data

Description	Hostname and port to bind the HA server.
Valid values	ha.host.data is a hostname and port
Default value	`0.0.0.0:6001-6011`

Table 99. ha.initial_hosts

Description	A comma-separated list of other members of the cluster to join.
Valid values	ha.initial_hosts is a list separated by "," where items are a hostname and port
Mandatory	The `ha.initial_hosts` configuration setting is mandatory.

Table 100. ha.internal_role_switch_timeout

Description	Timeout for waiting for internal conditions during state switch, like for transactions to complete, before switching to master or slave.
Valid values	ha.internal_role_switch_timeout is a duration (valid units are `ms`, `s`, `m`; default unit is `ms`)
Default value	`10000`

Table 101. ha.join_timeout

Description	Timeout for joining a cluster. Defaults to ha.broadcast_timeout.
Valid values	ha.join_timeout is a duration (valid units are `ms`, `s`, `m`; default unit is `ms`)
Default value	`30000`

Table 102. ha.learn_timeout

Description	Timeout for learning values. Defaults to ha.default_timeout.
Valid values	ha.learn_timeout is a duration (valid units are `ms`, `s`, `m`; default unit is `ms`)
Default value	`5000`

Table 103. ha.leave_timeout

Description	Timeout for waiting for cluster leave to finish. Defaults to ha.broadcast_timeout.
Valid values	ha.leave_timeout is a duration (valid units are `ms`, `s`, `m`; default unit is `ms`)
Default value	`30000`

Table 104. ha.max_acceptors

Description	Maximum number of servers to involve when agreeing to membership changes. In very large clusters, the probability of half the cluster failing is low, but protecting against any arbitrary half failing is expensive. Therefore you may wish to set this parameter to a value less than the cluster size.
Valid values	ha.max_acceptors is an integer which is minimum `1`
Default value	`21`

Table 105. ha.max_channels_per_slave

Description	Maximum number of connections a slave can have to the master.
Valid values	ha.max_channels_per_slave is an integer which is minimum `1`
Default value	`20`

Table 106. ha.paxos_timeout

Description	Default value for all Paxos timeouts. This setting controls the default value for the ha.phase1_timeout, ha.phase2_timeout and ha.election_timeout settings. If it is not given a value it defaults to ha.default_timeout and will implicitly change if ha.default_timeout changes. This is an advanced parameter which should only be changed if specifically advised by Neo4j Professional Services.
Valid values	ha.paxos_timeout is a duration (valid units are `ms`, `s`, `m`; default unit is `ms`)
Default value	`5000`

Table 107. ha.phase1_timeout

Description	Timeout for Paxos phase 1. If it is not given a value it defaults to ha.paxos_timeout and will implicitly change if ha.paxos_timeout changes. This is an advanced parameter which should only be changed if specifically advised by Neo4j Professional Services.
Valid values	ha.phase1_timeout is a duration (valid units are `ms`, `s`, `m`; default unit is `ms`)
Default value	`5000`

Table 108. ha.phase2_timeout

Description	Timeout for Paxos phase 2. If it is not given a value it defaults to ha.paxos_timeout and will implicitly change if ha.paxos_timeout changes. This is an advanced parameter which should only be changed if specifically advised by Neo4j Professional Services.
Valid values	ha.phase2_timeout is a duration (valid units are `ms`, `s`, `m`; default unit is `ms`)
Default value	`5000`

Table 109. ha.pull_batch_size

Description	Size of batches of transactions applied on slaves when pulling from master.
Valid values	ha.pull_batch_size is an integer
Default value	`100`

Table 110. ha.pull_interval

Description	Interval of pulling updates from master.
Valid values	ha.pull_interval is a duration (valid units are `ms`, `s`, `m`; default unit is `ms`)
Default value	`0`

Table 111. ha.role_switch_timeout

Description	Timeout for request threads waiting for instance to become master or slave.
Valid values	ha.role_switch_timeout is a duration (valid units are `ms`, `s`, `m`; default unit is `ms`)
Default value	`120000`

Table 112. ha.server_id

Description	Id for a cluster instance. Must be unique within the cluster.
Valid values	ha.server_id is an instance id, which has to be a valid integer
Mandatory	The `ha.server_id` configuration setting is mandatory.

