Neo4j Upgrade and Migration Guide

Table of Contents

Chapter 1. About this guide

Keeping your Neo4j deployment always up-to-date ensures that you are provided with the latest improvements in performance, security, and bug fixes.

Who should read this?

This upgrade and migration guide is written for experienced system administrators and operations engineers who want to upgrade or migrate Neo4j.

This page introduces some important Neo4j concepts before referring to the version-specific pages.

Chapter 2. Preparation

Preparation is key to any successful upgrade or migration. Before making changes to a production DBMS, it is highly recommended to use a test environment to check:

- The upgrade/migration process.
- Compatibility with other systems.

Chapter 3. Version numbers

Neo4j version numbers are in the pattern `MAJOR.MINOR.PATCH`.

- `MAJOR` versions introduce significant architectural improvements and features. They are not compatible with previous `MAJOR` versions. Systems that interact with the database may require updating.
- `MINOR` versions introduce improvements and new features. They are backward compatible with other `MINOR` versions of the `MAJOR` version.
- `PATCH` versions fix critical bugs and security issues. They are backward compatible and replace previous releases of the same `MAJOR.MINOR` version.

Neo4j's fully managed cloud service Neo4j Aura uses only `MAJOR` versioning and is always on the latest `MINOR` version.

Chapter 4. Downtime, Store formats, and downgrades

4.1. Downtime

- `MAJOR` version migrations require downtime.
- `MINOR` and `PATCH` upgrades can be applied to a cluster without downtime.
- Standalone servers require downtime to upgrade.

When you move to a new *major* version of Neo4j, you must migrate the databases from the old server to the new server. Servers are upgraded by updating their binaries and restarting.

4.2. Store format

- Store format updates are optional unless you are moving to a new `MAJOR` version that removes support for your old store format. For more information on the available store formats per Neo4j version, see Operations Manual → Display store information.

4.3. Downgrades

Neo4j does not support downgrades. If the upgrade or migration is not successful, you have to do a full rollback, including restoring a pre-upgrade or a pre-migration backup.

Chapter 5. Continue reading

If you are already on Neo4j 5, or want to migrate your databases from 4.4 you can proceed to the Neo4j 5 section.

If you are upgrading to a version of Neo4j 4, read the Neo4j 4 section.

Chapter 6. Neo4j 5 upgrade and migration

It is recommended to read the Introduction section before continuing.

6.1. Upgrading Neo4j 5

You can upgrade from any version of Neo4j 5 to any subsequent `MINOR` or `PATCH` version, by following the instructions in Upgrade or patch Neo4j 5.

6.2. Migrating databases from Neo4j 4.4

We do not recommend you upgrade a Neo4j 4.4 server to 5. Instead, you should configure a new Neo4j server and migrate the databases from the old server to the new server.

If you are using Neo4j 4.4, you can migrate your databases to Neo4j 5 by following the instructions in Migrate from 4.4.

Otherwise, you must first migrate your databases to Neo4j 4.4 by following the instructions in the Neo4j 4 specific section.

6.3. Migrating databases from Neo4j Community Edition to Neo4j Enterprise Edition

You can upgrade from Neo4j 5 Community Edition to Neo4j 5 Enterprise Edition using the following steps. If the information stored in the `system` database is not required when migrating, you can skip the steps associated with the `system` database.

1. Stop the running Neo4j 5 Community Edition server:

```
neo4j stop
```

2. Backup the `system` and `neo4j` databases using the `neo4j-admin database dump` command:

```
neo4j-admin database dump neo4j
```

```
neo4j-admin database dump system
```

By default, dumps are located to *data/dumps/*. Alternatively, the location can be defined via the *conf/neo4j.conf* setting `server.directories.dumps.root`.

3. Load the `system` and `neo4j` databases into the Neo4j 5 Enterprise Edition server using `neo4j-admin database load`:

```
neo4j-admin database load neo4j --from-path=<path-to-community-neo4j-dump>
```

```
neo4j-admin database load system --from-path=<path-to-community-system-dump>
```

4. Navigate to the Neo4j 5 Enterprise Edition server and start it:

```
neo4j start
```

5. If you are using 5.8 or before of Enterprise Edition, you need to manually upgrade the `system` database. Connect to the server and run the following procedure:

```
CALL dbms.upgrade();
```

   5.9 onwards automatically does the upgrade when it is safe to do so.

6. Drop the existing `neo4j` database that was created by default on startup. Note, this does not remove any store files loaded via `neo4j-admin database load`.

```
DROP DATABASE neo4j;
```

7. Create the `neo4j` database using the store files loaded via `neo4j-admin database load`:

```
CREATE DATABASE neo4j;
```

6.4. Upgrade or patch Neo4j 5

This page is an overview of what needs to be checked and completed to ensure your upgrade goes smoothly and as planned.

It is recommended to read the Introduction before continuing.

6.4.1. Review changes

Review the Release Notes to see what has been introduced or changed in the new version.

6.4.2. Standalone

The steps to upgrade a standalone Neo4j server (one that's not part of a cluster) are:

1. Stop the old Neo4j server.
2. Install the new version of Neo4j.
3. Start the new Neo4j server.

We recommend the use of a package manager to install Neo4j 5. If you use a package manager then an upgrade can be performed with one command:

```
(sudo) yum update neo4j
```

When you restart the Neo4j server, it is a good idea to monitor the logs for any errors or warnings caused by the upgrade. The neo4j.log file contains information on the upgrade.

Follow the links to some walkthroughs of upgrading a range of Neo4j standalone installations:

- Debian package manager
- Docker
- RPM package manager
- TAR/ZIP

6.4.3. Clusters

There are two strategies to upgrade a basic cluster:

- **New server rolling upgrade** - A new server is added before an existing server is taken offline. This approach keeps the cluster available and fault-tolerant throughout the upgrade. This approach is suitable for deployments that use replaceable cloud or container resources.
- **In-place rolling upgrade** - Each server is taken offline and upgraded one at a time. The server is reconnected to the cluster before the next is taken offline. This approach keeps the cluster available throughout the upgrade. However, it does reduce fault tolerance while each server is offline.

Should the data stores fall out of sync during the process, you may need to re-seed the cluster. For more information, see Operations Manual → Seed a cluster.

Follow the links to some walkthroughs of upgrading a range of Neo4j cluster installations:

- New server rolling upgrade
- In-place rolling upgrade

To upgrade a analytics cluster, that is, a cluster with a single primary for `system` and one or more `system` secondaries (using the setting `server.cluster.system_database_mode`), it is important to upgrade the secondaries before you upgrade the primary. If not, the primary automatically updates the `system` database, and the secondaries are no longer compatible. Similarly to in-place rolling upgrades, the steps for an individual server are identical to upgrading a standalone server.

Upgrade a standalone server with a Debian package manager

An example of how to upgrade or patch a 5.x standalone server using a Debian package manager.

It is recommended to read the following pages before continuing:

- Introduction
- Upgrade or patch Neo4j 5

For critical systems, it is recommended to:

- Backup your server(s), including databases, configuration, and encryption keys to avoid losing data in case of failure.
- Perform the upgrade in a test environment.

Upgrade

Upgrade your Neo4j deployment by running:

```
sudo apt upgrade neo4j-enterprise
```

If you are upgrading Community Edition, replace `neo4j-enterprise` with `neo4j`.

