



The Neo4j Operations Manual v3.0

Table of Contents

1. Introduction	2
1.1. Neo4j editions	2
1.2. Neo4j for the enterprise	3
1.2.1. Architecture	3
1.2.2. Design considerations	4
Read scalability	5
High availability	6
Disaster recovery	7
Analytics	8
2. Deployment	9
2.1. System Requirements	9
2.2. File locations	10
2.2.1. Log Files	10
2.2.2. Configuration	11
2.2.3. Permissions	11
2.3. Single instance install	11
2.3.1. Linux installation	11
Linux Packages	11
Unix Console Application	12
Linux Service	12
2.3.2. OSX installation	12
Mac OSX Installer	12
Running Neo4j from the Terminal	12
OSX Service	12
2.3.3. Windows installation	12
Windows Installer	13
Windows Console Application	13
Windows service	13
Windows PowerShell module	13
2.3.4. Multiple server instances on one machine	15
2.4. Neo4j Cluster install	15
2.4.1. Setup and configuration	15
Important configuration settings	16
2.4.2. Arbiter instances	17
2.4.3. Endpoints for status information	18
Introduction	18
The endpoints	18
Examples	18
2.4.4. HAProxy for load balancing	19
Configuring HAProxy for the Bolt Protocol	20
Configuring HAProxy for the REST API	21
Optimizing for reads and writes	22
Cache-based sharding with HAProxy	22
2.5. Post-install tasks	23
2.5.1. Waiting for Neo4j to start	23
2.5.2. Setting the number of open files	23

2.5.3. Setup for remote debugging	24
2.5.4. Usage Data Collector	25
Technical Information	25
How to disable UDC	25
2.6. Upgrading	26
2.6.1. Single-instance upgrade	26
Supported upgrade paths	26
Upgrade instructions	26
2.6.2. Neo4j cluster upgrade	27
Back up the Neo4j database	27
Shut down the cluster	28
Upgrade the master	28
Upgrade the slaves	28
Restart the cluster	28
2.7. Import tool	28
2.7.1. CSV file header format	29
Nodes	29
Relationships	29
ID spaces	30
2.7.2. Command line usage	30
Linux	30
Windows	30
Options	30
Output and statistics	32
Verbose error information	32
2.7.3. Import tool examples	32
Basic example	32
Customizing configuration options	33
Using separate header files	34
Multiple input files	35
Types and labels	36
Property types	38
ID handling	38
Bad input data	39
3. Security	41
3.1. Securing Neo4j Server	41
3.1.1. Secure the port and remote client connection accepts	41
3.1.2. Server authentication and authorization	41
3.1.3. HTTPS support	41
3.1.4. Arbitrary code execution	42
3.1.5. Server authorization rules	42
3.1.6. Enforcing Server Authorization Rules	43
3.1.7. Using Wildcards to Target Security Rules	43
3.1.8. Using Complex Wildcards to Target Security Rules	44
3.1.9. Using a proxy	45
3.1.10. LOAD CSV	45
4. Backup	47
4.1. Introducing backup	47
4.2. Performing backups	47

4.2.1. Backup commands	47
4.2.2. Incremental backups	47
4.3. Restoring a backup	48
5. Monitoring	49
5.1. Adjusting remote JMX access to the Neo4j Server	49
5.2. How to connect to a Neo4j instance using JMX and JConsole	49
5.3. Reference of supported JMX MBeans	51
5.4. Metrics Reporting	51
5.4.1. Introducing Metrics	51
5.4.2. Graphite Configuration	52
5.4.3. Export to CSV Configuration	52
5.4.4. Configuration settings reference for metrics	52
5.4.5. Available metrics	52
6. Performance tuning	53
6.1. Modifying configuration settings	53
6.2. Cypher tuning	53
6.3. Memory tuning	53
6.3.1. OS memory sizing	53
6.3.2. Page cache sizing	53
6.3.3. Heap sizing	54
6.3.4. Tuning of the garbage collector	54
6.4. Transaction logs	56
6.5. Compressed property value storage	56
6.6. Linux file system tuning	58
6.7. Disks, RAM and other tips	58
7. Tutorials	60
7.1. Set up a Neo4j cluster	60
7.1.1. Important configuration settings	60
dbms.mode	60
ha.server_id	60
ha.host.coordination	60
ha.initial_hosts	60
ha.host.data	61
7.1.2. Download and configure	61
7.1.3. Start the Neo4j Servers	62
7.2. Set up a local cluster	63
7.2.1. Download and configure	63
Start the Neo4j Servers	65
8. Configuration	66
8.1. Configuration Settings Reference	66
8.1.1. Configuring Bolt Connectors	85
8.1.2. Configuring HTTP Connectors	86
8.2. JMX Beans	86
8.3. Available metrics	97
8.3.1. Java Virtual Machine Metrics	99
9. License	100

© 2016 Neo Technology

License: [Creative Commons 3.0](#)

Chapter 1. Introduction

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 tuning](#) — Instructions on how to go about performance tuning for Neo4j.
- [Tutorials](#) — Step-by-step instructions on various scenarios for setting up Neo4j.
- [Configuration Settings Reference](#) — Listings of all Neo4j configuration parameters.

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.

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	-
High-Performance Cache	X	-
Clustering	X	-
Hot Backups	X	-
Advanced Monitoring	X	-

1.2. Neo4j for the enterprise

This section covers the major features of Neo4j Enterprise Edition.

1.2.1. Architecture

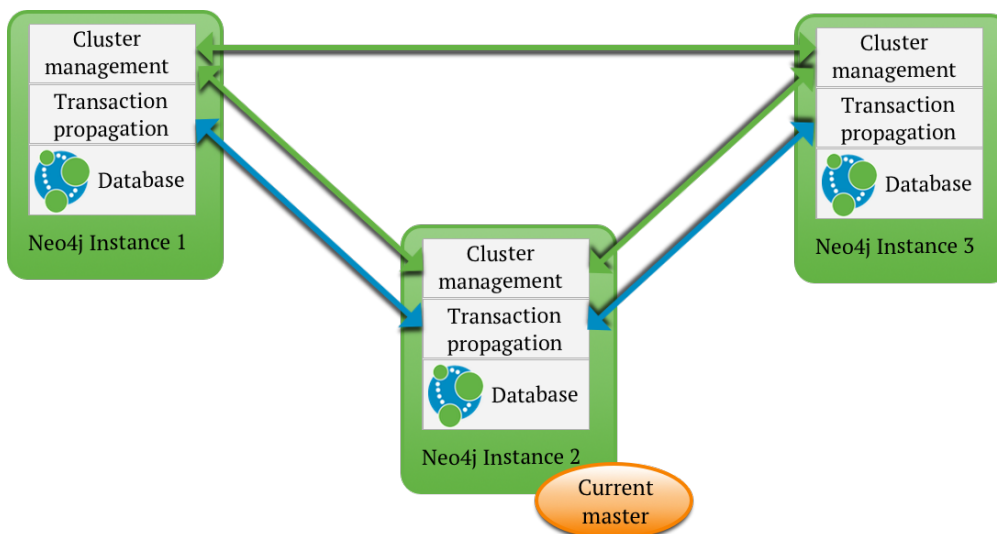


Figure 1. Neo4j cluster

Neo4j Clustering is comprised of a single master instance and zero or more slave instances. All

instances in the cluster have full copies of your data in their local database files. Each database instance contains the logic needed in order to coordinate with the other members of the cluster for data replication and election management.

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 has to be up to date with the master before performing a write operation. This is built into the communication protocol between the slave and master, so that updates will be applied to a slave communicating with its master automatically.

Write transactions performed directly through the master will execute in the same way as running in normal 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.

Whenever a Neo4j database becomes unavailable, by means of for example hardware failure or network outages, the other database instances in the cluster will detect that and mark it as temporarily failed. A database instance that becomes available after being unavailable 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 a quorum 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 just a few seconds and during this time no writes can take place

A special case of a slave instance is the arbiter instance. The arbiter instance does not operate any database, but can be regarded as cluster participants in that its 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 plus an arbiter instance, and still enjoy tolerance of a single failure of either of the three instances.

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.

1.2.2. Design considerations

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

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 slave 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 slave instance without much concern for data locality.

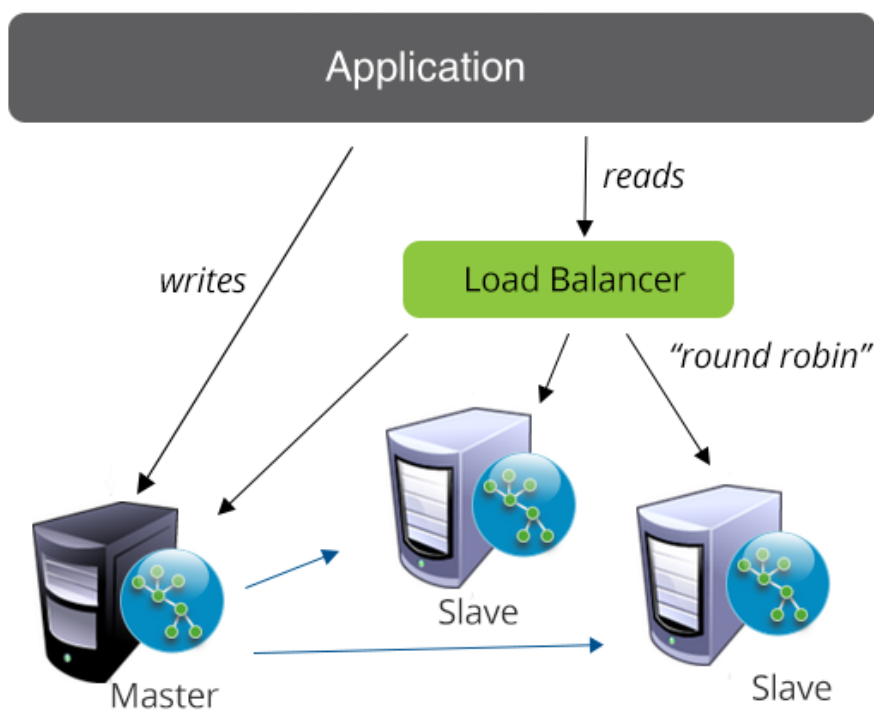


Figure 2. Distribute load balance reads to any slave instance in the cluster

When would you chose this method?

1. You need to scale up the number of concurrent read requests
2. Your data has no natural or obvious way of partitioning reads
3. 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-based partitioning". 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.