Table 113. ha.slave_lock_timeout

Description	Timeout for taking remote (write) locks on slaves. Defaults to ha.slave_read_timeout.
Valid values	ha.slave_lock_timeout is a duration (valid units are `ms`, `s`, `m`; default unit is `ms`)
Default value	`20000`

Table 114. ha.slave_only

Description	Whether this instance should only participate as slave in cluster. If set to `true`, it will never be elected as master.
Valid values	ha.slave_only is a boolean
Default value	`false`

Table 115. ha.slave_read_timeout

Description	How long a slave will wait for response from master before giving up.
Valid values	ha.slave_read_timeout is a duration (valid units are `ms`, `s`, `m`; default unit is `ms`)
Default value	`20000`

Table 116. ha.tx_push_factor

Description	The amount of slaves the master will ask to replicate a committed transaction.
Valid values	ha.tx_push_factor is an integer which is minimum `0`
Default value	`1`

Table 117. ha.tx_push_strategy

Description	Push strategy of a transaction to a slave during commit.
Valid values	ha.tx_push_strategy is one of `round_robin`, `fixed_descending`, `fixed_ascending`
Default value	`fixed_ascending`

Table 118. metrics.bolt.messages.enabled

Description	Enable reporting metrics about Bolt Protocol message processing.
Valid values	metrics.bolt.messages.enabled is a boolean
Default value	`false`

Table 119. metrics.csv.enabled

Description	Set to `true` to enable exporting metrics to CSV files.
Valid values	metrics.csv.enabled is a boolean
Default value	`false`

Table 120. metrics.csv.interval

Description	The reporting interval for the CSV files. That is, how often new rows with numbers are appended to the CSV files.
Valid values	metrics.csv.interval is a duration (valid units are `ms`, `s`, `m`; default unit is `ms`)
Default value	`3000`

Table 121. metrics.cypher.replanning.enabled

Description	Enable reporting metrics about number of occurred replanning events.
Valid values	metrics.cypher.replanning.enabled is a boolean
Default value	`false`

Table 122. metrics.enabled

Description	The default enablement value for all the supported metrics. Set this to `false` to turn off all metrics by default. The individual settings can then be used to selectively re-enable specific metrics.
Valid values	metrics.enabled is a boolean
Default value	`false`

Table 123. metrics.graphite.enabled

Description	Set to `true` to enable exporting metrics to Graphite.
Valid values	metrics.graphite.enabled is a boolean
Default value	`false`

Table 124. metrics.graphite.interval

Description	The reporting interval for Graphite. That is, how often to send updated metrics to Graphite.
Valid values	metrics.graphite.interval is a duration (valid units are `ms`, `s`, `m`; default unit is `ms`)
Default value	`3000`

Table 125. metrics.graphite.server

Description	The hostname or IP address of the Graphite server.
Valid values	metrics.graphite.server is a hostname and port
Default value	`:2003`

Table 126. metrics.jvm.buffers.enabled

Description	Enable reporting metrics about the buffer pools.

Valid values	metrics.jvm.buffers.enabled is a boolean
Default value	`false`

Table 127. metrics.jvm.gc.enabled

Description	Enable reporting metrics about the duration of garbage collections.
Valid values	metrics.jvm.gc.enabled is a boolean
Default value	`false`

Table 128. metrics.jvm.memory.enabled

Description	Enable reporting metrics about the memory usage.
Valid values	metrics.jvm.memory.enabled is a boolean
Default value	`false`

Table 129. metrics.jvm.threads.enabled

Description	Enable reporting metrics about the current number of threads running.
Valid values	metrics.jvm.threads.enabled is a boolean
Default value	`false`

Table 130. metrics.neo4j.checkpointing.enabled

Description	Enable reporting metrics about Neo4j check pointing; when it occurs and how much time it takes to complete.
Valid values	metrics.neo4j.checkpointing.enabled is a boolean
Default value	`false`

Table 131. metrics.neo4j.cluster.enabled

Description	Enable reporting metrics about HA cluster info.
Valid values	metrics.neo4j.cluster.enabled is a boolean
Default value	`false`

Table 132. metrics.neo4j.counts.enabled

Description	Enable reporting metrics about approximately how many entities are in the database; nodes, relationships, properties, etc.
Valid values	metrics.neo4j.counts.enabled is a boolean
Default value	`false`

Table 133. metrics.neo4j.enabled

Description	The default enablement value for all Neo4j specific support metrics. Set this to `false` to turn off all Neo4j specific metrics by default. The individual `metrics.neo4j.*` metrics can then be turned on selectively.
Valid values	metrics.neo4j.enabled is a boolean
Default value	`false`

Table 134. metrics.neo4j.logrotation.enabled

Description	Enable reporting metrics about the Neo4j log rotation; when it occurs and how much time it takes to complete.
Valid values	metrics.neo4j.logrotation.enabled is a boolean

Default value	`false`

Table 135. metrics.neo4j.network.enabled

Description	Enable reporting metrics about the network usage.
Valid values	metrics.neo4j.network.enabled is a boolean
Default value	`false`

Table 136. metrics.neo4j.pagecache.enabled

Description	Enable reporting metrics about the Neo4j page cache; page faults, evictions, flushes, exceptions, etc.
Valid values	metrics.neo4j.pagecache.enabled is a boolean
Default value	`false`