Monitor the logs

When Neo4j restarts, it can be useful to monitor the logs for any errors or warnings caused by the upgrade. You can find information about the upgrade in the neo4j.log file.

Further information

- For more information on how to configure the Neo4j `apt` repository, see Operations Manual → Debian.
- For more information about the supported package and service managers, see Operations Manual → Linux installation.

Upgrade a standalone server (Docker)

This example shows how to upgrade or patch a standalone server version 5 running on Docker.

It is recommended to read the following pages before continuing:

- Introduction
- Upgrade or patch Neo4j 5

For critical systems, it is recommended to:

- Backup your server(s), including databases, configuration, and encryption keys to avoid losing data in case of failure.
- Perform the upgrade in a test environment.

The following steps assume that Neo4j DBMS 5.x.a Enterprise Edition is running in a Docker container named `neo4j-5.x.a`, with the */data directory* in the container mapped to the */v5/data* on the host machine as in the following example:

```
docker run -d \
    --name=neo4j-5.x.a \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=/path/to/v5/data:/data \
    --env=NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \ ①
    neo4j:5.x.a-enterprise
```

① Only for Enterprise Edition.

If you want to monitor the logs, you should also map the container's */logs* directory to a */logs* directory on the host machine.

Upgrade steps

1. Stop the container `neo4j-5.x.a` by running:

   ```
   docker stop neo4j-5.x.a
   ```

2. Remove the `neo4j-5.x.a` container:

   ```
   docker rm neo4j-5.x.a
   ```

3. Start a 5.y.b container with the */data* directory in the container mapped to the same */v5/data* on the host machine:

   ```
   docker run -d \
       --name=neo4j-5.y.b \
       --publish=7474:7474 --publish=7687:7687 \
       --volume=/path/to/v5/data:/data \
       --env=NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
       neo4j:5.y.b-enterprise
   ```

   The upgrade process takes place during startup.

Monitor the logs

The neo4j.log file contains information on how many steps the upgrade will involve and how far it has progressed.

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

Upgrade a standalone server with an RPM package manager

An example of how to upgrade or patch a 5.x standalone server using an RPM package manager.

> It is recommended to read the following pages before continuing:
>
> * Introduction
> * Upgrade or patch Neo4j 5
>
> For critical systems, it is recommended to:
>
> * Backup your server(s), including databases, configuration, and encryption keys to avoid losing data in case of failure.
> * Perform the upgrade in a test environment.

Upgrade

Upgrade your Neo4j deployment by running:

```
sudo yum update neo4j-enterprise
```

> If you are upgrading Community Edition, replace `neo4j-enterprise` with `neo4j`.

Monitor the logs

When Neo4j restarts, it can be useful to monitor the logs for any errors or warnings caused by the upgrade. You can find information about the upgrade in the neo4j.log file.

Further information

* The Neo4j RPM `yum` repository is configured as described in Operations Manual → Deploy Neo4j using the Neo4j RPM package.
* For more information about the supported package and service managers, see Operations Manual → Linux installation.

Upgrade a standalone server (TAR/ZIP)

This example shows how to upgrade or patch a 5.x standalone server from a TAR/ZIP. We recommend the use of Redhat or Debian package managers to install/upgrade Neo4j 5 because it simplifies this process.

It is recommended to read the following pages before continuing:

- Introduction
- Upgrade or patch Neo4j 5

For critical systems, it is recommended to:

- Backup your server(s), including databases, configuration, and encryption keys to avoid losing data in case of failure.
- Perform the upgrade in a test environment.

The following example steps assume that the Neo4j 5.x has its binaries in its default location, while the */data* and */conf* folders are configured using the `server.directories.data` configuration setting and the environment variable `NEO4J_CONF`. This means you can replace the DBMS binaries without affecting the configuration and state.

You must be careful to not overwrite your data or config directories when you install the new version.

Upgrade steps

1. Stop Neo4j 5.x.

```
/neo4j-enterprise-5.x/bin/neo4j stop
```

2. From Neo4j Download Center, download latest version, 5.y TAR or ZIP and unpack it.
3. Start Neo4j 5.y:

```
/neo4j-enterprise-5.Y/bin/neo4j start
```

Monitor the logs

The *neo4j.log* file contains information on how many steps the upgrade will involve and how far it has progressed. It is advisable to monitor this log.

In-place rolling upgrade of a cluster

It is recommended to read the Introduction and Upgrade or patch 5.X before continuing.

This is an example of how to do a rolling upgrade, by upgrading individual servers one at a time. This approach keeps the cluster available throughout the upgrade. However, it does reduce fault tolerance while each server is offline.

Before you begin

Perform these steps in full on each server, before moving on to the next server. You must ensure that the cluster returns to a stable state before moving on to the next.

Upgrade steps (to be repeated for each server)

1. Check the cluster is healthy.

   a. Use the following query to check servers are hosting all their assigned databases. The query should return no results:

   ```
   SHOW SERVERS YIELD name, hosting, requestedHosting, serverId WHERE requestedHosting <> hosting
   ```

   > Up to version 5.9, if you have composite databases, they will not appear in `requestedHosting`, and so all servers are returned by this query. Thus, you must manually compare the `hosting` and `requestedHosting` fields, excluding any composite databases. From version 5.10 onwards, composite databases appear in both `hosting` and `requestedHosting` fields.

   b. Use the following query to check all databases are in their expected state. The query should return no results:

   ```
   SHOW DATABASES YIELD name, address, currentStatus, requestedStatus, statusMessage WHERE
   currentStatus <> requestedStatus RETURN name, address, currentStatus, requestedStatus,
   statusMessage
   ```

2. For in-place rolling upgrades, the steps for an individual server are identical to upgrading a standalone server. The exact steps depend on how you have chosen to deploy Neo4j.

   Examples are available for common deployments:

   - Debian package manager
   - Docker
   - RPM package manager
   - TAR/ZIP

3. Once the server has been restarted, confirm the server is running successfully.

   Run the following command and check the server has state `Enabled` and health `Available`.

   ```
   SHOW SERVERS WHERE name = [server-id];
   ```

4. Confirm that the server has started all the databases that it should.

   This command shows any databases that are not in their expected state:

   ```
   SHOW DATABASES YIELD name, address, currentStatus, requestedStatus, serverID WHERE currentStatus <>
   requestedStatus AND serverID = [server-id] RETURN name, address, currentStatus, requestedStatus
   ```

5. Repeat these steps on the next server, until all servers are upgraded.

Completion

If you are using version 5.8 or before, you need to manually upgrade the `system` database. If you are using version 5.9 or later, you can skip this step, the `system` database is automatically upgraded when safe to do so. Run the following Cypher against the cluster, using Cypher Shell or another Cypher client.

```
CALL dbms.upgrade()
```

Monitor the logs

When Neo4j restarts, it is a good idea to monitor the logs for any errors or warnings caused by the upgrade. The neo4j.log file contains information on the upgrade.

New server rolling upgrade of a cluster

It is recommended to read the Introduction and Upgrade or patch 5.X before continuing.

This is an example of how to do a rolling upgrade without reducing fault tolerance, by adding servers before removing old ones. This approach is suitable for deployments that use replaceable cloud or container resources.