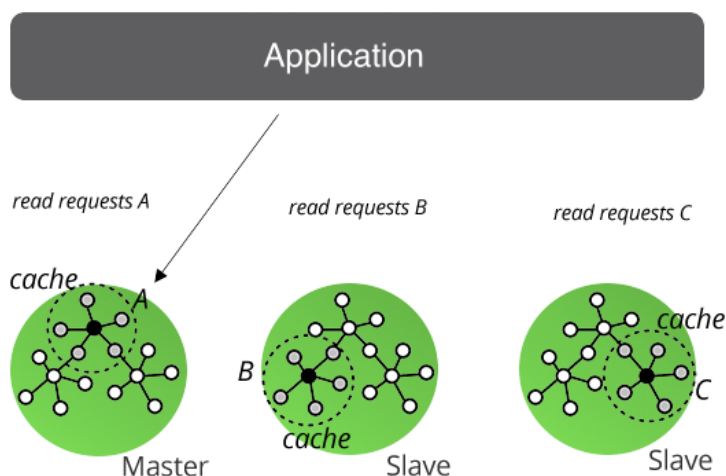


Figure 3. Cache-based partitioning

When would you choose this method?

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

High availability

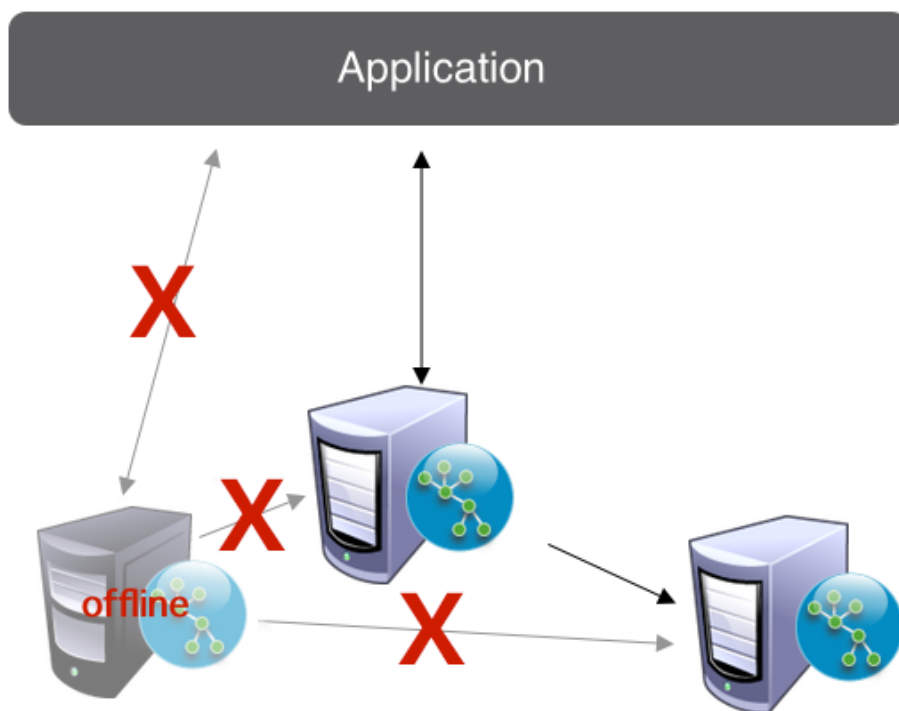


Figure 4. High availability cluster

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?

1. 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.
2. Global end-users with random access behavior are depending on the data stored in Neo4j.
3. Business continuity must be ensured by availability of disaster recovery features.

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.

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?

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

Chapter 2. Deployment

2.1. System Requirements

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 to operate. Community Edition installers for Windows and Mac include a JVM for convenience. Other distributions, including all distributions of Neo4j Enterprise Edition, require a pre-installed JVM.

Java

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

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

Operating Systems

Linux, HP-UX, Windows Server 2012 for production

Additionally, Windows XP and Mac OS X for development

Architectures

x86

OpenPOWER (POWER8)

2.2. File locations

This table shows where important files can be found by default in various Neo4j distribution packages.

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	/var/lib/neo4j/bin	/var/lib/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.

Filename	Description
<i>query.log</i>	Log of executed queries that takes longer than a specified threshold. (Enterprise 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
<code><neo4j-home></code>	parent of <code>bin</code>	<code>NEO4J_HOME</code>	Must be set explicitly if <code>bin</code> is not a subdirectory.
<code>bin</code>	directory where <code>neo4j</code> script is located	<code>NEO4J_BIN</code>	Must be set explicitly if <code>neo4j</code> script is invoked as a symlink.
<code>conf</code>	<code><neo4j-home>/conf</code>	<code>NEO4J_CONF</code>	Must be set explicitly if it is not a subdirectory of <code><neo4j-home></code> .

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`

Execute

- all files in `bin`

2.3. Single instance install

2.3.1. Linux installation

Linux Packages

- For Debian packages, see the instructions at <http://debian.neo4j.org/>.

After installation you may have to do some platform specific configuration and performance tuning. For that, refer to [Post-install 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. See the [neo4j man page](#) for further details.



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) (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) (<https://gist.github.com/chrisvest/7673244>) and [tutorials](http://www.linux.com/learn/tutorials/442412-managing-linux-daemons-with-init-scripts) (<http://www.linux.com/learn/tutorials/442412-managing-linux-daemons-with-init-scripts>).

2.3.2. OSx installation

Mac OSx 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.

Running Neo4j from the Terminal

The server can be started in the background from the terminal with the command `neo4j start`, and then stopped again with `neo4j stop`. The server can also be started in the foreground with `neo4j console` — then it's log output will be printed to the terminal.

OSX Service

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

2.3.3. Windows installation

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/) (<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) (<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. Multiple server instances on one machine

Neo4j can be configured to run as several instances on one machine. This might be done to run several databases for testing or development. This is not recommended for a production deployment.

For how to set this up, see [Set up a local cluster](#). Just use the Neo4j edition of your choice, follow the guide and remember to not set the servers to run in HA mode.

2.4. Neo4j Cluster install

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 install](#).

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

Each instance in a Neo4j HA cluster must be assigned an integer ID, which serves as its unique identifier. At startup, a Neo4j instance contacts the other instances specified in the `ha.initial_hosts` configuration option.

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. Neo4j will attempt to configure IP addresses for itself in the absence of explicit configuration.

`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 communications (like heartbeat messages). The default port is `5001`. In the absence of a specified IP address, Neo4j will attempt to find a valid interface for binding. While this behavior typically results in a well-behaved server, it is *strongly recommended* that users explicitly choose an IP address bound to the network interface of their choosing to ensure a coherent cluster deployment.

For example, `ha.host.coordination=192.168.33.22:5001` will listen for cluster communications on the network interface bound to the 192.168.33.0 subnet on port 5001.

`ha.initial_hosts`

`ha.initial_hosts` is a comma separated list of address/port pairs, which specify 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. Specifying an instance's own address is permitted. Do not use any whitespace in this configuration option.

For example, `ha.initial_hosts=192.168.33.22:5001,192.168.33.21:5001` will attempt to reach Neo4j instances listening on 192.168.33.22 on port 5001 and 192.168.33.21 on port 5001 on the 192.168.33.0 subnet.

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`. In the absence of a specified IP address, Neo4j will attempt to find a valid interface for binding. While this behavior typically results in a well-behaved server, it is *strongly recommended* that users explicitly choose an IP address bound to the network interface of their choosing to ensure a coherent cluster topology.

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

For example, `ha.host.data=192.168.33.22:6001` will listen for transactions from the cluster master on the network interface bound to the 192.168.33.0 subnet on port 6001.

Address and port formats

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

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

For `ha.host.coordination` the IP address *must be* the 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 address is omitted, then the port must be preceded with a colon (eg. `:5001`).

The syntax for setting the port range is: `<hostname>:<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 the [Configuration Settings Reference](#) section for a list of all available configuration settings.

2.4.2. Arbiter instances

A typical deployment of Neo4j will use a cluster of 3 machines to provide fault-tolerance and read scalability. This setup is described in [Set up a Neo4j cluster](#).

While having at least 3 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 2 Neo4j database instances and an additional arbiter instance and still enjoy tolerance of a single failure of either of the 3 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 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 REST endpoint to discover which instance is the master and direct write load to it. In this section, we'll deal with this REST 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 REST endpoint responses

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

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.

Let's first 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, let's 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 REST 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.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.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-install tasks

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's 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.properties` 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 attachment 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.

2.5.4. 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.

2.6. Upgrading

2.6.1. Single-instance upgrade

This section describes upgrading a *single* Neo4j instance. To upgrade a *Neo4j HA cluster (Neo4j Enterprise)*, 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 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.

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 → 3.0.1
- 2.1.latest → 3.0.1
- 2.2.latest → 3.0.1
- 2.3.latest → 3.0.1
- 3.0.any → 3.0.1

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, ensure that backups have completed successfully.
3. Install Neo4j 3.0.1.
4. Review the settings in the configuration files of the previous installation and transfer any custom settings to the 3.0.1 installation. Since many settings have been changed between Neo4j 2.x and 3.0.1, 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 3.0.1 installation. Neo4j will fail to start without this configuration.

8. Start up Neo4j 3.0.1.
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) (<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, ensure that backups have completed successfully.
3. Install Neo4j 3.0.1.
4. Review the settings in the configuration files of the previous installation and transfer any custom settings to the 3.0.1 installation.
5. When 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 3.0.1 installation. Neo4j will fail to start without this configuration.
8. Start up Neo4j 3.0.1.
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

Upgrading a Neo4j HA cluster to Neo4j 3.0.1 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 3.0.1 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 3.0.1 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 3.0.1.
3. Review the settings in the configuration files in the previous installation, and transfer any custom settings to the 3.0.1 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.

2.7. Import tool

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

This chapter explains how to use the tool, format the input data and concludes with an example bringing everything together.

These are some things you'll 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'll need to add indexes afterwards (see [\[graphdb-neo4j-schema-indexes\]](#)).



Data cannot be imported into an existing database using this tool.

- If you want to load small to medium sized CSV files see [\[query-load-csv\]](#).
- If you want to bulk import into an existing database see [\[batchinsert\]](#).

2.7.1. CSV file header format

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.

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`.

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.

ID spaces

The import tool assumes that node identifiers are unique across node files. If this isn't 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)**.

2.7.2. Command line usage

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.

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](#).

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, i.e. `""` 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. The default option is **STRING**. 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 entites 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.properties>

(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

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 isn't visible in this example, because it's 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).

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.

2.7.3. Import tool examples

Let's look at a few examples. We'll use a data set containing movies, actors and roles.



While you'll usually want to store your node identifier as a property on the node for looking it up later, it's not mandatory. If you don't want the identifier to be persisted then don't specify a property name in the `:ID` field.

Basic example

First we'll look at the movies. Each movie has an id, which is used to refer to it in other data sources, a title and a year Along with these properties we'll also add the node labels `Movie` and `Sequel`.

By default the import tool expects CSV files to be comma delimited.