Table 137. metrics.neo4j.server.enabled

Description	Enable reporting metrics about Server threading info.
Valid values	metrics.neo4j.server.enabled is a boolean
Default value	`false`

Table 138. metrics.neo4j.tx.enabled

Description	Enable reporting metrics about transactions; number of transactions started, committed, etc.
Valid values	metrics.neo4j.tx.enabled is a boolean
Default value	`false`

Table 139. metrics.prefix

Description	A common prefix for the reported metrics field names. By default, this is either be 'neo4j', or a computed value based on the cluster and instance names, when running in an HA configuration.
Valid values	metrics.prefix is a string
Default value	`neo4j`

Table 140. tools.consistency_checker.check_graph

Description	Perform checks between nodes, relationships, properties, types and tokens.
Valid values	tools.consistency_checker.check_graph is a boolean
Default value	`true`

Table 141. tools.consistency_checker.check_indexes

Description	Perform checks on indexes. Checking indexes is more expensive than checking the native stores, so it may be useful to turn off this check for very large databases.
Valid values	tools.consistency_checker.check_indexes is a boolean
Default value	`true`

Table 142. tools.consistency_checker.check_label_scan_store

Description	Perform checks on the label scan store. Checking this store is more expensive than checking the native stores, so it may be useful to turn off this check for very large databases.
Valid values	tools.consistency_checker.check_label_scan_store is a boolean
Default value	`true`

Table 143. tools.consistency_checker.check_property_owners

Description	Perform optional additional checking on property ownership. This can detect a theoretical inconsistency where a property could be owned by multiple entities. However, the check is very expensive in time and memory, so it is skipped by default.
Valid values	tools.consistency_checker.check_property_owners is a boolean
Default value	`false`

Appendix B: Tutorial

This chapter contains tutorials for deploying and operating Neo4j.

B.1. Set up a Neo4j cluster

This guide will give step-by-step instructions for setting up a basic cluster of three separate machines. For a description of the clustering architecture and related design considerations, refer to Introduction.

B.1.1. Download and configure

- Download Neo4j Enterprise Edition from the Neo4j download site *(http://neo4j.com/download/)*, and unpack on three separate machines.
- Configure the HA related settings for each installation as outlined below. Note that all three installations have the same configuration except for the ha.server_id property.

Neo4j instance #1 — neo4j-01.local

conf/neo4j.conf

```
# Unique server id for this Neo4j instance
# can not be negative id and must be unique
ha.server_id = 1

# List of other known instances in this cluster
ha.initial_hosts = neo4j-01.local:5001,neo4j-02.local:5001,neo4j-03.local:5001
# Alternatively, use IP addresses:
#ha.initial_hosts = 192.168.0.20:5001,192.168.0.21:5001,192.168.0.22:5001

# HA - High Availability
# SINGLE - Single mode, default.
dbms.mode=HA

dbms.connector.http.type=HTTP
dbms.connector.http.enabled=true
dbms.connector.http.address=0.0.0.0:7474
```

Neo4j instance #2 — neo4j-02.local

conf/neo4j.conf

```
# Unique server id for this Neo4j instance
# can not be negative id and must be unique
ha.server_id = 2

# List of other known instances in this cluster
ha.initial_hosts = neo4j-01.local:5001,neo4j-02.local:5001,neo4j-03.local:5001
# Alternatively, use IP addresses:
#ha.initial_hosts = 192.168.0.20:5001,192.168.0.21:5001,192.168.0.22:5001

# HA - High Availability
# SINGLE - Single mode, default.
dbms.mode=HA

dbms.connector.http.type=HTTP
dbms.connector.http.enabled=true
dbms.connector.http.address=0.0.0.0:7474
```

Neo4j instance #3 — neo4j-03.local

conf/neo4j.conf

```
# Unique server id for this Neo4j instance
# can not be negative id and must be unique
ha.server_id = 3

# List of other known instances in this cluster
ha.initial_hosts = neo4j-01.local:5001,neo4j-02.local:5001,neo4j-03.local:5001
# Alternatively, use IP addresses:
#ha.initial_hosts = 192.168.0.20:5001,192.168.0.21:5001,192.168.0.22:5001

# HA - High Availability
# SINGLE - Single mode, default.
dbms.mode=HA

dbms.connector.http.type=HTTP
dbms.connector.http.enabled=true
dbms.connector.http.address=0.0.0.0:7474
```

B.1.2. Start the Neo4j Servers

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

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

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

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

Startup Time

When running in HA mode, the startup script returns immediately instead of waiting for the server to become available. The database will be unavailable until all members listed in `ha.initial_hosts` are online and communicating with each other. In the example above this happens when you have started all three instances. To keep track of the startup state you can follow the messages in *neo4j.log* — the path is printed before the startup script returns.