Before you begin

1. Ensure that the databases cannot be stopped, created, or dropped during the rolling upgrade by using the following command:

   ```
   DENY STOP ON DATABASE * TO PUBLIC
   DENY DATABASE MANAGEMENT ON DBMS TO PUBLIC
   ```

2. Verify that all databases are online. See Operations Manual → Start databases for details.

Upgrade steps (to be repeated for each server)

Perform these steps in full for each server, before moving on to the next server. You must ensure that the cluster returns to a stable state before moving on to the next.

1. Check the cluster is healthy.

   a. Before replacing the server, ensure that the cluster is in a healthy state. Use the following query to check servers are hosting all their assigned databases. The query should return no results:

   ```
   SHOW SERVERS YIELD name, hosting, requestedHosting, serverId WHERE requestedHosting <> hosting
   ```

   > Up to version 5.9, if you have composite databases, they will not appear in `requestedHosting`, and so all servers are returned by this query. Thus, you must manually compare the `hosting` and `requestedHosting` fields, excluding any composite databases. From version 5.10 onwards, composite databases appear in both `hosting` and `requestedHosting` fields.

   b. Use the following query to check all databases are in their expected state. The query should return no results:

   ```
   SHOW DATABASES YIELD name, address, currentStatus, requestedStatus, statusMessage WHERE
   currentStatus <> requestedStatus RETURN name, address, currentStatus, requestedStatus,
   statusMessage
   ```

2. Create the new server.

3. Add the new server to the cluster:

   a. Start the new server and wait for it to be visible to the cluster.

   b. Use `SHOW SERVERS` to see the new server, it should have the status `Free`. Make a note of the new server's ID.

   c. Add the server to the DBMS by running `ENABLE SERVER [new-server-id]`. This allows the DBMS to use the server to host databases.

4. Prepare a server for removal:

   a. Run `SHOW SERVERS` to retrieve the ID of one of the old servers. The server must be `Available`.

   b. Tell Neo4j to move any databases off the old server by running `DEALLOCATE DATABASES FROM SERVER [old-server-id]`.

5. Wait for all databases to be deallocated. You can use `SHOW SERVER [old-server-id]` to follow the process. Once the databases complete their move, the `hosting` column will contain only `system`.

   > It can take time for a database to start on its new host and provide fault tolerance. If you stop the old server before this point, you reduce the fault tolerance of the database.

6. Remove the old server:

   a. Remove the old server by running `DROP SERVER [old-server-id]`. It will still appear as `Dropped` in `SHOW SERVERS` until the process stops.

   b. Server can then be shut down.

7. (Optional) Restore the privilege of the `PUBLIC` role to stop databases:

```
REVOKE DENY STOP ON DATABASE * FROM PUBLIC
```

8. (Optional) Restore the privilege of the `PUBLIC` role to create and drop databases:

```
REVOKE DENY DATABASE MANAGEMENT ON DBMS FROM PUBLIC
```

9. (Optional) If you have started offline databases during the preparation phase for a rolling upgrade, you will need to stop each of them in order to restore them to the original state.
10. If using the `LIST` discovery method, after you upgrade all the servers make sure to set `dbms.cluster.discovery.endpoints` to include only the new servers.

    See also `dbms.cluster.discovery.resolver_type`.

11. Complete the upgrade.

    If you are using version 5.8 or before, you need to manually upgrade the `system` database. If you are using version 5.9 or later, you can skip this step, the `system` database is automatically upgraded when safe to do so. Run the following Cypher against the cluster using Cypher Shell or another Cypher client:

```
CALL dbms.upgrade()
```

Monitor the logs

When Neo4j restarts, it is a good idea to monitor the logs for any errors or warnings caused by the upgrade. The neo4j.log file contains information on the upgrade.

6.5. Migrate databases from 4.4

Databases can only be migrated to Neo4j 5 from Neo4j 4.4.

It is recommended to read the Introduction section before continuing.

To migrate databases from Neo4j 4.4 you must first configure a new Neo4j 5 DBMS, whether standalone or in a cluster. The databases hosted on Neo4j 4.4 can then be moved to the new Neo4j 5 DBMS.

Depending on your environment, this could require substantial preparation and planning. The steps have broken down into four stages, as outlined below:

1. **Prepare** - Changes between Neo4j 4.4 and Neo4j 5 can have implications for systems and users relying on Neo4j. Additionally, the migration process itself will require some downtime. See Prepare for your migration for details.
2. **Backup** - Migrating your 4.4 databases requires that you have a dump or a backup of every database you want to migrate to Neo4j 5. See Backup your databases for details.
3. **Install and configure Neo4j 5** - There are changes to how Neo4j is configured, particularly the new

autonomous clusters. See Install and configure Neo4j 5 for details. Roles and privileges must also be copied over.

4. **Migrate databases** - Once affected systems are prepared, and Neo4j 5 is correctly configured, you are ready to migrate the databases. See Migrate your databases for details.

The Reference section contains:

- Additional resources that are referred to throughout the guide.
- A guide specific to migrating a standalone server deployed on Kubernetes with the *neo4j/neo4j-standalone* Helm chart from version 4.4 to 5.x; see Migrate a standalone server (Helm).

6.5.1. Prepare for your migration

Major versions introduce significant improvements and are not compatible with previous MAJOR versions. Systems that interact with the database will probably need updating. And downtime is always necessary.

Review changes

Start by reviewing the new features, fixes, and breaking changes. As you do, make a list of your external systems that connect to Neo4j that will need updating. In particular,

- The Release Notes describe what's been introduced or changed in the new version.
- The Breaking changes page is a consolidated list of all breaking changes between Neo4j 4.x and Neo4j 5.x.
- See Cypher Manual → Deprecations, additions and compatibility for changes in the Cypher language.

Downtime

Migrating databases between versions of Neo4j always requires downtime. Although it is possible to leave the old versions online in read-only mode.

Since every case is different, it is recommended that you prepare an environment in which you can test the migration process. This will give you more accurate timings than you can estimate and an idea of what to expect in terms of duration for the process.

The time to migrate a database will depend on the size of the database because each database must be backed up and restored. There're no file format changes between 4.4 and 5 that require lengthy transformations operations.

Disk space considerations

To migrate a database you must create a backup (or dump-file for Community Edition). The backup/dump-file will require the same amount of disk space as the database it is created from. For this reason, you should reserve at least the size of the database directory in free space before you begin.

Prepare indexes

In Neo4j 5, BTREE indexes are replaced by RANGE, POINT, and TEXT indexes. It is good to create a matching RANGE, POINT, or TEXT index for each BTREE index and index-backed constraint and populate them in Neo4j 4.4 before migration. This helps to avoid any performance impairment when Neo4j 5 drops the BTREE indexes during the migration.

Do not remove the old BTREE indexes. They must co-exist with the newly created ones in Neo4j 4.4, as the new indexes cannot be used by your 4.4 installation. The old indexes will be dropped automatically during the migration.

Choosing a replacement index

In most cases, RANGE indexes can replace BTREE. However, there might be occasions when a different index type is more suitable, such as:

- Use POINT indexes if the property value type is `point` and `distance` or `bounding box` queries are used for the property.
- Use TEXT indexes if the property value type is `text` and the values can be larger than 8Kb.
- Use TEXT indexes if the property value type is `text` and `CONTAINS` and `ENDS WITH` are used in queries for the property.

It is possible to have multiple indexes of different index types on the property.