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 - and a name and all have the Actor label.

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 refer to the ids used in *actors.csv* and *movies.csv* in the **START_ID** and **END_ID** fields. We also need to provide a relationship type (in this case **ACTS_IN**) in 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
```

With all data in place, we execute the following command:

```
neo4j-import --into path_to_target_directory --nodes movies.csv --nodes actors.csv --relationships roles.csv
```

We're now ready to start up a database from the target directory. (see [\[server-installation\]](#))

Once we've got the database up and running we can add appropriate indexes. (see [\[cypherdoc-labels-constraints-and-indexes\]](#).)



It is possible to import only nodes using the import tool - just don't specify a relationships file when calling `neo4j-import`. If you do this you'll need to create relationships later by another method - the import tool only works for initial graph population.

Customizing configuration options

We can customize the configuration options that the import tool uses (see [Options](#)) if our data doesn't fit the default format. The following CSV files are delimited by `;`, use `|` as their 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
```

We can then import these files with the following command line options:

```
neo4j-import --into path_to_target_directory --nodes movies2.csv --nodes actors2.csv --relationships
roles2.csv --delimiter ";" --array-delimiter "|" --quote ""
```

Using separate header files

When dealing with very large CSV files it's 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.



import-tool can also process single file compressed archives. e.g. `--nodes nodes.csv.gz` or `--relationships rels.zip`

We'll 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
```

Note how the file groups are enclosed in quotation marks in the command:

```
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"
```

Multiple input files

As well as using a separate header file you can also provide multiple nodes or relationships files. This may be useful when processing the output from a Hadoop pipeline for example. 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-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"
```

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`. In this example we'll put the label `Movie` on every node specified in `movies5.csv`:

movies5.csv

```
movieId:ID,title,year:int  
tt0133093,"The Matrix",1999
```



There's 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 case, we'll put the labels `Movie` and `Sequel` on the nodes specified in `sequels5.csv`.

sequels5.csv

```
movieId:ID,title,year:int  
tt0234215,"The Matrix Reloaded",2003  
tt0242653,"The Matrix Revolutions",2003
```


actors5.csv

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

roles5.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-import --into path_to_target_directory --nodes:Movie movies5.csv --nodes:Movie:Sequel sequels5.csv
--nodes:Actor actors5.csv --relationships roles5.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 you can do this by specifying the appropriate value as an option to `neo4j-import`. In this example we'll put the relationship type `ACTS_IN` on every relationship specified in `roles6.csv`:

movies6.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
```

actors6.csv

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

roles6.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
```



If you provide a relationship type on the command line and in the relationships file the one in the file will be applied.

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

```
neo4j-import --into path_to_target_directory --nodes movies6.csv --nodes actors6.csv
--relationships:ACTED_IN roles6.csv
```

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.

movies10.csv

```
movieId:ID,title,year:int,:LABEL
tt0099892,"Joe Versus the Volcano",1990,Movie
```

actors10.csv

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

roles10.csv

```
:START_ID,roles:string[],:END_ID,:TYPE
meg,"DeDe;Angelica Graynamore;Patricia Graynamore",tt0099892,ACTED_IN
```

The arguments to **neo4j-import** would be the following:

```
neo4j-import --into path_to_target_directory --nodes movies10.csv --nodes actors10.csv --relationships
roles10.csv
```

ID handling

Each node processed by **neo4j-import** must provide a unique id. We use this id 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),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),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),role,:END_ID(Movie)
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 command line arguments would remain the same as before:

```
neo4j-import --into path_to_target_directory --nodes movies7.csv --nodes actors7.csv
--relationships:ACTED_IN roles7.csv
```

Bad input data

The import tool has a threshold of how many bad entities (nodes/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 `<<import-tool-option-bad-tolerance, --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 `<<import-tool-option-skip-bad-relationships, --skip-bad-relationships>>` option.

In the following example there is a missing `emil` node referenced in the roles file.

movies9.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
```

actors9.csv

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

roles9.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 command line arguments would remain the same as before:

```
neo4j-import --into path_to_target_directory --nodes movies9.csv --nodes actors9.csv --relationships
roles9.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: roles9.csv:11
  properties: [role, Emil]
  startNode: emil
  endNode: tt0133093
  type: ACTED_IN
  referring to missing node emil
```

Multiple nodes with same id within same id space

Nodes that specify **:ID** which has already been specified within the id space are considered bad nodes. Whether or not such nodes are skipped is controlled with **--skip-duplicate-nodes** flag which can have the values **true** or **false** or no value, which means **true**. Specifying **false** means that any duplicate node is considered an error and will fail the import. For more information, see the `<<import-tool-option-skip-duplicate-nodes, --skip-duplicate-nodes>>` option.

In the following example there is a node id that is specified twice within the same id space.

actors10.csv

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

```
neo4j-import --into path_to_target_directory --nodes actors10.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 actors10.csv:3 and actors10.csv:5
```

Chapter 3. Security

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 webservice 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) (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

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. Arbitrary code execution



The Neo4j server exposes remote scripting functionality by default that allow full access to the underlying system. Exposing your server without implementing a security layer presents a substantial security vulnerability.

By default, the Neo4j database comes with some places where arbitrary code code execution can happen. These are the *REST endpoints*. To secure these, either disable them completely by removing offending plugins from the server classpath, or secure access to these URLs through proxies or Authorization Rules. Also, the Java Security Manager, see <http://docs.oracle.com/javase/7/docs/technotes/guides/security/index.html>, can be used to secure parts of the codebase.

3.1.5. Server authorization rules

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](#) (<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.

3.1.6. 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"

3.1.7. 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"

3.1.8. 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_repla

cement/a/b/c/final/bit/more/stuff

- **Accept:** application/json

Example response

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

3.1.9. Using a proxy

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
```

3.1.10. LOAD CSV

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. Backup



The backup features are only available in the Neo4j Enterprise Edition.

4.1. Introducing backup

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



Neo4j Server must be configured to run a backup service. This is enabled via the configuration parameter `dbms.backup.enabled`, and is enabled by default. The interface and port the backup service listens on is configured via the parameter `dbms.backup.address` and defaults to the loopback interface and port 6362. It is typical to reconfigure this to listen on an external interface, by setting `dbms.backup.address=<my-host-ip-address>:6362`. It can also be configured to listen on all interfaces by setting `dbms.backup.address=0.0.0.0:6362`.

Performing a backup requires specifying the target host, an optional port, and the backup location. The backup tool will automatically select a full or incremental backup, based on whether an existing backup is present at that location.

See the [configuration reference section](#) for detailed documentation on available configuration options.

4.2. Performing backups

4.2.1. Backup commands

```
# 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
```

4.2.2. 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.

However, 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.

4.3. Restoring a backup

The Neo4j backups are fully functional databases. To use a backup, simply shut down the database and replace all the files in the data directory with the backup. Then start the database.

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 5. Monitoring



Most of the monitoring features are only available in the Enterprise edition of Neo4j.

In order to be able to continuously get an overview of the health of a Neo4j database, there are different levels of monitoring facilities available. Most of these are exposed through [JMX](http://www.oracle.com/technetwork/java/javase/tech/javamanagement-140525.html) (<http://www.oracle.com/technetwork/java/javase/tech/javamanagement-140525.html>). Neo4j Enterprise also has the ability to automatically report *metrics* to commonly used monitoring systems.

5.1. Adjusting remote JMX access to the Neo4j Server

Per default, the Neo4j Enterprise Server edition does not allow remote JMX connections, since the relevant options in the 'conf/neo4j-wrapper.conf' configuration file are commented out. To enable this feature, you have to remove the # characters from the various com.sun.management.jmxremote options there.

When commented in, the default values are set up to allow remote JMX connections with certain roles, refer to the 'conf/jmx.password', 'conf/jmx.access' and 'conf/neo4j-wrapper.conf' files for details.

Make sure that 'conf/jmx.password' has the correct file permissions. The owner of the file has to be the user that will run the service, and the permissions should be read only for that user. On Unix systems, this is 0600.

On Windows, follow the tutorial at <http://docs.oracle.com/javase/7/docs/technotes/guides/management/security-windows.html> to set the correct permissions. If you are running the service under the Local System Account, the user that owns the file and has access to it should be SYSTEM.

With this setup, you should be able to connect to JMX monitoring of the Neo4j server using <IP-OF-SERVER>:3637, with the username monitor and the password Neo4j.

Note that it is possible that you have to update the permissions and/or ownership of the 'conf/jmx.password' and 'conf/jmx.access' files — refer to the relevant section in 'conf/neo4j-wrapper.conf' for details.



For maximum security, please adjust at least the password settings in 'conf/jmx.password' for a production installation.

For more details, see:

<http://docs.oracle.com/javase/7/docs/technotes/guides/management/agent.html>.

5.2. How to connect to a Neo4j instance using JMX and JConsole

First, start your Neo4j database, for instance using

```
$NEO4j_HOME/bin/neo4j start
```

Now, start JConsole with

```
$JAVA_HOME/bin/jconsole
```

Connect to the process running your Neo4j database instance:

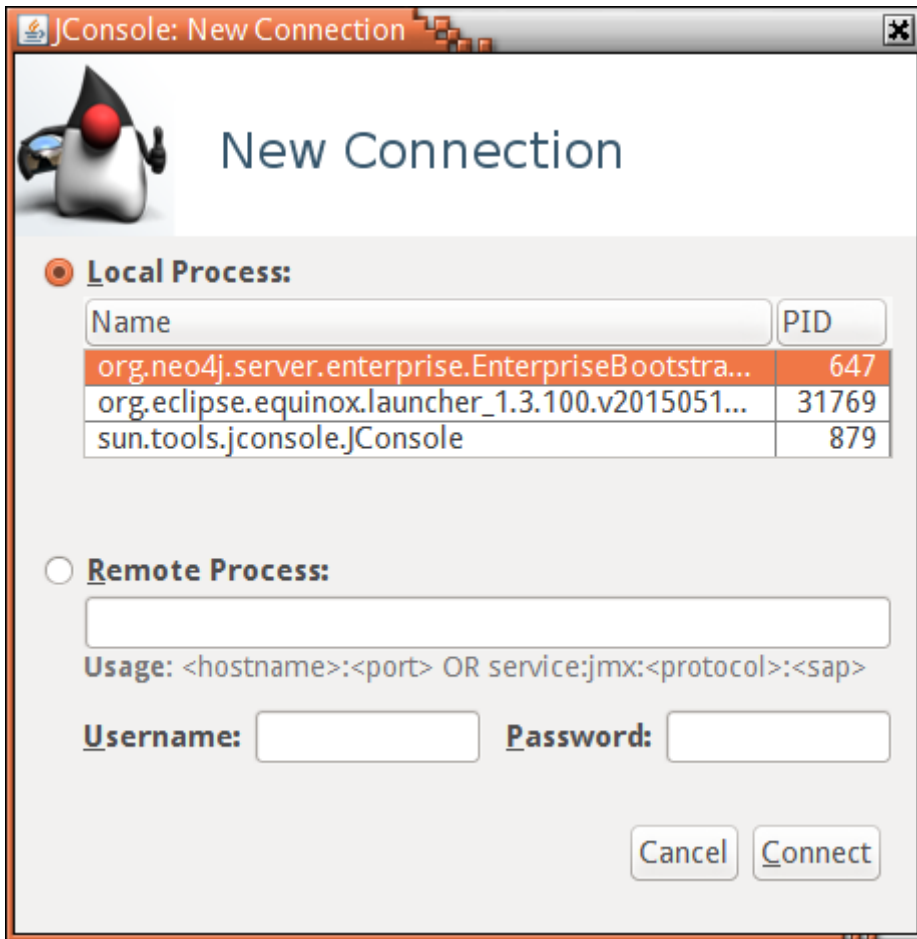


Figure 5. Connecting JConsole to the Neo4j Java process

Now, beside the MBeans exposed by the JVM, you will see an org.neo4j section in the MBeans tab. Under that, you will have access to all the monitoring information exposed by Neo4j.