Now, you should be able to access the three servers and check their HA status. Open the locations below in a web browser and issue the following command in the editor after having set a password for the database: `:play sysinfo`

- http://neo4j-01.local:7474/
- http://neo4j-02.local:7474/
- http://neo4j-03.local:7474/

You can replace database #3 with an 'arbiter' instance, see Arbiter instances.

That's it! You now have a Neo4j cluster of three instances running. You can start by making a change on any instance and those changes will be propagated between them. For more cluster related configuration options take a look at Setup and configuration.

B.2. Set up a local cluster

If you want to start a cluster similar to the one described above, but for development and testing purposes, it is convenient to run all Neo4j instances on the same machine. This is easy to achieve, although it requires some additional configuration as the defaults will conflict with each other.

Furthermore, the default `dbms.memory.pagecache.size` assumes that Neo4j has the machine to itself. If we in this example assume that the machine has 4 gigabytes of memory, and that each JVM consumes 500 megabytes of memory, then we can allocate 500 megabytes of memory to the page cache of each server.

B.2.1. Download and configure

1. Download Neo4j Enterprise Edition from the Neo4j download site *(http://neo4j.com/download/)*, and unpack into three separate directories on your test machine.
2. Configure the HA related settings for each installation as outlined below.

Neo4j instance #1 — ~/neo4j-01

conf/neo4j.conf

```
# Reduce the default page cache memory allocation
dbms.memory.pagecache.size=500m

# Port to listen to for incoming backup requests.
dbms.backup.address=127.0.0.1:6366

# Unique server id for this Neo4j instance
# can not be negative id and must be unique
ha.server_id=1

# List of other known instances in this cluster
ha.initial_hosts=127.0.0.1:5001,127.0.0.1:5002,127.0.0.1:5003

# IP and port for this instance to bind to for communicating cluster information
# with the other neo4j instances in the cluster.
ha.host.coordination=127.0.0.1:5001

# IP and port for this instance to bind to for communicating data with the
# other neo4j instances in the cluster.
ha.host.data=127.0.0.1:6363

# HA - High Availability
# SINGLE - Single mode, default.
dbms.mode=HA

# HTTP Connector
dbms.connector.http.type=HTTP
dbms.connector.http.enabled=true
dbms.connector.http.address=0.0.0.0:7474

# HTTPS Connector
dbms.connector.https.type=HTTP
dbms.connector.https.enabled=true
dbms.connector.https.encryption=TLS
dbms.connector.https.address=localhost:7471

# Bolt connector
dbms.connector.bolt.type=BOLT
dbms.connector.bolt.enabled=true
dbms.connector.bolt.tls_level=OPTIONAL
dbms.connector.bolt.address=0.0.0.0:7687
```

Neo4j instance #2 — ~/neo4j-02

conf/neo4j.conf

```
# Reduce the default page cache memory allocation
dbms.memory.pagecache.size=500m

# Port to listen to for incoming backup requests.
dbms.backup.address=127.0.0.1:6367

# Unique server id for this Neo4j instance
# can not be negative id and must be unique
ha.server_id=2

# List of other known instances in this cluster
ha.initial_hosts=127.0.0.1:5001,127.0.0.1:5002,127.0.0.1:5003

# IP and port for this instance to bind to for communicating cluster information
# with the other neo4j instances in the cluster.
ha.host.coordination=127.0.0.1:5002

# IP and port for this instance to bind to for communicating data with the
# other neo4j instances in the cluster.
ha.host.data=127.0.0.1:6364

# HA - High Availability
# SINGLE - Single mode, default.
dbms.mode=HA

# HTTP Connector
dbms.connector.http.type=HTTP
dbms.connector.http.enabled=true
dbms.connector.http.address=0.0.0.0:7475

# HTTPS Connector
dbms.connector.https.type=HTTP
dbms.connector.https.enabled=true
dbms.connector.https.encryption=TLS
dbms.connector.https.address=localhost:7472

# Bolt connector
dbms.connector.bolt.type=BOLT
dbms.connector.bolt.enabled=true
dbms.connector.bolt.tls_level=OPTIONAL
dbms.connector.bolt.address=0.0.0.0:7688
```