Creating an index

You can run `SHOW INDEXES` on your Neo4j 4.4 database to display its indexes. For each index and index-backed constraint, create an **additional** index or constraint of type RANGE, POINT or TEXT.

The following code block shows how to create a RANGE index:

```
CREATE RANGE INDEX range_index_name FOR (n:Label) ON (n.prop1)
```

An example of how to create a constraint backed by a RANGE index:

```
CREATE CONSTRAINT constraint_with_provider FOR (n:Label) REQUIRE (n.prop1) IS UNIQUE OPTIONS
{indexProvider: 'range-1.0'}
```

During database migration, it is possible to force Neo4j 5 to drop all BTREE indexes and replace them with RANGE indexes using the option `--force-btree-indexes-to-range` of the `neo4j-admin database migrate` command. However, this is not advised because RANGE indexes are not always the optimal replacement, and performance will be impaired on start-up while the new indexes are populated.

For more information about creating indexes, see Cypher Manual → Creating indexes.

Application code

Depending on how your application interacts with Neo4j, you should be prepared to review your application code. All breaking changes that may affect your application and Neo4j can be found in Breaking changes between Neo4j 4.4 and Neo4j 5.x.

Drivers

Neo4j 4.4 drivers are forward compatible with Neo4j 5, and the Neo4j 5 drivers are backward compatible with Neo4j 4.4. This means that you can migrate your app using your existing drivers to a Neo4j 5 first, and then upgrade your drivers or vice versa. Note that you can access new Neo4j 5 database features only if you are using the latest Neo4j 5 drivers.

Autonomous cluster vs Causal cluster

Neo4j's clustering has been significantly improved in v5, and the old causal clusters are no longer supported. If you are configuring a cluster, please read the clustering chapter in the Operations Manual to familiarise yourself with the new concepts: Operations Manual → Clustering.

Plugins (including custom plugins)

Take note of the plugins you are using and make sure they are compatible with Neo4j 5.x. If you have developed any custom plugins, you should review them as you would your application code.

If you are using Neo4j Bloom or Graph Data Science, you can find the most recent versions of these products in the Neo4j Download Center.

If you are using APOC, see the APOC Migration Guide for guidance.

Other 3rd-party software and tools

Be mindful of any other 3rd-party software and tools you are using alongside Neo4j. Maybe you have leveraged operational scripts to install, manage, backup, or monitor your Neo4j deployment. You might also have set alarms and built complete monitoring dashboards. You need to revise these as there have been changes in the Neo4j Admin tool, backup and restore, clustering, configurations, etc. Therefore, it is recommended to review all scripts/tools/3rd-party software and ensure they are prepared and compatible with Neo4j 5.x.

6.5.2. Backup your databases

The following steps assume that:

- You have one or more databases hosted on Neo4j 4.4.
- That you want to migrate those databases when upgrading to Neo4j 5.

Prepare the databases for migration

Confirm that BTREE indexes have RANGE, POINT, or TEXT equivalents

It is recommended that you create new RANGE, POINT, or TEXT indexes for each of your BTREE indexes and index-backed constraints. See the instructions in Prepare indexes.

Make your databases read-only

To ensure the databases do not get updated during the backup, put them into read-only mode using Cypher:

```
ALTER DATABASE <databasename> SET ACCESS READ ONLY
```

Create a backup of each database

You can back up all databases from a single 4.4 cluster member.

> If you are migrating databases from an installation of Neo4j 4.4 *Community Edition*, you will need to take the database offline and perform a backup using the `neo4j-admin dump` command.
>
> For more information, see Operations Manual 4.4 → Back up an offline database.

If you are backing up a database from a cluster, run the Cypher command `SHOW DATABASES YIELD *`, and choose a server that is up-to-date with the last committed transaction on all databases as a backup source.

1. Create a directory to store backups. These steps use */migration-backups*.
2. Run the `neo4j-admin backup` command to back up **each of your databases**.

Use the option `--include-metadata=all` to include all roles and users associated with each of your databases.

```
/usr/bin/neo4j-admin backup --database=<databasename>  --backup-dir=/migration-backups --include-metadata
=all
```

Ensure that you have successfully backed up all your databases. The result is a folder for each database, called *<databasename>* and located in the */migration-backups* folder, and a metadata script for each database, located in *migration-backups/databasename/tools/metadata_script.cypher*.

For more information about the `neo4j-admin backup` command, see Operations Manual 4.4 → Back up an online database.

6.5.3. Install and configure Neo4j 5

There are changes to how Neo4j is configured. This section discusses how to configure Neo4j 5 in the same way as your Neo4j 4.4 DBMS.

Install Neo4j 5

For the initial installation, refer to the Neo4j 5 Operations Manual for methods of installing Neo4j.

> ⚠ If you have one or more existing databases created with earlier versions of Neo4j, you must back these up **before** installing 5. For more information, see Backup your databases. This is **especially** important if you install Neo4j via Debian or RPM packages.

Also note that:

- You must remove the default database if you are migrating a database with the same name.
- Neo4j 5.x runs on Java 17, and from Neo4j 5.14, it also supports Java 21.

  If you have other Java applications running on the machine hosting Neo4j, ensure those applications are compatible with the Java version that your Neo4j is running on. Alternatively, you should configure it to run multiple JDKs on the same machine.

Using Debian or RPM packages

If you install Neo4j 5 via Debian or RPM packages, the Neo4j binaries are upgraded "in-place". This means that your 5 installation will overwrite your 4.4 installation.

During the upgrade process, the package manager will warn you to take backups before continuing. Moreover, the upgrade creates a folder `${NEO4J_HOME}/data-unmigrated-from-4.4` with all 4.4 databases and metadata in raw format (not backups or dumps).

If you have already taken valid dumps or backups of every database you want to migrate to Neo4j 5, you should remove this generated folder and its contents to ensure you have sufficient disk space for the restored and migrated copies produced later in the upgrade process (see Migrate your databases).

If you do **not** possess such dumps or backups you must perform the following steps:

1. Reinstate your previous Neo4j 4.4 installation: Operations Manual → Linux installation.
2. Rename the previously generated unmigrated data folder:

   ```
   mv ${NEO4J_HOME}/data-unmigrated-from-4.4 ${NEO4j_HOME}/data
   ```

3. Start this earlier version of Neo4j:

   ```
   $NEO4J_HOME/bin/neo4j start
   ```

4. Take backups of all databases you want to migrate to Neo4j 5 using the provided instructions in Backup your databases.
5. Stop Neo4j and re-attempt the upgrade.

Migrate your configuration

Neo4j 5 changes many configuration settings. For the complete list of changes, see Changes to configuration settings in Neo4j 5.

Neo4j Admin has a utility that can convert most of your 4.4 settings to 5 equivalents. For more information, see the Operations Manual → Migrate the Neo4j configuration file. The changes to clustering are such that it is recommended to configure those from scratch.

> As mentioned in Prepare for your migration, Neo4j clustering has changed **significantly** in v5, with the introduction of Autonomous clusters. While it is technically possible to automatically migrate the configuration file for a Neo4j cluster member, not all clustering settings can be directly translated. Given the scope of the changes, it is recommended that you *configure your new cluster from scratch*.

Steps for migrating 4.4 configuration files

1. Copy the Neo4j 4.4 configuration file(s) to the new Neo4j 5 configuration directory if you want to migrate an existing configuration, and have concurrent Neo4j 4.4 and 5 installations (e.g., if installing using the tarball executable).