For opening JMX to remote monitoring access, please see [Adjusting remote JMX access to the Neo4j Server](#) and [the JMX documentation](#) (<http://docs.oracle.com/javase/7/docs/technotes/guides/management/agent.html>).

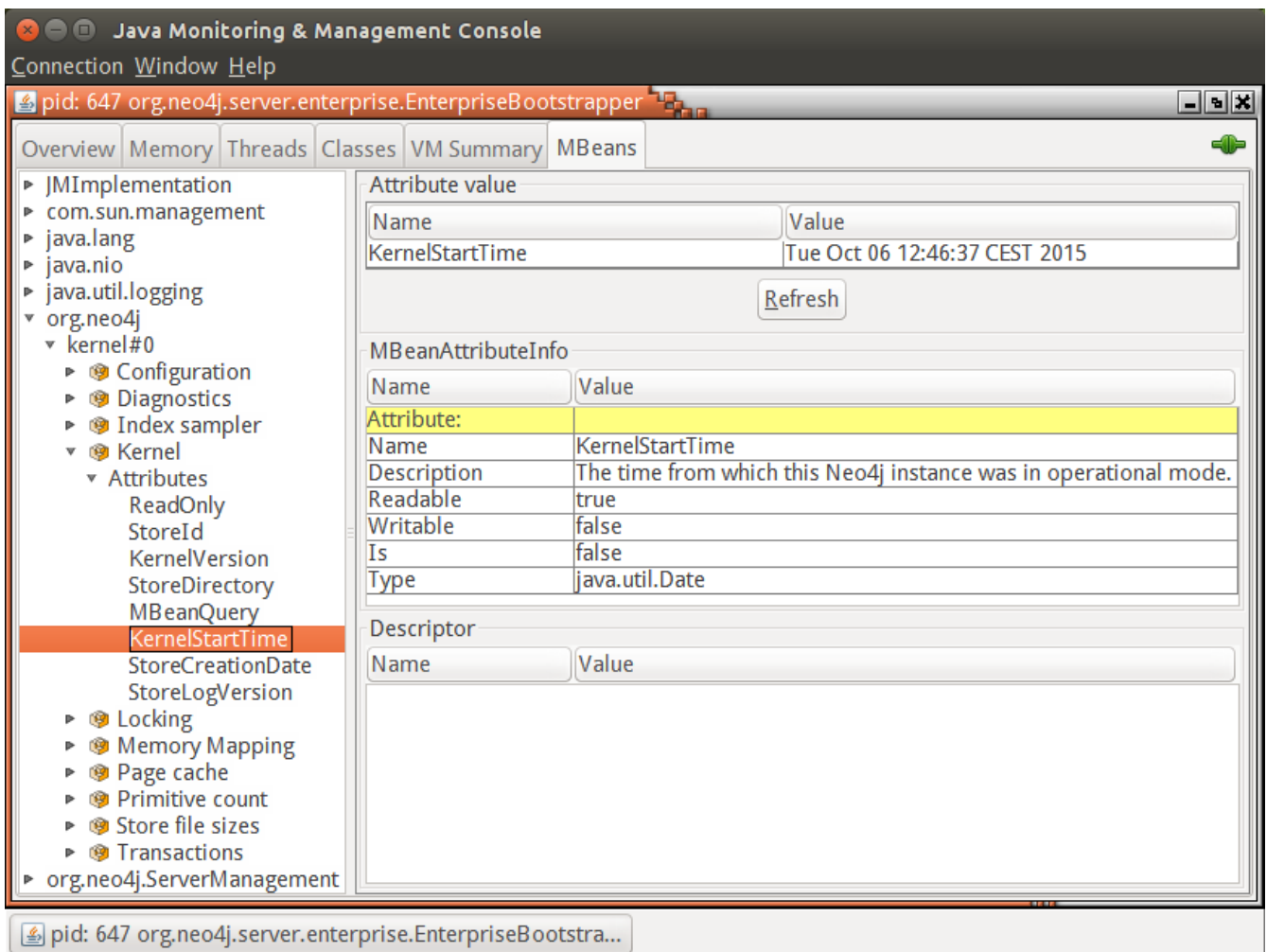


Figure 6. Neo4j MBeans View

5.3. Reference of supported JMX MBeans

For a reference to all the parameters specific to MBeans exposed by Neo4j, see [MBeans exposed by Neo4j](#).

5.4. Metrics Reporting



Metrics reporting is only available in the Neo4j Enterprise Edition.

5.4.1. Introducing Metrics

Neo4j Enterprise can be configured to continuously export Neo4j-specific metrics to Graphite or CSV files. This makes it easy to monitor the health of running Neo4j instances.

Neo4j Enterprise can expose metrics for the following parts of the database, and does so by default:

```
// default setting for enabling all supported metrics
metrics.enabled=true

// default 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
```

5.4.2. Graphite Configuration

For Graphite integration add the following settings to *neo4j.conf*:

```
metrics.graphite.enabled=true // default is 'false'
metrics.graphite.server=<ip>:2003
metrics.graphite.interval=<how often to send data, defaults to 3s>
metrics.prefix=<Neo4j instance name, e.g. wwwneo1>
```

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

5.4.3. Export to CSV Configuration

For storing metrics in local CSV files add the following settings to *neo4j.conf*:

```
metrics.csv.enabled=true // default is 'false'
metrics.csv.path=<file or directory path, defaults to "metrics/" in the store directory>
metrics.csv.interval=<how often to store data, defaults to 3s>
```



The CSV exporter does not automatically rotate the output files, so it is recommended to also set up a CRON job to periodically archive the files.

5.4.4. Configuration settings reference for metrics

See the [configuration reference section](#) for detailed documentation on available configuration options.

5.4.5. Available metrics

For a reference to all the parameters specific to metrics, see [Available metrics](#).

Chapter 6. Performance tuning

This section describes some of the internal workings of Neo4j memory settings and how to adjust them for optimal performance.

6.1. Modifying configuration settings

6.2. Cypher tuning

The first thing to look at when Neo4j is not performing as expected is how the Cypher queries are being executed. Make sure that they don't do more work than they have to. Some queries may accidentally be written in a way that generates a large cartesian product. Other queries may have to perform expensive label scans because an important index is missing. The Neo4j developer manual has more information on how to investigate Cypher performance issues.

6.3. Memory tuning

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:

$\text{Actual OS allocation} = \text{available RAM} - (\text{page cache} + \text{heap size})$



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

6.3.1. 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.

$\text{OS Memory} = 1\text{GB} + (\text{size of graph.db/index}) + (\text{size of graph.db/schema})$

6.3.2. 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/neostore.*.db` files and adding 20% for growth.

The parameter for specifying 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
<code>dbms.memory.pagecache.size</code>	The maximum amount of memory to use for the page cache, either in bytes, or greater byte-like units, such as <code>100m</code> for 100 mega-bytes, or <code>4g</code> 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.
<code>unsupported.dbms.report_configuration</code>	<code>true</code> or <code>false</code>	If set to <code>true</code> the current configuration settings will be written to the default system output, mostly the console or the logfiles.

6.3.3. 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.

6.3.4. 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 4. neo4j-wrapper.conf JVM tuning properties

Property Name	Meaning
<code>dbms.memory.heap.initial_size</code>	initial heap size (in MB)
<code>dbms.memory.heap.max_size</code>	maximum heap size (in MB)
<code>dbms.jvm.additional</code>	additional literal JVM parameter

6.4. Transaction logs

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"

6.5. Compressed property value storage

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

Compressed storage of short arrays

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

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

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

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

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

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

Compressed storage of short strings

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

The various classes for short strings are:

- Numerical, consisting of digits 0..9 and the punctuation space, period, dash, plus, comma and apostrophe.
- Date, consisting of digits 0..9 and the punctuation space dash, colon, slash, plus and comma.
- Hex (lower case), consisting of digits 0..9 and lower case letters a..f
- Hex (upper case), consisting of digits 0..9 and upper case letters a..f
- Upper case, consisting of upper case letters A..Z, and the punctuation space, underscore, period, dash, colon and slash.
- Lower case, like upper but with lower case letters a..z instead of upper case
- E-mail, consisting of lower case letters a..z and the punctuation comma, underscore, period, dash, plus and the at sign (@).
- URI, consisting of lower case letters a..z, digits 0..9 and most punctuation available.
- Alpha-numerical, consisting of both upper and lower case letters a..zA..Z, digits 0..9 and punctuation space and underscore.
- Alpha-symbolical, consisting of both upper and lower case letters a..zA..Z and the punctuation space, underscore, period, dash, colon, slash, plus, comma, apostrophe, at sign, pipe and semicolon.
- European, consisting of most accented european characters and digits plus punctuation space, dash, underscore and period — like latin1 but with less punctuation.
- Latin 1.

- UTF-8.

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

Table 5. Character count limits

String class	Character count limit
Numerical, Date and Hex	54
Uppercase, Lowercase and E-mail	43
URI, Alphanumerical and Alphasympobical	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.

6.6. Linux file system tuning

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.

6.7. Disks, RAM and other tips

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.

Chapter 7. Tutorials

7.1. 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](#).

7.1.1. Important configuration settings

Each instance in a Neo4j HA cluster must be assigned an integer ID, which serves as its unique identifier. At startup, a Neo4j instance contacts the other instances specified in the `ha.initial_hosts` configuration option.

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. Neo4j will attempt to configure IP addresses for itself in the absence of explicit configuration.