Neo4j instance #3 — ~/neo4j-03

conf/neo4j.conf

```
# Reduce the default page cache memory allocation
dbms.memory.pagecache.size=500m

# Port to listen to for incoming backup requests.
dbms.backup.address=127.0.0.1:6368

# Unique server id for this Neo4j instance
# can not be negative id and must be unique
ha.server_id=3

# List of other known instances in this cluster
ha.initial_hosts=127.0.0.1:5001,127.0.0.1:5002,127.0.0.1:5003

# IP and port for this instance to bind to for communicating cluster information
# with the other neo4j instances in the cluster.
ha.host.coordination=127.0.0.1:5003

# IP and port for this instance to bind to for communicating data with the
# other neo4j instances in the cluster.
ha.host.data=127.0.0.1:6365

# HA - High Availability
# SINGLE - Single mode, default.
dbms.mode=HA

# HTTP Connector
dbms.connector.http.type=HTTP
dbms.connector.http.enabled=true
dbms.connector.http.address=0.0.0.0:7476

# HTTPS Connector
dbms.connector.https.type=HTTP
dbms.connector.https.enabled=true
dbms.connector.https.encryption=TLS
dbms.connector.https.address=localhost:7473

# Bolt connector
dbms.connector.bolt.type=BOLT
dbms.connector.bolt.enabled=true
dbms.connector.bolt.tls_level=OPTIONAL
dbms.connector.bolt.address=0.0.0.0:7689
```

Start the Neo4j Servers

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

```
localhost:~/neo4j-01$ ./bin/neo4j start
```

```
localhost:~/neo4j-02$ ./bin/neo4j start
```

```
localhost:~/neo4j-03$ ./bin/neo4j start
```

Now, you should be able to access the three servers and check their HA status. Open the locations below in a web browser and issue the following command in the editor after having set a password for the database: `:play sysinfo`

- http://127.0.0.1:7474/
- http://127.0.0.1:7475/
- http://127.0.0.1:7476/

B.3. Use the Import tool

This tutorial provides detailed examples of using the Neo4j import tool.

This tutorial walks us through a series of examples to illustrate the capabilities of the Import tool.

When using CSV files for loading a database, each node must have a unique identifier, a *node identifier*, in order to be able to create relationships between nodes in the same process. Relationships are created by connecting the node identifiers. In the examples below, the node identifiers are stored as properties on the nodes. Node identifiers may be of interest later for cross-reference to other systems, traceability etc., but they are not mandatory. If you do not want the identifiers to persist after a completed import, then do not specify a property name in the `:ID` field.

It is possible to import only nodes using the import tool For doing so simply omit a relationships file when calling `neo4j-import`. Any relationships between the imported nodes will have to be created later by another method, since the import tool works for initial graph population only.

For this tutorial we will use a data set containing movies, actors and roles. If running the examples, exchange `path_to_target_directory` with the path to the database file directory. In a default installation, the `path_to_target_directory` is: *<neo4j-home>/data/databases/graph.db*. Note that if you wish to run one example after another you have to remove the database files in between.

B.3.1. Basic example

First we will look at the movies. Each movie has an id, which is used for referring to it from other data sources Moreover, each movie has a title and a year. Along with these properties we also add the node labels `Movie` and `Sequel`.

movies.csv

```
movieId:ID,title,year:int,:LABEL
tt0133093,"The Matrix",1999,Movie
tt0234215,"The Matrix Reloaded",2003,Movie;Sequel
tt0242653,"The Matrix Revolutions",2003,Movie;Sequel
```

Next up are the actors. They have an id - in this case a shorthand of their name - and a name. All the actors have the node label `Actor`.

actors.csv

```
personId:ID,name,:LABEL
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
```

Finally we have the roles that an actor plays in a movie, which will be represented by relationships in the database. In order to create a relationship between nodes we use the ids defined in `actors.csv` and `movies.csv` for the `START_ID` and `END_ID` fields. We also need to provide a relationship type (in this case `ACTED_IN`) for the `:TYPE` field.

roles.csv

```
:START_ID,role,:END_ID,:TYPE
keanu,"Neo",tt0133093,ACTED_IN
keanu,"Neo",tt0234215,ACTED_IN
keanu,"Neo",tt0242653,ACTED_IN
laurence,"Morpheus",tt0133093,ACTED_IN
laurence,"Morpheus",tt0234215,ACTED_IN
laurence,"Morpheus",tt0242653,ACTED_IN
carrieanne,"Trinity",tt0133093,ACTED_IN
carrieanne,"Trinity",tt0234215,ACTED_IN
carrieanne,"Trinity",tt0242653,ACTED_IN
```

The call to `neo4j-import` would look like this:

```
neo4j_home$ ./bin/neo4j-import --into path_to_target_directory --nodes movies.csv --nodes actors.csv
--relationships roles.csv
```

Now start up a database from the target directory:

```
neo4j_home$ ./bin/neo4j start
```

B.3.2. Customizing configuration options

We can customize the configuration options that the import tool uses (see Options) if our data does not fit the default format. The following CSV files are delimited by `;`, use `|` as the array delimiter and use `'` for quotes.