   The default location that Neo4j looks for configuration files depends on how Neo4j is installed:

   - Operations Manual → Default file locations in 4.4.
   - Operations Manual → Default file locations in 5.

2. Use the `neo4j-admin server migrate-configuration` command in Neo4j 5 to migrate the configuration file to a 5.x-compatible format:

   ```
   $NEO4J_5_HOME/bin/neo4j-admin server migrate-configuration
   ```

3. Check the report written to the console for messages related to settings that could not be converted.

Configure roles and privileges

The role-based access control (RBAC) is unchanged in 5 and you can re-create your previous configuration. For more information on roles and privileges, see Operations Manual → Authentication and authorization.

List all configured roles and recreate them

Run the following Cypher command on Neo4j 4.4 to list the configured roles:

```
SHOW ROLES
```

Then, recreate them on Neo4j 5 using:

```
CREATE ROLE <new_role>
```

List all privileges and recreate them

Run the following Cypher on Neo4j 4.4 to list the privileges assigned to roles:

```
SHOW PRIVILEGES AS COMMANDS
```

Then, use the output to recreate the privileges on Neo4j 5.

⚠	You cannot create privileges related to specific databases or graphs that do not exist yet on your new Neo4j 5 DBMS. These privileges can be restored as part of the database migration steps, see Restore roles and privileges.

Aliases

If you are using aliases, you must recreate them manually. For more information on aliases, see Operations Manual → Managing database aliases for standard databases.

Performance metrics

- All metric names now include `dbms` or `database` namespaces, and the setting `metrics.namespaces.enabled` is removed.

  A complete list of metrics is available in Operations Manual → Metrics reference

- All settings to enable and disable a metric type (`metrics.*.enabled`) are removed. They are replaced by `server.metrics.filter`, which takes a regex of what metrics to enable.

Certificates

Copy all the files used for encryption, such as private key, public certificate, and the contents of the trusted and revoked directories over to Neo4j 5.

Plugins

If you are using custom plugins, make sure they are compatible with Neo4j 5 and Java 17, and place them in the */plugins* directory.

Enterprise Edition includes APOC Core, Bloom, and Graph Data Science. The latest versions of these plugins are also available at the Neo4j Download Center.

6.5.4. Migrate your databases

The following steps assume that:

* You have taken backups of any Neo4j 4.4 databases you want to migrate.
* You have installed an equivalent Neo4j 5 DBMS to move the databases to.

Switching to Neo4j 5

If your method of installation (e.g. tarball or docker) allows you to have Neo4j 4.4 and 5 installations concurrently on the same server, you may now shut down your Neo4j 4.4 process. Alternatively, you may prefer to leave it running, with all its databases in read-only mode until the migration is complete.

If you do choose to run two Neo4j installations concurrently, ensure that your server has sufficient resources to support this and that your configuration for the two installations does not conflict (e.g., on ports).

From this point on, you are on the Neo4j 5 DBMS. If both versions are on the same machine, you may need to change the default Java runtime to Java 17. From version 5.14, Neo4j also supports Java 21.

Make sure Neo4j 5 does not already host any databases with the same name as those that you want to migrate (ignoring the `system` database).

Restore the database on Neo4j 5

You can now restore the database backups to Neo4j 5.

If you are migrating to a cluster, only restore them to one server; later you will use this server to seed the others.

If you are migrating databases from an installation of Neo4j 4.4 *Community Edition*, you need to restore your dump file using the `neo4j-admin database load` command.

For more information, see Operations Manual 5 → Restore a database dump.

1. Use the `neo4j-admin restore` command to restore each of your databases, except the `system` database:

   ```
   /usr/bin/neo4j-admin database restore <databasename> --from-path=/migration-backups/<databasename>
   ```

2. Use the `neo4j-admin database migrate` command to update the file metadata to the 5 format:

   ```
   /usr/bin/neo4j-admin database migrate <databasename>
   ```

The command will fail if you do not have RANGE, POINT, or TEXT indexes for each of your BTREE indexes and index-backed constraints. You can use the option `--force-btree-indexes-to-range` to force `neo4j-admin database migrate` to drop the BTREE indexes and replace them with RANGE indexes. It is advisable to read the section on preparing indexes from Prepare for your migration to understand the consequences.

If you also want to compress your databases when moving to 5.x, store copy can be used as an alternative to migrate a database between 4.4 and 5.x.

For more information, see Operations Manual 5 → Copy a database store.

Recreate the databases

Start Neo4j 5 and recreate each of your databases using the following Cypher:

```
CREATE DATABASE <databasename>
```

1. Start Neo4j 5 and retrieve the server ID for the server where you restored the backups by running:

   ```
   SHOW SERVERS YIELD address, serverId
   ```

   When you create a database on a Neo4j 5 cluster you can optionally use the `TOPOLOGY` parameter to specify how many primary and secondary copies are required. These values can be changed later using `ALTER DATABASE`.

   For more information, see Operations Manual 5 → Create Database.

2. Recreate each of your databases using the following Cypher:

   ```
   CREATE DATABASE <databasename>
   TOPOLOGY [desired number of primaries] PRIMARIES [desired number of secondaries] SECONDARIES
   OPTIONS {existingData: 'use', existingDataSeedInstance: '[ServerId for a]'}
   ```

3. (Optional) Each `CREATE DATABASE` allocates the databases, but this may result in an unbalanced distribution. You can run `REALLOCATE DATABASES` to make the cluster re-balance databases across all servers that are part of the cluster.

   ```
   REALLOCATE DATABASES
   ```

   For more information, see Operations Manual 5 → Reallocate databases.

(Optional) Restore roles and privileges

You can restore the roles and privileges associated with each of your databases in case you first performed a backup including the metadata. This can be achieved by using the command `--include -metadata=none|all|users|roles`. For more information about this procedure, see Back up an online database.

In case you already performed this action, run the respective metadata script *data/scripts/<databasename>/restore_metadata.cypher*, with the `neo4j-admin restore` command output, using Cypher Shell:

Using `cat` (UNIX):

```
cat $NEO4J_HOME/data/scripts/<databasename>/restore_metadata.cypher | bin/cypher-shell -u user -p password
-a ip_address:port -d system --param "database => '<databasename>'"
```

Using `type` (Windows):

```
type $NEO4J_HOME\data\scripts\<databasename>\restore_metadata.cypher | bin\cypher-shell.bat -u user -p
password -a ip_address:port -d system --param "database => '<databasename>'"
```

Monitor the logs

When Neo4j starts, it can be useful to monitor the logs for any errors or warnings caused by the migration. You can find information about the upgrade in the *neo4j.log* file.

6.5.5. Reference

Lists of changes

- Breaking changes between Neo4j 4.4 and Neo4j 5
- Changes to procedures & functions in Neo4j 5
- Changes to Java API in Neo4j 5
- Changes to configuration settings in Neo4j 5

See Cypher Manual → Deprecations, additions and compatibility for changes in the Cypher language.

Examples of special cases

- Migrate a standalone server (Helm)

Breaking changes between Neo4j 4.4 and Neo4j 5.x

The following are all the breaking changes between Neo4j 4.4 and Neo4j 5.x.

JDK 17 support and Scala upgrade

Neo4j 5 supports JDK 17 and Scala 2.13 upgrade.