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 communications (like heartbeat messages). The default port is `5001`. In the absence of a specified IP address, Neo4j will attempt to find a valid interface for binding. While this behavior typically results in a well-behaved server, it is *strongly recommended* that users explicitly choose an IP address bound to the network interface of their choosing to ensure a coherent cluster deployment.

For example, `ha.host.coordination=192.168.33.22:5001` will listen for cluster communications on the network interface bound to the 192.168.33.0 subnet on port 5001.

ha.initial_hosts

`ha.initial_hosts` is a comma separated list of address/port pairs, which specify 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. Specifying an instance's own address is permitted. Do not use any whitespace in this configuration option.

For example, `ha.initial_hosts=192.168.33.22:5001,192.168.33.21:5001` will attempt to reach Neo4j instances listening on 192.168.33.22 on port 5001 and 192.168.33.21 on port 5001 on the 192.168.33.0 subnet.

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`. In the absence of a specified IP address, Neo4j will attempt to find a valid interface for binding. While this behavior typically results in a well-behaved server, it is *strongly recommended* that users explicitly choose an IP address bound to the network interface of their choosing to ensure a coherent cluster topology.

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

For example, `ha.host.data=192.168.33.22:6001` will listen for transactions from the cluster master on the network interface bound to the 192.168.33.0 subnet on port 6001.

Address and port formats

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

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



For `ha.host.coordination` the IP address *must be* the 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 address is omitted, then the port must be preceded with a colon (eg. `:5001`).

The syntax for setting the port range is: `<hostname>:<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).