movies2.csv

```
movieId:ID;title;year:int;:LABEL
tt0133093;'The Matrix';1999;Movie
tt0234215;'The Matrix Reloaded';2003;Movie|Sequel
tt0242653;'The Matrix Revolutions';2003;Movie|Sequel
```

actors2.csv

```
personId:ID;name;:LABEL
keanu;'Keanu Reeves';Actor
laurence;'Laurence Fishburne';Actor
carrieanne;'Carrie-Anne Moss';Actor
```

roles2.csv

```
:START_ID;role;:END_ID;:TYPE
keanu;'Neo';tt0133093;ACTED_IN
keanu;'Neo';tt0234215;ACTED_IN
keanu;'Neo';tt0242653;ACTED_IN
laurence;'Morpheus';tt0133093;ACTED_IN
laurence;'Morpheus';tt0234215;ACTED_IN
laurence;'Morpheus';tt0242653;ACTED_IN
carrieanne;'Trinity';tt0133093;ACTED_IN
carrieanne;'Trinity';tt0234215;ACTED_IN
carrieanne;'Trinity';tt0242653;ACTED_IN
```

The call to `neo4j-import` would look like this:

```
neo4j_home$ ./bin/neo4j-import --into path_to_target_directory --nodes movies2.csv --nodes actors2.csv
--relationships roles2.csv --delimiter ";" --array-delimiter "|" --quote "'"
```

B.3.3. Using separate header files

When dealing with very large CSV files it is more convenient to have the header in a separate file. This makes it easier to edit the header as you avoid having to open a huge data file just to change it.

The import tool can also process single file compressed archives, for example: . `--nodes nodes.csv.gz` . `--relationships rels.zip`

We will use the same data as in the previous example but put the headers in separate files.

movies3-header.csv

```
movieId:ID,title,year:int,:LABEL
```

movies3.csv

```
tt0133093,"The Matrix",1999,Movie
tt0234215,"The Matrix Reloaded",2003,Movie;Sequel
tt0242653,"The Matrix Revolutions",2003,Movie;Sequel
```

actors3-header.csv

```
personId:ID,name,:LABEL
```

actors3.csv

```
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
```

roles3-header.csv

```
:START_ID,role,:END_ID,:TYPE
```

roles3.csv

```
keanu,"Neo",tt0133093,ACTED_IN
keanu,"Neo",tt0234215,ACTED_IN
keanu,"Neo",tt0242653,ACTED_IN
laurence,"Morpheus",tt0133093,ACTED_IN
laurence,"Morpheus",tt0234215,ACTED_IN
laurence,"Morpheus",tt0242653,ACTED_IN
carrieanne,"Trinity",tt0133093,ACTED_IN
carrieanne,"Trinity",tt0234215,ACTED_IN
carrieanne,"Trinity",tt0242653,ACTED_IN
```

The call to `neo4j-import` would look as follows. Note how the file groups are enclosed in quotation marks in the command.

```
neo4j_home$ ./bin/neo4j-import --into path_to_target_directory --nodes "movies3-header.csv,movies3.csv"
--nodes "actors3-header.csv,actors3.csv" --relationships "roles3-header.csv,roles3.csv"
```

B.3.4. Multiple input files

In addition to using a separate header file you can also provide multiple nodes or relationships files. This may be useful for example for processing the output from a Hadoop pipeline. Files within such an input group can be specified with multiple match strings, delimited by `,`, where each match string can be either: the exact file name or a regular expression matching one or more files. Multiple matching

files will be sorted according to their characters and their natural number sort order for file names containing numbers.

movies4-header.csv

```
movieId:ID,title,year:int,:LABEL
```

movies4-part1.csv

```
tt0133093,"The Matrix",1999,Movie
tt0234215,"The Matrix Reloaded",2003,Movie;Sequel
```

movies4-part2.csv

```
tt0242653,"The Matrix Revolutions",2003,Movie;Sequel
```

actors4-header.csv

```
personId:ID,name,:LABEL
```

actors4-part1.csv

```
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
```

actors4-part2.csv

```
carrieanne,"Carrie-Anne Moss",Actor
```

roles4-header.csv

```
:START_ID,role,:END_ID,:TYPE
```

roles4-part1.csv

```
keanu,"Neo",tt0133093,ACTED_IN
keanu,"Neo",tt0234215,ACTED_IN
keanu,"Neo",tt0242653,ACTED_IN
laurence,"Morpheus",tt0133093,ACTED_IN
laurence,"Morpheus",tt0234215,ACTED_IN
```

roles4-part2.csv

```
laurence,"Morpheus",tt0242653,ACTED_IN
carrieanne,"Trinity",tt0133093,ACTED_IN
carrieanne,"Trinity",tt0234215,ACTED_IN
carrieanne,"Trinity",tt0242653,ACTED_IN
```

The call to `neo4j-import` would look like this:

```
neo4j_home$ ./bin/neo4j-import --into path_to_target_directory --nodes "movies4-header.csv,movies4-
part1.csv,movies4-part2.csv" --nodes "actors4-header.csv,actors4-part1.csv,actors4-part2.csv"
--relationships "roles4-header.csv,roles4-part1.csv,roles4-part2.csv"
```

B.3.5. Types and labels

Using the same label for every node

If you want to use the same node label(s) for every node in your nodes file you can do this by specifying the appropriate value as an option to `neo4j-import`. There is then no need to specify the `:LABEL` field in the node file if you pass it as a command line option. If you do then both the label provided in the file and the one provided on the command line will be added to the node.