The following table shows the compatible Java Virtual Machine (JVM) to run a Neo4j instance.

Table 1. Neo4j version and JVM requirements

Neo4j version	Java version
5.x	Java SE 17
v4.x	Java SE 11
v3.x	Java SE 8

> If you have other Java applications running on the machine hosting Neo4j, make sure those applications are compatible with the Java version your Neo4j is running on. Alternatively, you should configure to run multiple JDKs on the same machine.

Java API

All breaking changes to Java API classes and settings are listed in the Changelogs section - Changes to Java API in Neo4j 5.

The Traversal API is reintroduced in Neo4j 5 after a thorough code review, more documentation, and examples.

Autonomous clusters Enterprise only

In Neo4j 5, the Causal Cluster is replaced by a **new clustering implementation**, called *Autonomous clusters*. It provides automated placement of databases and horizontal scaling. DBAs can specify the number of copies of each database within the cluster so that you can have more servers than copies of the database, e.g., 5 databases, with 3 copies of each database distributed across 6 servers. Autonomous clusters offer improved orchestration capabilities.

> When migrating to Neo4j 5, it is advisable to start from scratch with the new cluster. Make sure that you read all the details in Operations Manual → Clustering.

Key features of Autonomous clusters

- Databases are replicated to a subset of servers within a cluster.
- All servers are created equal.
- The cluster decides which databases are allocated to which servers.
- A database copy can be Primary or Secondary.
- The allocation may change as servers are added and removed over time.

Deprecated terms

- Core — replaced by Primary database copy.
- Read Replica — replaced by Secondary database copy.

- Standalone — no longer a distinct server behaviour, it is just a cluster of size one. This does mean that it opens additional ports compared to 4.4.

Neo4j Admin and Neo4j CLIs

Neo4j Admin CLI

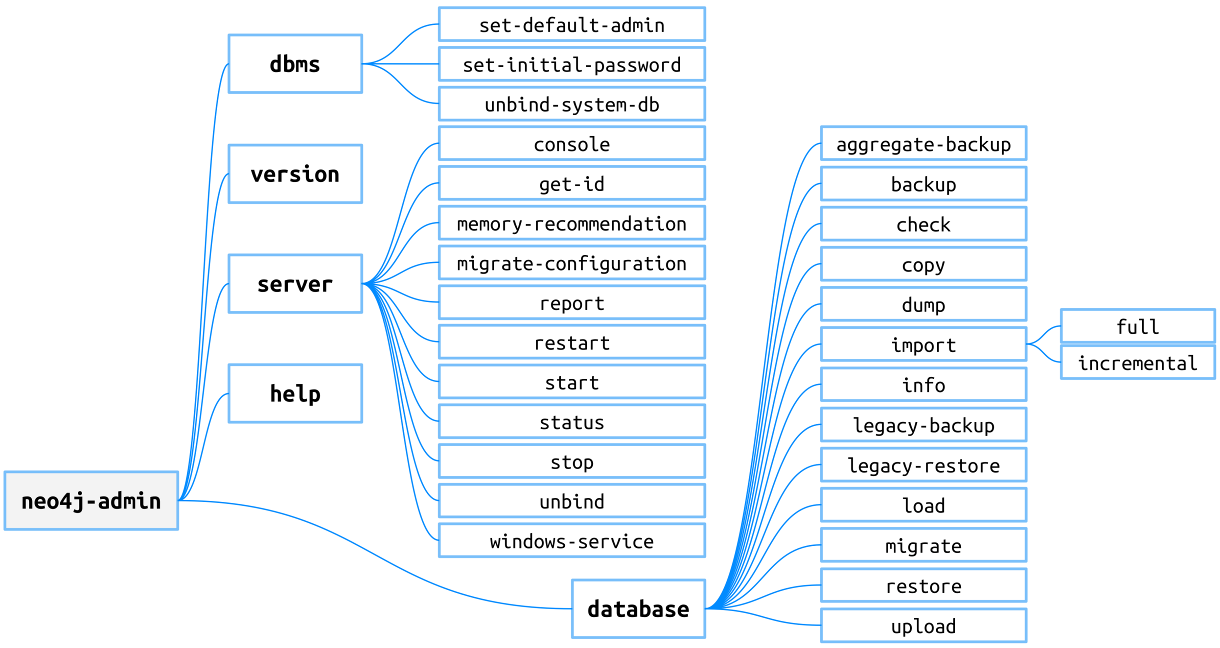
Ne4j Admin is refreshed. All admin functionalities are consolidated in a single tool and grouped by scope. It has improved features, more control, and a consistent set of arguments for the admin.

`neo4j-admin` has the following general synopsis:

`neo4j-admin [category] [command] [subcommand]`

All administration commands, except for `help` and `version`, are organized into the following three categories:

- `dbms` - DBMS-wide (for single and clustered environments) administration tasks
- `server` - server-wide administration tasks
- `database` - database-specific administration tasks



For more information about the new administration commands, see Operations Manual → Neo4j Admin and Neo4j CLI.

Neo4j CLI

`neo4j` has the following general synopsis:

```
neo4j [command]
```

The `command` is an alias for a command in the `neo4j-admin server` category.

[command]	neo4j-admin
`console`	`neo4j-admin server console`
`restart`	`neo4j-admin server restart`
`start`	`neo4j-admin server start`
`status`	`neo4j-admin server status`
`stop`	`neo4j-admin server stop`

For more information, see Operations Manual → Neo4j Admin and Neo4j CLI.

Backup and restore Enterprise only

The backup and restore have undergone a major overhaul for Neo4j 5. The new subsystem for backup and restore now supports differential backups, (limited) point-in-time restore, and artefacts. It is not compatible with the legacy subsystem.

The legacy Neo4j 4 backup/restore commands are still available (with a naming change) for backward compatibility so that you can restore Neo4j 4 backups.

Some of its main features include but are not limited to:

- Full and differential backups.
- Backups are immutable artifacts (compressed single items in the file system)
- An API to ease operability.
- Restore until point-in-time or a transaction ID across differential backup sets.
- Aggregation can be performed on a backup chain to create a new, recovered Full Backup.

  For more information, see Operations Manual → Backup and restore.
- Backup for autonomous cluster*

With an autonomous cluster, you no longer have a guarantee that a database will be available on any given server. By supplying an appropriate set of URIs and `--database` parameter, the operator can build an appropriate backup command. The command will loop through the URIs, attempting to serve the request in the `-database=` parameter. Databases already backed up on a previous URI will be ignored.

Configuration settings refresh

- In Neo4j 5, all configuration settings are divided into the namespaces `server`, `dbms`, `db`, `initial`, `browser`, and `client`. This means several settings are renamed. For example, `metrics.enabled` is renamed `server.metrics.enabled`.
- The existing Fabric configuration settings identified by the `fabric` namespace are removed from the

neo4j.conf file.

- The term `internal` replaces the terms `unsupported` and `experimental` used in previous versions. This namespace is dedicated to features that are used internally and may change without notice.
- Settings in *neo4j.conf* have strict validation enabled by default.
  The old parameter `dbms.config.strict_validation=false` by default is replaced by `server.config.strict_validation.enabled=true` by default. This ensures the configuration file does not contain errors or incorrect information and prevents Neo4j from starting with faulty settings in the *neo4j.conf* file.
- Enterprise only `db.tx_state.memory_allocation` is now `ON_HEAP` by default.
- Neo4j Admin now includes a tool for migrating configuration files. When migrating 4.4 configurations to 5.x, the tool notifies about settings that have not migrated, for instance, because a concept or behavior does not exist in the new `MAJOR` version. For more information, see the Operations Manual → Migrate the Neo4j configuration file.