7.1.2. Download and configure

- Download Neo4j Enterprise from [the Neo4j download site](http://neo4j.com/download/) (<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
```

7.1.3. 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. This is because the instance does not accept any requests until a cluster has been formed. In the example above this happens when you start the second instance. 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 HA cluster of three instances running. You can start by making a change on any instance and those changes will be propagated between them. For more HA related configuration options take a look at [Setup and configuration](#).

7.2. 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.

7.2.1. Download and configure

1. Download Neo4j Enterprise from [the Neo4j download site](http://neo4j.com/download/) (<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

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

Neo4j instance #2 — ~/neo4j-02

conf/neo4j.conf

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

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

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

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

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

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

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

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

Neo4j instance #3 — ~/neo4j-03

conf/neo4j.conf

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

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

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

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

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

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

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

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

Start the Neo4j Servers

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

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

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

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

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

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

Chapter 8. Configuration

8.1. Configuration Settings Reference

This page documents Neo4j's configuration settings. They can be set in [neo4j.conf](#).

Table 6. 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 <code>LOAD CSV</code> clause.
dbms.directories.lib	Path of the lib directory.
dbms.directories.logs	Path of the logs directory.
dbms.directories.metrics	The target location of the CSV files: a path to a directory wherein a CSV file per reported field will be written.
dbms.directories.plugins	Location of the database plugin directory.
dbms.directories.run	Path of the run directory.
dbms.index_sampling.background_enabled	Enable or disable background index sampling.
dbms.index_sampling.buffer_size	Size of buffer used by index sampling.

Name	Description
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 takes longer than the configured 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 <code>LOAD CSV</code> .
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-separated list of custom security rules for Neo4j to use.
dbms.shell.enabled	Enable a remote shell server which Neo4j Shell clients can log in to.

Name	Description
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	Timeout for idle transactions.
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_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	Timeout for heartbeats between cluster members.
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_channels_per_slave	Maximum number of connections a slave can have to the master.
ha.paxos_timeout	Default timeout 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.

Name	Description
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 <code>true</code> 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 <code>true</code> 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.
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 7. *browser.allow_outgoing_connections*

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

Table 8. *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 <code>ms</code> , <code>s</code> , <code>m</code>)
Default value	<code>0</code>

Table 9. *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	<code>http://guides.neo4j.com,https://guides.neo4j.com,http://localhost,https://localhost</code>

Table 10. *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	<code>true</code>

Table 11. *cypher.default_language_version*

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

Table 12. *cypher.forbid_exhaustive_shortestpath*

Description	This setting is associated with performance optimization. Set this to <code>true</code> in situations where it is preferable to have any queries using the 'shortestPath' function terminate as soon as possible with no answer, rather than potentially running for a long time attempting to find an answer (even if there is no path to be found). For most queries, the 'shortestPath' algorithm will return the correct answer very quickly. However there are some cases where it is possible that the fast bidirectional breadth-first search algorithm will find no results even if they exist. This can happen when the predicates in the 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 <code>true</code> , which means that Neo4j will never consider using the exhaustive search for shortestPath queries. However, please note that if no paths are found, an error will be thrown at run time, which will need to be handled by the application.
Valid values	cypher.forbid_exhaustive_shortestpath is a boolean
Default value	<code>false</code>

Table 13. *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	<code>false</code>

Table 14. *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 <code>ms</code> , <code>s</code> , <code>m</code>)
Default value	<code>1000</code>

Table 15. *cypher.planner*

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

Table 16. *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 <code>0.0</code> , and is maximum <code>1.0</code>
Default value	<code>0.5</code>

Table 17. *dbms.active_database*

Description	Name of the database to load.
Valid values	dbms.active_database is a string
Default value	<code>graph.db</code>

Table 18. *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 <code>true</code> does not guarantee successful upgrade, it just allows an upgrade to be performed.
Valid values	dbms.allow_format_migration is a boolean
Default value	<code>false</code>

Table 19. *dbms.backup.address*

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

Table 20. *dbms.backup.enabled*

Description	Enable support for running online backups.
Valid values	dbms.backup.enabled is a boolean
Default value	<code>true</code>

Table 21. *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. The default is '5m' for a check-point every 5 minutes. Other supported units are 's' for seconds, and 'ms' for milliseconds.
Valid values	dbms.checkpoint.interval.time is a duration (valid units are <code>ms</code> , <code>s</code> , <code>m</code>)

Default value	300000
---------------	--------

Table 22. *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 23. *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 24. *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 25. *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 26. *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 27. *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 28. *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 29. *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 30. *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 31. *dbms.directories.run*

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

Table 32. *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 33. *dbms.index_sampling.buffer_size*

Description	Size of buffer used by index sampling.
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

Table 34. *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 35. *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 36. *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 37. *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 value	300000

Table 38. *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 39. *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 40. *dbms.logs.gc.enabled*

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

Table 41. *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 42. *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 43. *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 44. *dbms.logs.http.enabled*

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

Table 45. *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 46. *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 47. *dbms.logs.query.enabled*

Description	Log executed queries that takes longer than the configured threshold. <i>NOTE: This feature is only available in the Neo4j Enterprise Edition.</i>
Valid values	dbms.logs.query.enabled is a boolean
Default value	false

Table 48. *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 49. *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 50. *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 51. *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 value	0

Table 52. *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	3443740672

Table 53. *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 54. *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	<code>SINGLE</code>

Table 55. *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 <code>0</code>
Default value	<code>1000</code>

Table 56. *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	<code>false</code>

Table 57. *dbms.record_format*

Description	Database record format. Enterprise edition only. Valid values: <code>standard</code> , <code><<config_high_limit,high_limit>></code> . Default value: <code>standard</code> .
Valid values	dbms.record_format is a string
Default value	

Table 58. *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 <code>1</code>
Default value	<code>50</code>

Table 59. *dbms.security.allow_csv_import_from_file_urls*

Description	Determines if Cypher will allow using file URLs when loading data using <code>LOAD CSV</code> . Setting this value to <code>false</code> will cause Neo4j to fail <code>LOAD CSV</code> clauses that load data from the file system.
Valid values	dbms.security.allow_csv_import_from_file_urls is a boolean
Default value	<code>true</code>

Table 60. *dbms.security.auth_enabled*

Description	Enable auth requirement to access Neo4j.
Valid values	dbms.security.auth_enabled is a boolean
Default value	<code>false</code>

Table 61. *dbms.security.ha_status_auth_enabled*

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

Default value	<code>true</code>
---------------	-------------------

Table 62. *dbms.security.http_authorization_classes*

Description	Comma-separated list of custom security rules for Neo4j to use.
Valid values	<code>dbms.security.http_authorization_classes</code> is a comma-separated string
Default value	<code>[]</code>

Table 63. *dbms.shell.enabled*

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

Table 64. *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 <code>0.0.0.0</code> for all interfaces.
Valid values	<code>dbms.shell.host</code> is a string which must be a valid name
Default value	<code>127.0.0.1</code>

Table 65. *dbms.shell.port*

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

Table 66. *dbms.shell.read_only*

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

Table 67. *dbms.shell.rmi_name*

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

Table 68. *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	<code>dbms.threads.worker_count</code> is an integer which is in the range 1 to 44738
Default value	<code>2</code>

Table 69. *dbms.transaction_timeout*

Description	Timeout for idle transactions.
Valid values	<code>dbms.transaction_timeout</code> is a duration (valid units are <code>ms</code> , <code>s</code> , <code>m</code>)
Default value	<code>60000</code>

Table 70. *dbms.tx_log.rotation.retention_policy*

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

Table 71. *dbms.tx_log.rotation.size*

Description	Specifies at which file size the logical log will auto-rotate. <code>0</code> 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 <code>k, m, g, K, M, G</code>) which is minimum <code>1048576</code>
Default value	<code>262144000</code>

Table 72. *dbms.udc.enabled*

Description	Enable the UDC extension.
Valid values	dbms.udc.enabled is a boolean
Default value	<code>true</code>

Table 73. *dbms.unmanaged_extension_classes*

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

Table 74. *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	<code>true</code>

Table 75. *ha.branched_data_policy*

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

Table 76. *ha.broadcast_timeout*

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

Table 77. *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 <code>ms, s, m</code>)
Default value	<code>1000</code>

Table 78. *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 79. *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 <<config_ha.heartbeat_interval,ha.heartbeat_interval>>, <<config_ha.paxos_timeout,ha.paxos_timeout>> and <<config_ha.learn_timeout,ha.learn_timeout>> settings.
Valid values	ha.default_timeout is a duration (valid units are ms, s, m)
Default value	5000

Table 80. *ha.election_timeout*

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

Table 81. *ha.heartbeat_interval*

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

Table 82. *ha.heartbeat_timeout*

Description	Timeout for heartbeats between cluster members. Should be at least twice that of <<config_ha.heartbeat_interval,ha.heartbeat_interval>>.
Valid values	ha.heartbeat_timeout is a duration (valid units are ms, s, m)
Default value	11000

Table 83. *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 84. *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 85. *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 86. *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 value	10000

Table 87. *ha.join_timeout*

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

Table 88. *ha.learn_timeout*

Description	Timeout for learning values. Defaults to <<config_ha.default_timeout,ha.default_timeout>>.
Valid values	ha.learn_timeout is a duration (valid units are ms, s, m)
Default value	5000

Table 89. *ha.leave_timeout*

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

Table 90. *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 91. *ha.paxos_timeout*

Description	Default timeout for all Paxos timeouts. Defaults to <<config_ha.default_timeout,ha.default_timeout>>. This value is the default value for the <<config_ha.phase1_timeout,ha.phase1_timeout>>, <<config_ha.phase2_timeout,ha.phase2_timeout>> and <<config_ha.election_timeout,ha.election_timeout>> settings.
Valid values	ha.paxos_timeout is a duration (valid units are ms, s, m)
Default value	5000

Table 92. *ha.phase1_timeout*

Description	Timeout for Paxos phase 1. Defaults to <<config_ha.paxos_timeout,ha.paxos_timeout>>.
Valid values	ha.phase1_timeout is a duration (valid units are ms, s, m)
Default value	5000

Table 93. *ha.phase2_timeout*

Description	Timeout for Paxos phase 2. Defaults to <<config_ha.paxos_timeout,ha.paxos_timeout>>.
Valid values	ha.phase2_timeout is a duration (valid units are ms, s, m)
Default value	5000

Table 94. *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 95. 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 value	0

Table 96. 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 value	120000

Table 97. 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 98. ha.slave_lock_timeout

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

Table 99. 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 100. 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 value	20000

Table 101. 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 102. 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 103. *metrics.bolt.messages.enabled*

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

Table 104. *metrics.csv.enabled*

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

Table 105. *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 <code>ms</code> , <code>s</code> , <code>m</code>)
Default value	<code>3000</code>

Table 106. *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	<code>false</code>

Table 107. *metrics.enabled*

Description	The default enablement value for all the supported metrics. Set this to <code>false</code> 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	<code>false</code>

Table 108. *metrics.graphite.enabled*