In this example we put the label `Movie` on every node specified in `movies5a.csv`, and we put the labels `Movie` and `Sequel` on the nodes specified in `sequels5a.csv`.

movies5a.csv

```
movieId:ID,title,year:int
tt0133093,"The Matrix",1999
```

sequels5a.csv

```
movieId:ID,title,year:int
tt0234215,"The Matrix Reloaded",2003
tt0242653,"The Matrix Revolutions",2003
```

actors5a.csv

```
personId:ID,name
keanu,"Keanu Reeves"
laurence,"Laurence Fishburne"
carrieanne,"Carrie-Anne Moss"
```

roles5a.csv

```
:START_ID,role,:END_ID,:TYPE
keanu,"Neo",tt0133093,ACTED_IN
keanu,"Neo",tt0234215,ACTED_IN
keanu,"Neo",tt0242653,ACTED_IN
laurence,"Morpheus",tt0133093,ACTED_IN
laurence,"Morpheus",tt0234215,ACTED_IN
laurence,"Morpheus",tt0242653,ACTED_IN
carrieanne,"Trinity",tt0133093,ACTED_IN
carrieanne,"Trinity",tt0234215,ACTED_IN
carrieanne,"Trinity",tt0242653,ACTED_IN
```

The call to `neo4j-import` would look like this:

```
neo4j_home$ ./bin/neo4j-import --into path_to_target_directory --nodes:Movie movies5a.csv
--nodes:Movie:Sequel sequels5a.csv --nodes:Actor actors5a.csv --relationships 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-import`. If you provide a relationship type both on the command line and in the relationships file, the one in the file will be applied. In this example we put the relationship type `ACTED_IN` on every relationship specified in `roles5b.csv`:

movies5b.csv

```
movieId:ID,title,year:int,:LABEL
tt0133093,"The Matrix",1999,Movie
tt0234215,"The Matrix Reloaded",2003,Movie;Sequel
tt0242653,"The Matrix Revolutions",2003,Movie;Sequel
```

actors5b.csv

```
personId:ID,name,:LABEL
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
```

roles5b.csv

```
:START_ID,role,:END_ID
keanu,"Neo",tt0133093
keanu,"Neo",tt0234215
keanu,"Neo",tt0242653
laurence,"Morpheus",tt0133093
laurence,"Morpheus",tt0234215
laurence,"Morpheus",tt0242653
carrieanne,"Trinity",tt0133093
carrieanne,"Trinity",tt0234215
carrieanne,"Trinity",tt0242653
```

The call to `neo4j-import` would look like this:

```
neo4j_home$ ./bin/neo4j-import --into path_to_target_directory --nodes movies5b.csv --nodes actors5b.csv
--relationships:ACTED_IN roles5b.csv
```

B.3.6. Property types

The type for properties specified in nodes and relationships files is defined in the header row. (see CSV file header format)

The following example creates a small graph containing one actor and one movie connected by an `ACTED_IN` relationship. There is a `roles` property on the relationship which contains an array of the characters played by the actor in a movie.

movies6.csv

```
movieId:ID,title,year:int,:LABEL
tt0099892,"Joe Versus the Volcano",1990,Movie
```

actors6.csv

```
personId:ID,name,:LABEL
meg,"Meg Ryan",Actor
```

roles6.csv

```
:START_ID,roles:string[],:END_ID,:TYPE
meg,"DeDe;Angelica Graynamore;Patricia Graynamore",tt0099892,ACTED_IN
```

The call to `neo4j-import` would look like this:

```
neo4j_home$ ./bin/neo4j-import --into path_to_target_directory --nodes movies6.csv --nodes actors6.csv
--relationships roles6.csv
```

B.3.7. ID handling

Each node processed by `neo4j-import` must provide a unique id. This id is used to find the correct nodes when creating relationships.

Working with sequential or auto incrementing identifiers

The import tool makes the assumption that identifiers are unique across node files. This may not be the case for data sets which use sequential, auto incremented or otherwise colliding identifiers. Those data sets can define id spaces where identifiers are unique within their respective id space.

For example if movies and people both use sequential identifiers then we would define `Movie` and `Actor` id spaces.

movies7.csv