For the full list of changed settings, see Changes to configuration settings in Neo4j 5.

Composite databases for Fabric

All configuration settings in the `fabric` namespace in the *neo4j.conf* file are moved to the `system` database. These settings are now managed through Cypher.

Fabric is used to manage sharded and federated databases with dynamic compositions. A composite database is treated like any other database and created with a name that identifies a more complex storage infrastructure, formed by an aggregation of local, clustered, and remote databases.

As part of this improvement, Neo4j users can now use Cypher to configure Fabric and:

- Change the Fabric configuration without a restart
- Support for multiple Fabric databases
- Provide a common approach to managing databases, whether they are local, remote, or sharded.
- Integrate the Neo4j database infrastructure by combining Fabric with autonomous clusters.

For more information, see Operations Manual → Composite databases.

Cypher

Cypher syntax

All changes in the Cypher language syntax are detailed in Cypher Manual → Removals, deprecations, additions and extensions. Thoroughly review this section in the version you are moving to and make the necessary changes in your code.

Error/Warning/Info handling in Cypher

Many semantic errors that Cypher finds are reported as `Neo.ClientError.Statement.SyntaxError` when they are semantic errors. In Neo4j 5, the metadata returned by Cypher queries is improved.

- The severity of some of the Warning codes is moved to Info:
  - `SubqueryVariableShadowingWarning` → `SubqueryVariableShadowing`
  - `NoApplicableIndexWarning` → `NoApplicableIndex`
  - `CartesianProductWarning` → `CartesianProduct`
  - `DynamicPropertyWarning` → `DynamicProperty`
  - `EagerOperatorWarning` → `EagerOperator`
  - `ExhustiveShortestPathWarning` → `ExhaustiveShortestPath`
  - `UnboundedVariableLengthPatternWarning` → `UnboundedVariableLengthPattern`
  - `ExperimentalFeature` → `RuntimeExperimental`

Neo4j 5 APOC Core Library

APOC Core is a subset of the APOC library that is fully supported by Neo4j engineering.

The complete APOC library is now referred to as **APOC Extended** to avoid confusion. APOC Extended is supported by the Neo4j Community exclusively.

> Neo4j does not recommend or encourage the use of the APOC Extended methods in production.

APOC Core

Please see APOC Core documentation.

The APOC Core Library is included as part of the Neo4j installation package, docker images, and most features are available in Aura. For more information, see Neo4j Aura → APOC support.

For the source code, please see the APOC Core source code repository.

As of APOC 5, the *neo4j.conf* file no longer supports `apoc.*` configuration settings. All configuration settings must be specified in *apoc.conf* or set using environment variables.

> If the config settings are in the *neo4j.conf* and `server.config.strict_validation.enabled` is set to `true`, the database will not start. Further to that, if `server.config.strict_validation.enabled` is disabled, any `apoc.*` settings left in *neo4j.conf* will not be read and therefore not set.

APOC Extended

The APOC Full package (APOC Core + APOC Extended) is no longer available. If you are interested in using non-supported APOC Extended methods, you need to refer to the https://github.com/neo4j-contrib/neo4j-apoc-procedures repository.

The source code is available in the APOC Extended source code repository.

Neo4j indexes

New RANGE and POINT indexes replace BTREE index

The BTREE index type is removed. Therefore, all BTREE indexes and index-backed constraints with index providers `native-btree-1.0` or `lucene+native-3.0` must be replaced or removed before switching to the 5.x binaries. The most appropriate approach would be to recreate them in 4.4 before migrating to 5.x. Then, during the migration, 5.x will check that each BTREE index and index-backed constraint has an equivalent type of index and provider and will drop them.

> Neo4j will not start if:
>
> - Any BTREE index exists and there is not at least one index of RANGE/POINT/TEXT type on the same schema.
> - Any index-backed constraints with providers `native-btree-1.0` or `lucene+native-3.0` exist unless there is a constraint of the same constraint type on the same schema with the `range-1.0` provider.

Logging

As of Neo4j 5, Neo4j uses Log4j 2 for logging.

Changes to configuration

The configuration settings for file management are moved to a Log4j configuration file, and a review and simplification of content settings for the query log are performed.

> For more advanced usages, see the Log4j official documentation.
>
> For more information on how a default Neo4j installation behaves and some typical examples of configuring it to use Log4j, see Operations Manual → Logging.

Metrics

All metric names now include `dbms` or `database` namespaces, and the setting `metrics.namespaces.enabled` that toggled this behavior is removed.

All settings to enable and disable a metric type (`metrics.*.enabled`) are removed. They are replaced by `server.metrics.filter` which takes a regex of what metrics to enable.

Removals

Unused and legacy code is removed from the product:

- Legacy HTTP transactional API - specific requests without the database name in the URI, such as `db/data/transaction`.
- `version` / `version_info` of result summary.
- Session #reset from Java driver (long deprecated).
- BTREE indexes - use RANGE and POINT indexes instead.
- Cypher 3.5 mode.
- Disallow repeated relationship variables.
- Automatic coercion of a list to a boolean.
- `exists()` function to test if property is null.
- Multi-datacenter settings, including `causal_clustering.multi_dc_license`.
- Procedures:

Table 2. Removed procedures

Procedure	Replacement
`db.indexes`	`SHOW INDEXES` command
`db.indexDetails`	`SHOW INDEXES YIELD *` command
`db.schemaStatements`	`SHOW INDEXES YIELD *` command and `SHOW CONSTRAINTS YIELD *` command
`db.constraints`	`SHOW CONSTRAINTS` command
`db.createIndex`	`CREATE INDEX` command
`db.createUniquePropertyConstraint`	`CREATE CONSTRAINT … IS UNIQUE` command
`db.index.fulltext.createNodeIndex`	`CREATE FULLTEXT INDEX` command
`db.index.fulltext.createRelationshipIndex`	`CREATE FULLTEXT INDEX` command
`db.index.fulltext.drop`	`DROP INDEX` command
`dbms.procedures`	`SHOW PROCEDURES` command
`dbms.functions`	`SHOW FUNCTIONS` command
`dbms.listTransactions`	`SHOW TRANSACTIONS` command
`dbms.killTransaction`	`TERMINATE TRANSACTIONS` command
`dbms.killTransactions`	`TERMINATE TRANSACTIONS` command

Procedure	Replacement
`dbms.listQueries`	`SHOW TRANSACTIONS` command
`dbms.killQuery`	`TERMINATE TRANSACTIONS` command
`dbms.killQueries`	`TERMINATE TRANSACTIONS` command
`dbms.scheduler.profile`	-

Deprecations

- `apoc.create.uuid()` and `apoc.create.uuids()` — deprecated and replaced by the existing `UUID.randomUUID()`.

External dependencies

All Neo4j 5 external dependencies and their versions are listed at Neo4j Maven repository.

Changes to procedures and functions in Neo4j 5

The following are all changes and deprecations for the Neo4j procedures in Neo4j 5.