Description	Set to <code>true</code> to enable exporting metrics to Graphite.
Valid values	metrics.graphite.enabled is a boolean
Default value	<code>false</code>

Table 109. *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 <code>ms</code> , <code>s</code> , <code>m</code>)
Default value	<code>3000</code>

Table 110. *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	<code>:2003</code>

Table 111. *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 112. *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 113. *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 114. *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 115. *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 116. *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 117. *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 118. *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 <code>metrics.neo4j.*</code> metrics can then be turned on selectively.
Valid values	metrics.neo4j.enabled is a boolean
Default value	false

Table 119. *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 120. *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 121. *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 122. *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 123. *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 124. *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 125. *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 126. *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 127. *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 128. *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	<code>tools.consistency_checker.check_property_owners</code> is a boolean
Default value	<code>false</code>

8.1.1. Configuring Bolt Connectors

Bolt Connectors are ports that accept connections via the Bolt Database Protocol, which is the protocol used by official Neo4j Driver Libraries. Neo4j can be configured with one or more Bolt connectors. This allows separate connectors to be configured for remote and local connections, with different encryption requirements.

Each connector has a unique key to identify it, denoted (`bolt-connector-key`) in the listing below.

Table 129. *Configuration options for Bolt connectors. "(bolt-connector-key)" is a placeholder for a unique name for the connector, for instance "bolt-public" or some other name that describes what the connector is for.*

Name	Description
<code>dbms.connector.(bolt-connector-key).address</code>	Address the connector should bind to.
<code>dbms.connector.(bolt-connector-key).enabled</code>	Enable this connector.
<code>dbms.connector.(bolt-connector-key).tls_level</code>	Encryption level to require this connector to use.
<code>dbms.connector.(bolt-connector-key).type</code>	Connector type.

Table 130. *dbms.connector.(bolt-connector-key).address*

Description	Address the connector should bind to.
Valid values	address is a hostname and port
Default value	<code>localhost:7687</code>

Table 131. *dbms.connector.(bolt-connector-key).enabled*

Description	Enable this connector.
Valid values	enabled is a boolean
Default value	<code>false</code>

Table 132. *dbms.connector.(bolt-connector-key).tls_level*

Description	Encryption level to require this connector to use.
Valid values	tls_level is one of <code>REQUIRED</code> , <code>OPTIONAL</code> , <code>DISABLED</code>
Default value	<code>OPTIONAL</code>

Table 133. *dbms.connector.(bolt-connector-key).type*

Description	Connector type. You should always set this to the connector type you want.
Valid values	type is one of <code>BOLT</code> , <code>HTTP</code>
Default value	<code>BOLT</code>

8.1.2. Configuring HTTP Connectors

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 configured.

Each connector has a unique key to identify it, denoted (`http-connector-key`) in the listing below.

Table 134. Configuration options for HTTP connectors. "(http-connector-key)" is a placeholder for a unique name for the connector, for instance "http-public" or some other name that describes what the connector is for.

Name	Description
<code>dbms.connector.(http-connector-key).address</code>	Address the connector should bind to.
<code>dbms.connector.(http-connector-key).enabled</code>	Enable this connector.
<code>dbms.connector.(http-connector-key).encryption</code>	Enable TLS for this connector.
<code>dbms.connector.(http-connector-key).type</code>	Connector type.

Table 135. `dbms.connector.(http-connector-key).address`

Description	Address the connector should bind to.
Valid values	address is a hostname and port
Default value	<code>localhost:7474</code>

Table 136. `dbms.connector.(http-connector-key).enabled`

Description	Enable this connector.
Valid values	enabled is a boolean
Default value	<code>false</code>

Table 137. `dbms.connector.(http-connector-key).encryption`

Description	Enable TLS for this connector.
Valid values	encryption is one of <code>NONE</code> , <code>TLS</code>
Default value	<code>NONE</code>

Table 138. `dbms.connector.(http-connector-key).type`

Description	Connector type. You should always set this to the connector type you want.
Valid values	type is one of <code>BOLT</code> , <code>HTTP</code>
Default value	<code>HTTP</code>

8.2. JMX Beans

Table 139. MBeans exposed by Neo4j

Name	Description
Branched Store	Information about the branched stores present in this HA cluster member.
Configuration	The configuration parameters used to configure Neo4j.
Diagnostics	Diagnostics provided by Neo4j.
High Availability	Information about an instance participating in a HA cluster.

Name	Description
Index sampler	Handle index sampling.
Kernel	Information about the Neo4j kernel.
Locking	Information about the Neo4j lock status.
Memory Mapping	The status of Neo4j memory mapping.
Page cache	Information about the Neo4j page cache. All numbers are counts and sums since the Neo4j instance was started.
Primitive count	Estimates of the numbers of different kinds of Neo4j primitives.
Store file sizes	Information about the sizes of the different parts of the Neo4j graph store.
Transactions	Information about the Neo4j transaction manager.



For additional information on the primitive datatypes (int, long etc.) used in the JMX attributes, please see [\[property-value-types\]](#) in [\[graphdb-neo4j-properties\]](#).

Table 140. MBean Branched Store (*org.neo4j.management.BranchedStore*) Attributes

Name	Description	Type	Read	Write
<i>Information about the branched stores present in this HA cluster member</i>				
<code>BranchedStores</code>	A list of the branched stores	<code>org.neo4j.management.BranchedStoreInfo</code> (javadocs/org/neo4j/management/BranchedStoreInfo.html)[] as <code>CompositeData</code> (http://docs.oracle.com/javase/7/docs/api/javax/management/openmbean/CompositeData.html)[]	yes	no

Table 141. MBean Configuration (*org.neo4j.jmx.impl.ConfigurationBean*) Attributes

Name	Description	Type	Read	Write
<i>The configuration parameters used to configure Neo4j</i>				
<code>cypher.default_language_version</code>	Set this to specify the default parser (language version).	<code>String</code>	yes	no

Name	Description	Type	Read	Write
<code>cypher.forbid_exhaustive_shortestpath</code>	This setting is associated with performance optimization. Set this to <code>true</code> in situations where it is preferable to have any queries using the 'shortestPath' function terminate as soon as possible with no answer, rather than potentially running for a long time attempting to find an answer (even if there is no path to be found). For most queries, the 'shortestPath' algorithm will return the correct answer very quickly. However there are some cases where it is possible that the fast bidirectional breadth-first search algorithm will find no results even if they exist. This can happen when the predicates in the <code>WHERE</code> clause applied to 'shortestPath' cannot be applied to each step of the traversal, and can only be applied to the entire path. When the query planner detects these special cases, it will plan to perform an exhaustive depth-first search if the fast algorithm finds no paths. However, the exhaustive search may be orders of magnitude slower than the fast algorithm. If it is critical that queries terminate as soon as possible, it is recommended that this option be set to <code>true</code> , which means that Neo4j will never consider using the exhaustive search for shortestPath queries. However, please note that if no paths are found, an error will be thrown at run time, which will need to be handled by the application.	String	yes	no
<code>cypher.hints_error</code>	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.	String	yes	no
<code>cypher.planner</code>	Set this to specify the default planner for the default language version.	String	yes	no
<code>dbms.allow_format_migration</code>	Whether to allow a store upgrade in case the current version of the database starts against an older store version. Setting this to <code>true</code> does not guarantee successful upgrade, it just allows an upgrade to be performed.	String	yes	no
<code>dbms.auto_index.nodes.enabled</code>	Controls the auto indexing feature for nodes. Setting it to <code>false</code> shuts it down, while <code>true</code> enables it by default for properties listed in the <code>dbms.auto_index.nodes.keys</code> setting.	String	yes	no
<code>dbms.auto_index.nodes.keys</code>	A list of property names (comma separated) that will be indexed by default. This applies to <i>nodes</i> only.	String	yes	no
<code>dbms.auto_index.relationships.enabled</code>	Controls the auto indexing feature for relationships. Setting it to <code>false</code> shuts it down, while <code>true</code> enables it by default for properties listed in the <code>dbms.auto_index.relationships.keys</code> setting.	String	yes	no
<code>dbms.auto_index.relationships.keys</code>	A list of property names (comma separated) that will be indexed by default. This applies to <i>relationships</i> only.	String	yes	no
<code>dbms.backup.address</code>	Listening server for online backups	String	yes	no
<code>dbms.backup.enabled</code>	Enable support for running online backups	String	yes	no

Name	Description	Type	Read	Write
<code>dbms.checkpoint.interval.time</code>	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. The default is '5m' for a check-point every 5 minutes. Other supported units are 's' for seconds, and 'ms' for milliseconds.	String	yes	no
<code>dbms.checkpoint.interval.tx</code>	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.	String	yes	no
<code>dbms.checkpoint.iops.limit</code>	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.	String	yes	no
<code>dbms.directories.logs</code>	Path of the logs directory	String	yes	no
<code>dbms.directories.plugins</code>	Location of the database plugin directory. Compiled Java JAR files that contain database procedures will be loaded if they are placed in this directory.	String	yes	no
<code>dbms.index_sampling.background_enabled</code>	Enable or disable background index sampling	String	yes	no
<code>dbms.index_sampling.buffer_size</code>	Size of buffer used by index sampling	String	yes	no
<code>dbms.index_sampling.update_percentage</code>	Percentage of index updates of total index size required before sampling of a given index is triggered	String	yes	no
<code>dbms.logs.debug.level</code>	Debug log level threshold.	String	yes	no

Name	Description	Type	Read	Write
<code>dbms.logs.debug.rotation.delay</code>	Minimum time interval after last rotation of the debug log before it may be rotated again.	String	yes	no
<code>dbms.logs.debug.rotation.keep_number</code>	Maximum number of history files for the debug log.	String	yes	no
<code>dbms.logs.debug.rotation.size</code>	Threshold for rotation of the debug log.	String	yes	no
<code>dbms.logs.query.enabled</code>	Log executed queries that takes longer than the configured threshold. <i>NOTE: This feature is only available in the Neo4j Enterprise Edition.</i>	String	yes	no
<code>dbms.logs.query.parameter_logging_enabled</code>	Log parameters for executed queries that took longer than the configured threshold.	String	yes	no
<code>dbms.logs.query.rotation.keep_number</code>	Maximum number of history files for the query log.	String	yes	no
<code>dbms.logs.query.rotation.size</code>	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 <code>k</code> , <code>m</code> or <code>g</code> .	String	yes	no
<code>dbms.logs.query.threshold</code>	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.	String	yes	no
<code>dbms.memory.pagecache.size</code>	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.	String	yes	no
<code>dbms.memory.pagecache.swapper</code>	Specify which page swapper to use for doing paged IO. This is only used when integrating with proprietary storage technology.	String	yes	no
<code>dbms.read_only</code>	Only allow read operations from this Neo4j instance. This mode still requires write access to the directory for lock purposes.	String	yes	no
<code>dbms.record_format</code>	Database record format. Enterprise edition only. Valid values: <code>standard</code> , <code>high_limit</code> . Default value: <code>standard</code> .	String	yes	no
<code>dbms.relationship_grouping_threshold</code>	Relationship count threshold for considering a node to be dense	String	yes	no
<code>dbms.security.auth_enabled</code>	Enable auth requirement to access Neo4j.	String	yes	no
<code>dbms.security.ha_status_auth_enabled</code>	Require authorization for access to the HA status endpoints.	String	yes	no
<code>dbms.shell.enabled</code>	Enable a remote shell server which Neo4j Shell clients can log in to.	String	yes	no
<code>dbms.shell.host</code>	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 <code>0.0.0.0</code> for all interfaces.	String	yes	no
<code>dbms.shell.port</code>	The port the shell will listen on.	String	yes	no
<code>dbms.shell.read_only</code>	Read only mode. Will only allow read operations.	String	yes	no

Name	Description	Type	Read	Write
<code>dbms.shell.rmi_name</code>	The name of the shell.	String	yes	no
<code>dbms.tx_log.rotation.retention_policy</code>	Make Neo4j keep the logical transaction logs for being able to backup the database. Can be used for specifying the threshold to prune logical logs after. 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.	String	yes	no
<code>dbms.tx_log.rotation.size</code>	Specifies at which file size the logical log will auto-rotate. 0 means that no rotation will automatically occur based on file size.	String	yes	no
<code>ha.allow_init_cluster</code>	Whether to allow this instance to create a cluster if unable to join.	String	yes	no
<code>ha.branched_data_policy</code>	Policy for how to handle branched data.	String	yes	no
<code>ha.broadcast_timeout</code>	Timeout for broadcasting values in cluster. Must consider end-to-end duration of Paxos algorithm. This value is the default value for the <code>ha.join_timeout</code> and <code>ha.leave_timeout</code> settings.	String	yes	no
<code>ha.configuration_timeout</code>	Timeout for waiting for configuration from an existing cluster member during cluster join.	String	yes	no
<code>ha.data_chunk_size</code>	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.	String	yes	no
<code>ha.default_timeout</code>	Default timeout used for clustering timeouts. Override specific timeout settings with proper values if necessary. This value is the default value for the <code>ha.heartbeat_interval</code> , <code>ha.paxos_timeout</code> and <code>ha.learn_timeout</code> settings.	String	yes	no
<code>ha.election_timeout</code>	Timeout for waiting for other members to finish a role election. Defaults to <code>ha.paxos_timeout</code> .	String	yes	no
<code>ha.heartbeat_interval</code>	How often heartbeat messages should be sent. Defaults to <code>ha.default_timeout</code> .	String	yes	no
<code>ha.heartbeat_timeout</code>	Timeout for heartbeats between cluster members. Should be at least twice that of <code>ha.heartbeat_interval</code> .	String	yes	no
<code>ha.host.coordination</code>	Host and port to bind the cluster management communication.	String	yes	no
<code>ha.host.data</code>	Hostname and port to bind the HA server.	String	yes	no
<code>ha.initial_hosts</code>	A comma-separated list of other members of the cluster to join.	String	yes	no
<code>ha.internal_role_switch_timeout</code>	Timeout for waiting for internal conditions during state switch, like for transactions to complete, before switching to master or slave.	String	yes	no
<code>ha.join_timeout</code>	Timeout for joining a cluster. Defaults to <code>ha.broadcast_timeout</code> .	String	yes	no
<code>ha.learn_timeout</code>	Timeout for learning values. Defaults to <code>ha.default_timeout</code> .	String	yes	no
<code>ha.leave_timeout</code>	Timeout for waiting for cluster leave to finish. Defaults to <code>ha.broadcast_timeout</code> .	String	yes	no

Name	Description	Type	Read	Write
<code>ha.max_channels_per_slave</code>	Maximum number of connections a slave can have to the master.	String	yes	no