```
movieId:ID(Movie-ID),title,year:int,:LABEL
1,"The Matrix",1999,Movie
2,"The Matrix Reloaded",2003,Movie;Sequel
3,"The Matrix Revolutions",2003,Movie;Sequel
```

actors7.csv

```
personId:ID(Actor-ID),name,:LABEL
1,"Keanu Reeves",Actor
2,"Laurence Fishburne",Actor
3,"Carrie-Anne Moss",Actor
```

We also need to reference the appropriate id space in our relationships file so it knows which nodes to connect together:

roles7.csv

```
:START_ID(Actor-ID),role,:END_ID(Movie-ID)
1,"Neo",1
1,"Neo",2
1,"Neo",3
2,"Morpheus",1
2,"Morpheus",2
2,"Morpheus",3
3,"Trinity",1
3,"Trinity",2
3,"Trinity",3
```

The call to `neo4j-import` would look like this:

```
neo4j_home$ ./bin/neo4j-import --into path_to_target_directory --nodes movies7.csv --nodes actors7.csv
--relationships:ACTED_IN roles7.csv
```

B.3.8. Bad input data

The import tool has a threshold of how many bad entities (nodes or relationships) to tolerate and skip before failing the import. By default `1000` bad entities are tolerated. A bad tolerance of `0` will as an example fail the import on the first bad entity. For more information, see the `--bad-tolerance` option.

There are different types of bad input, which we will look into.

Relationships referring to missing nodes

Relationships that refer to missing node ids, either for `:START_ID` or `:END_ID` are considered bad relationships. Whether or not such relationships are skipped is controlled with `--skip-bad-relationships` flag which can have the values `true` or `false` or no value, which means `true`. Specifying `false` means that any bad relationship is considered an error and will fail the import. For more information, see the `--skip-bad-relationships` option.

In the following example there is a missing `emil` node referenced in the roles file.

movies8a.csv

```
movieId:ID,title,year:int,:LABEL
tt0133093,"The Matrix",1999,Movie
tt0234215,"The Matrix Reloaded",2003,Movie;Sequel
tt0242653,"The Matrix Revolutions",2003,Movie;Sequel
```

actors8a.csv

```
personId:ID,name,:LABEL
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
```

roles8a.csv

```
:START_ID,role,:END_ID,:TYPE
keanu,"Neo",tt0133093,ACTED_IN
keanu,"Neo",tt0234215,ACTED_IN
keanu,"Neo",tt0242653,ACTED_IN
laurence,"Morpheus",tt0133093,ACTED_IN
laurence,"Morpheus",tt0234215,ACTED_IN
laurence,"Morpheus",tt0242653,ACTED_IN
carrieanne,"Trinity",tt0133093,ACTED_IN
carrieanne,"Trinity",tt0234215,ACTED_IN
carrieanne,"Trinity",tt0242653,ACTED_IN
emil,"Emil",tt0133093,ACTED_IN
```

The call to `neo4j-import` would look like this:

```
neo4j_home$ ./bin/neo4j-import --into path_to_target_directory --nodes movies8a.csv --nodes actors8a.csv
--relationships roles8a.csv
```

Since there was only one bad relationship the import process will complete successfully and a `not-imported.bad` file will be created and populated with the bad relationships.

not-imported.bad

```
InputRelationship:
   source: roles8a.csv:11
   properties: [role, Emil]
   startNode: emil
   endNode: tt0133093
   type: ACTED_IN
 refering to missing node emil
```

Multiple nodes with same id within same id space

Nodes that specify `:ID` which has already been specified within the id space are considered bad nodes. Whether or not such nodes are skipped is controlled with `--skip-duplicate-nodes` flag which can have the values `true` or `false` or no value, which means `true`. Specifying `false` means that any duplicate node is considered an error and will fail the import. For more information, see the `--skip`

`-duplicate-nodes` option.

In the following example there is a node id that is specified twice within the same id space.

actors8b.csv

```
personId:ID,name,:LABEL
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
laurence,"Laurence Harvey",Actor
```

The call to `neo4j-import` would look like this:

```
neo4j_home$ ./bin/neo4j-import --into path_to_target_directory --nodes actors8b.csv --skip-duplicate-nodes
```

Since there was only one bad node the import process will complete successfully and a `not-imported.bad` file will be created and populated with the bad node.

not-imported.bad

```
Id 'laurence' is defined more than once in global id space, at least at actors8b.csv:3 and actors8b.csv:5
```

License

No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material.

See http://creativecommons.org/licenses/by-sa/3.0/ for further details. The full license text is available at http://creativecommons.org/licenses/by-sa/3.0/legalcode.