<code>ha.paxos_timeout</code>	Default timeout for all Paxos timeouts. Defaults to <code>ha.default_timeout</code> . This value is the default value for the <code>ha.phase1_timeout</code> , <code>ha.phase2_timeout</code> and <code>ha.election_timeout</code> settings.	String	yes	no
<code>ha.phase1_timeout</code>	Timeout for Paxos phase 1. Defaults to <code>ha.paxos_timeout</code> .	String	yes	no
<code>ha.phase2_timeout</code>	Timeout for Paxos phase 2. Defaults to <code>ha.paxos_timeout</code> .	String	yes	no
<code>ha.pull_batch_size</code>	Size of batches of transactions applied on slaves when pulling from master	String	yes	no
<code>ha.pull_interval</code>	Interval of pulling updates from master.	String	yes	no
<code>ha.role_switch_timeout</code>	Timeout for request threads waiting for instance to become master or slave.	String	yes	no
<code>ha.server_id</code>	Id for a cluster instance. Must be unique within the cluster.	String	yes	no
<code>ha.slave_lock_timeout</code>	Timeout for taking remote (write) locks on slaves. Defaults to <code>ha.slave_read_timeout</code> .	String	yes	no
<code>ha.slave_only</code>	Whether this instance should only participate as slave in cluster. If set to <code>true</code> , it will never be elected as master.	String	yes	no
<code>ha.slave_read_timeout</code>	How long a slave will wait for response from master before giving up.	String	yes	no
<code>ha.tx_push_factor</code>	The amount of slaves the master will ask to replicate a committed transaction.	String	yes	no
<code>ha.tx_push_strategy</code>	Push strategy of a transaction to a slave during commit.	String	yes	no
<code>jmx.port</code>	Configuration attribute	String	yes	no
<code>unsupported.cypher.compiler_tracing</code>	Enable tracing of compilation in cypher.	String	yes	no
<code>unsupported.cypher.runtime</code>	Set this to specify the default runtime for the default language version.	String	yes	no
<code>unsupported.dbms.block_size.array_properties</code>	Specifies the block size for storing arrays. This parameter is only honored when the store is created, otherwise it is ignored. Also note that each block carries a ~10B of overhead so record size on disk will be slightly larger than the configured block size	String	yes	no
<code>unsupported.dbms.block_size.labels</code>	Specifies the block size for storing labels exceeding in-lined space in node record. This parameter is only honored when the store is created, otherwise it is ignored. Also note that each block carries a ~10B of overhead so record size on disk will be slightly larger than the configured block size	String	yes	no
<code>unsupported.dbms.block_size.strings</code>	Specifies the block size for storing strings. This parameter is only honored when the store is created, otherwise it is ignored. Note that each character in a string occupies two bytes, meaning that e.g a block size of 120 will hold a 60 character long string before overflowing into a second block. Also note that each block carries a ~10B of overhead so record size on disk will be slightly larger than the configured block size	String	yes	no

Name	Description	Type	Read	Write
<code>unsupported.dbms.counts_store_rotation_timeout</code>	Maximum time to wait for active transaction completion when rotating counts store	String	yes	no
<code>unsupported.dbms.directories.neo4j_home</code>	Root relative to which directory settings are resolved. This is set in code and should never be configured explicitly.	String	yes	no
<code>unsupported.dbms.disconnect</code>	Disable all protocol connectors.	String	yes	no
<code>unsupported.dbms.edition</code>	Configuration attribute	String	yes	no
<code>unsupported.dbms.ephemeral</code>	Configuration attribute	String	yes	no
<code>unsupported.dbms.gc_monitor_threshold</code>	The amount of time in ms the monitor thread has to be blocked before logging a message it was blocked.	String	yes	no
<code>unsupported.dbms.gc_monitor_wait_time</code>	Amount of time in ms the GC monitor thread will wait before taking another measurement.	String	yes	no
<code>unsupported.dbms.id_generator_fast_rebuild_enabled</code>	Use a quick approach for rebuilding the ID generators. This give quicker recovery time, but will limit the ability to reuse the space of deleted entities.	String	yes	no
<code>unsupported.dbms.kernel_id</code>	An identifier that uniquely identifies this graph database instance within this JVM. Defaults to an auto-generated number depending on how many instance are started in this JVM.	String	yes	no
<code>unsupported.dbms.logs.debug.debug_loggers</code>	Debug log contexts that should output debug level logging	String	yes	no
<code>unsupported.dbms.memory.pagecache.pagesize</code>	Target size for pages of mapped memory. If set to 0, then a reasonable default is chosen, depending on the storage device used.	String	yes	no
<code>unsupported.dbms.report_configuration</code>	Print out the effective Neo4j configuration after startup.	String	yes	no
<code>unsupported.dbms.shutdown_transaction_end_timeout</code>	The maximum amount of time to wait for running transactions to complete before allowing initiated database shutdown to continue	String	yes	no
<code>unsupported.dbms.transaction_start_timeout</code>	The maximum amount of time to wait for the database to become available, when starting a new transaction.	String	yes	no
<code>unsupported.ha.cluster_name</code>	The name of a cluster.	String	yes	no
<code>unsupported.tools.batch_inserter.batch_size</code>	Specifies number of operations that batch inserter will try to group into one batch before flushing data into underlying storage.	String	yes	no

Table 142. MBean Diagnostics (*org.neo4j.management.Diagnostics*) Attributes

Name	Description	Type	Read	Write
<i>Diagnostics provided by Neo4j</i>				
<code>DiagnosticsProviders</code>	A list of the ids for the registered diagnostics providers.	List (java.util.List)	yes	no

Table 143. MBean Diagnostics (*org.neo4j.management.Diagnostics*) Operations

Name	Description	ReturnType	Signature
<code>dumpAll</code>	Dump diagnostics information to JMX	String	(no parameters)
<code>dumpToLog</code>	Dump diagnostics information to the log.	void	(no parameters)

Name	Description	ReturnType	Signature
dumpToLog	Dump diagnostics information to the log.	void	java.lang.String
extract	Operation exposed for management	String	java.lang.String

Table 144. MBean High Availability (org.neo4j.management.HighAvailability) Attributes

Name	Description	Type	Read	Write
<i>Information about an instance participating in a HA cluster</i>				
Alive	Whether this instance is alive or not	boolean	yes	no
Available	Whether this instance is available or not	boolean	yes	no
InstanceId	The identifier used to identify this server in the HA cluster	String	yes	no
InstancesInCluster	Information about all instances in this cluster	org.neo4j.management.ClusterMemberInfo (javadocs/org/neo4j/management/ClusterMemberInfo.html)[] as CompositeData (http://docs.oracle.com/javase/7/docs/api/javax/management/openmbean/CompositeData.html)[]	yes	no
LastCommittedTxId	The latest transaction id present in this instance's store	long	yes	no
LastUpdateTime	The time when the data on this instance was last updated from the master	String	yes	no
Role	The role this instance has in the cluster	String	yes	no

Table 145. MBean High Availability (org.neo4j.management.HighAvailability) Operations

Name	Description	ReturnType	Signature
update	(If this is a slave) Update the database on this instance with the latest transactions from the master	String	(no parameters)

Table 146. MBean Kernel (org.neo4j.jmx.Kernel) Attributes

Name	Description	Type	Read	Write
<i>Information about the Neo4j kernel</i>				
DatabaseName	The name of the mounted database	String	yes	no
KernelStartTime	The time from which this Neo4j instance was in operational mode.	Date (java.util.Date)	yes	no
KernelVersion	The version of Neo4j	String	yes	no
MBeanQuery	An ObjectName that can be used as a query for getting all management beans for this Neo4j instance.	javax.management.ObjectName	yes	no
ReadOnly	Whether this is a read only instance	boolean	yes	no
StoreCreationDate	The time when this Neo4j graph store was created.	Date (java.util.Date)	yes	no
StoreId	An identifier that, together with store creation time, uniquely identifies this Neo4j graph store.	String	yes	no
StoreLogVersion	The current version of the Neo4j store logical log.	long	yes	no

Table 147. MBean Locking (org.neo4j.management.LockManager) Attributes

Name	Description	Type	Read	Write
<i>Information about the Neo4j lock status</i>				
Locks	Information about all locks held by Neo4j	<code>java.util.List<org.neo4j.kernel.info.LockInfo> as CompositeData</code> (http://docs.oracle.com/javase/7/docs/api/javax/management/openmbean/CompositeData.html)[<code>]</code>]	yes	no
NumberOfAvertedDeadlocks	The number of lock sequences that would have lead to a deadlock situation that Neo4j has detected and averted (by throwing <code>DeadlockDetectedException</code>).	<code>long</code>	yes	no

Table 148. MBean Locking (*org.neo4j.management.LockManager*) Operations

Name	Description	ReturnType	Signature
<code>getContendedLocks</code>	<code>getContendedLocks</code>	<code>java.util.List<org.neo4j.kernel.info.LockInfo> as CompositeData</code> (http://docs.oracle.com/javase/7/docs/api/javax/management/openmbean/CompositeData.html)[<code>]</code>]	<code>long</code>

Table 149. MBean Memory Mapping (*org.neo4j.management.MemoryMapping*) Attributes

Name	Description	Type	Read	Write
<i>The status of Neo4j memory mapping</i>				
MemoryPools	Get information about each pool of memory mapped regions from store files with memory mapping enabled	<code>org.neo4j.management.WindowPoolInfo</code> (http://docs.oracle.com/javase/7/docs/api/javax/management/openmbean/CompositeData.html)[<code>]</code>]	yes	no

Table 150. MBean Page cache (*org.neo4j.management.PageCache*) Attributes

Name	Description	Type	Read	Write
<i>Information about the Neo4j page cache. All numbers are counts and sums since the Neo4j instance was started</i>				
BytesRead	Number of bytes read from durable storage.	<code>long</code>	yes	no
BytesWritten	Number of bytes written to durable storage.	<code>long</code>	yes	no
EvictionExceptions	Number of exceptions caught during page eviction. This number should be zero, or at least not growing, in a healthy database. Otherwise it could indicate drive failure, storage space, or permission problems.	<code>long</code>	yes	no
Evictions	Number of page evictions. How many pages have been removed from memory to make room for other pages.	<code>long</code>	yes	no
Faults	Number of page faults. How often requested data was not found in memory and had to be loaded.	<code>long</code>	yes	no
FileMappings	Number of files that have been mapped into the page cache.	<code>long</code>	yes	no
FileUnmappings	Number of files that have been unmapped from the page cache.	<code>long</code>	yes	no
Flushes	Number of page flushes. How many dirty pages have been written to durable storage.	<code>long</code>	yes	no

Name	Description	Type	Read	Write
Pins	Number of page pins. How many pages have been accessed (monitoring must be enabled separately).	long	yes	no



The page pin count metric is disabled by default for performance reasons, in which case the pin count value will always be zero. The page pin count metric can be enabled by adding this line to the `neo4j-wrapper.conf` file: `dbms.jvm.additional=-Dorg.neo4j.io.pagecache.tracing.tracePinUnpin=true`

Table 151. MBean Primitive count (*org.neo4j.jmx.Primitives*) Attributes

Name	Description	Type	Read	Write
<i>Estimates of the numbers of different kinds of Neo4j primitives</i>				
NumberOfNodeIdsInUse	An estimation of the number of nodes used in this Neo4j instance	long	yes	no
NumberOfPropertyIdsInUse	An estimation of the number of properties used in this Neo4j instance	long	yes	no
NumberOfRelationshipIdsInUse	An estimation of the number of relationships used in this Neo4j instance	long	yes	no
NumberOfRelationshipTypeIdsInUse	The number of relationship types used in this Neo4j instance	long	yes	no

Table 152. MBean Store file sizes (*org.neo4j.jmx.StoreFile*) Attributes

Name	Description	Type	Read	Write
<i>Information about the sizes of the different parts of the Neo4j graph store</i>				
ArrayStoreSize	The amount of disk space used to store array properties, in bytes.	long	yes	no
LogicalLogSize	The amount of disk space used by the current Neo4j logical log, in bytes.	long	yes	no
NodeStoreSize	The amount of disk space used to store nodes, in bytes.	long	yes	no
PropertyStoreSize	The amount of disk space used to store properties (excluding string values and array values), in bytes.	long	yes	no
RelationshipStoreSize	The amount of disk space used to store relationships, in bytes.	long	yes	no
StringStoreSize	The amount of disk space used to store string properties, in bytes.	long	yes	no
TotalStoreSize	The total disk space used by this Neo4j instance, in bytes.	long	yes	no

Table 153. MBean Transactions (*org.neo4j.management.TransactionManager*) Attributes

Name	Description	Type	Read	Write
<i>Information about the Neo4j transaction manager</i>				
LastCommittedTxId	The id of the latest committed transaction	long	yes	no
NumberOfCommittedTransactions	The total number of committed transactions	long	yes	no
NumberOfOpenedTransactions	The total number started transactions	long	yes	no
NumberOfOpenTransactions	The number of currently open transactions	long	yes	no

Name	Description	Type	Read	Write
NumberOfRolledBackTransactions	The total number of rolled back transactions	long	yes	no
PeakNumberOfConcurrentTransactions	The highest number of transactions ever opened concurrently	long	yes	no

Table 154. MBean Index sampler (org.neo4j.management.IndexSamplingManager) Operations

Name	Description	ReturnType	Signature
triggerIndexSampling	triggerIndexSampling	void	java.lang.String,java.lang.String,boolean

8.3. Available metrics

Table 155. 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 156. 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 157. 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 158. Database Transaction Metrics

Name	Description
neo4j.transaction.started	The total number of started transactions

Name	Description
<code>neo4j.transaction.peak_concurrent</code>	The highest peak of concurrent transactions ever seen on this machine
<code>neo4j.transaction.active</code>	The number of currently active transactions
<code>neo4j.transaction.active_read</code>	The number of currently active read transactions
<code>neo4j.transaction.active_write</code>	The number of currently active write transactions
<code>neo4j.transaction.committed</code>	The total number of committed transactions
<code>neo4j.transaction.committed_read</code>	The total number of committed read transactions
<code>neo4j.transaction.committed_write</code>	The total number of committed write transactions
<code>neo4j.transaction.rollbackbacks</code>	The total number of rolled back transactions
<code>neo4j.transaction.rollbackbacks_read</code>	The total number of rolled back read transactions
<code>neo4j.transaction.rollbackbacks_write</code>	The total number of rolled back write transactions
<code>neo4j.transaction.terminated</code>	The total number of terminated transactions
<code>neo4j.transaction.terminated_read</code>	The total number of terminated read transactions
<code>neo4j.transaction.terminated_write</code>	The total number of terminated write transactions
<code>neo4j.transaction.last_committed_tx_id</code>	The ID of the last committed transaction
<code>neo4j.transaction.last_closed_tx_id</code>	The ID of the last closed transaction

Table 159. Cypher Metrics

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

Table 160. Database LogRotation Metrics

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

Table 161. Network Metrics

Name	Description
<code>neo4j.network.slave_network_tx_writes</code>	The amount of bytes transmitted on the network containing the transaction data from a slave to the master in order to be committed
<code>neo4j.network.master_network_store_writes</code>	The amount of bytes transmitted on the network while copying stores from a machines to another

Name	Description
<code>neo4j.network.master_network_tx_writes</code>	The amount of bytes transmitted on the network containing the transaction data from a master to the slaves in order to propagate committed transactions

Table 162. Cluster Metrics

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

8.3.1. 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 9. License

Creative Commons 3.0

You are free to

Share

copy and redistribute the material in any medium or format

Adapt

remix, transform, and build upon the material

for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms

Attribution

You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

ShareAlike

If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

No additional restrictions

You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

Notices

You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation.

No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material.

See <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>.