Table 3. All changes to procedures and functions

Name	Community Edition	Enterprise Edition	Comment
`db.createNodeKey()`	No	Yes	Removed Replaced by `OPTIONS` of the `CREATE CONSTRAINT ... IS NODE KEY` command.
`db.constraints()`	Yes	Yes	Removed Replaced by `SHOW CONSTRAINTS`.
`db.createUniquePropertyConstraint()`	Yes	Yes	Removed Replaced by `OPTIONS` of the `CREATE CONSTRAINT ... IS UNIQUE` command.
`db.createIndex()`	Yes	Yes	Removed Replaced by `OPTIONS` of the `CREATE INDEX` command.
`db.indexes()`	Yes	Yes	Removed Replaced by `SHOW INDEXES`.
`db.indexDetails()`	Yes	Yes	Removed Replaced by `SHOW INDEXES YIELD *`.
`db.schemaStatements()`	Yes	Yes	Removed Replaced by `SHOW INDEXES YIELD *` and `SHOW CONSTRAINTS YIELD *`.
`dbms.procedures()`	Yes	Yes	Removed Replaced by `SHOW PROCEDURES`.
`dbms.functions()`	Yes	Yes	Removed Replaced by `SHOW FUNCTIONS`.
`dbms.killQueries()`	Yes	Yes	Removed Replaced by `TERMINATE TRANSACTIONS`.
`dbms.killQuery()`	Yes	Yes	Removed Replaced by `TERMINATE TRANSACTIONS`.
`dbms.killTransaction()`	Yes	Yes	Removed Replaced by `TERMINATE TRANSACTIONS`.

Name	Community Edition	Enterprise Edition	Comment
`dbms.killTransactions()`	Yes	Yes	Removed Replaced by `TERMINATE TRANSACTIONS`.
`dbms.listQueries()`	Yes	Yes	Removed Replaced by `SHOW TRANSACTIONS`.
`dbms.listTransactions()`	Yes	Yes	Removed Replaced by `SHOW TRANSACTIONS`.
`dbms.security.createUser()`	Yes	Yes	Removed Replaced by `CREATE USER`.
`dbms.security.deleteUser()`	Yes	Yes	Removed Replaced by `DROP USER`.
`dbms.security.suspendUser()`	No	Yes	Removed Replaced by `ALTER USER`.
`dbms.security.activateUser()`	No	Yes	Removed Replaced by `ALTER USER`.
`dbms.security.changeUserPassword()`	No	Yes	Removed Replaced by `ALTER USER`.
`dbms.security.changePassword()`	Yes	Yes	Removed Replaced by `ALTER CURRENT USER SET PASSWORD`.
`dbms.security.createRole()`	No	Yes	Removed Replaced by `CREATE ROLE`.
`dbms.security.deleteRole()`	No	Yes	Removed Replaced by `DROP ROLE`.
`dbms.security.addRoleToUser()`	No	Yes	Removed Replaced by `GRANT ROLE`.
`dbms.security.removeRoleFromUser()`	No	Yes	Removed Replaced by `REVOKE ROLE`.
`dbms.security.listUsers()`	Yes	Yes	Removed Replaced by `SHOW USERS`.
`dbms.security.listRolesForUser()`	No	Yes	Removed Replaced by `SHOW USERS`.
`dbms.security.listRoles()`	No	Yes	Removed Replaced by `SHOW ROLES`.
`dbms.security.listUsersForRole()`	No	Yes	Removed Replaced by `SHOW ROLES WITH USERS`.
`db.index.fulltext.createNodeIndex()`	Yes	Yes	Removed Replaced by `CREATE FULLTEXT INDEX …`.
`db.index.fulltext.createRelationshipIndex()`	Yes	Yes	Removed Replaced by `CREATE FULLTEXT INDEX …`.
`db.index.fulltext.drop()`	Yes	Yes	Removed Replaced by `DROP INDEX …`.

Changes to Java API in Neo4j 5

This page lists all changes to Java API between Neo4j 4.4 and Neo4j 5.x.

Changes to classes, interfaces, and enums

List of removals and additions

Class	Change	Action
FormattedLogFormat	Removed	Format is configured in the Log4j 2 configuration file.
Logger	Removed	Use Log directly.
OnlineBackup	Removed	Use standalone tool.
TransactionStreamingStrategy	Removed	Used by online backup.
ClusterDatabaseManagementService	Removed	Use EnterpriseDatabaseManagementService.
ClusterDatabaseManagementServiceBuilder	Removed	Use EnterpriseDatabaseManagementServiceBuilder.
InProcessNeo4j	Removed	Accidentally added to the public API.
CypherParserVersion	Removed	Enum used in removed setting.
DriverApi	Removed	Enum used in removed setting.
Mode	Removed	Enum used in removed setting.
SchemaIndex	Removed	Enum used in removed setting.
NamedService	New	Used to provide a .getName() to a @ServiceProvider.
SeedProvider	New	Entry point for seed providers. See SeedProvider.
Neo4jDatabaseManagementServiceBuilder	New	Common interface for *DatabaseManagementServiceBuilder.
InstanceModeConstraint	New	New enum used in settings.
SystemDatabaseMode	New	New enum used in settings.

Changes to methods

org.neo4j.logging.Log

The following methods are removed:

Method	Change	Action
`Logger errorLogger()`	Removed	Use `Log.error(…)` directly.
`Logger warnLogger()`	Removed	Use `Log.warn(…)` directly.
`Logger infoLogger()`	Removed	Use `Log.info(…)` directly.
`Logger debugLogger()`	Removed	Use `Log.debug(…)` directly.

org.neo4j.graphdb.config.Setting<T>

The following method is removed:

Method	Change	Action
`boolean internal()`	Removed	Not used anymore. Used to mark settings as internal hidden settings.

org.neo4j.graphdb.Entity

The following methods are added. They are visible from the inheriting `Node` and `Relationship`:

Method	Change	Action
`String getElementId()`	New	Returns the unique element ID. Guaranteed to be unique only within a transaction.
`void delete()`	New	Moved from `Node` and `Relationship` to the common interface.

org.neo4j.graphdb.schema.IndexType

The following index type is removed:

Index type	Change	Action
`IndexType.BTREE`	Removed	Is replaced by `IndexType.RANGE`.

`org.neo4j.graphdb.spatial.Coordinate`

The following method is removed:

Method	Change	Action
`List<Double> getCoordinate()`	Removed	Replaced by the two following methods: `double[] getCoordinate()` — Get the coordinates. To be used only to read the values. `double[] getCoordinateCopy()` — Get a copy of the coordinates. Can safely be modified and stored.

`org.neo4j.graphdb.Transaction`

The following methods are added:

Method	Change	Action
`Node getNodeByElementId(String)`	New	Looks up a node by element ID.
`Relationship getRelationshipByElementId(String)`	New	Looks up a relationship by element ID.

`org.neo4j.harness.junit.extension.Neo4jExtensionBuilder`

The following methods are removed:

Method	Change	Action
`Neo4jExtensionBuilder copyFrom(File)`	Removed	Use `Neo4jExtensionBuilder.copyFrom(Path)`.
`Neo4jExtensionBuilder withFolder(File)`	Removed	Use `Neo4jExtensionBuilder.withFolder(Path)`.
`Neo4jExtensionBuilder withExtensionFactories(Iterable<ExtensionFactory<?>>)`	Removed	Removed.
`Neo4jExtensionBuilder withFixture(File)`	Removed	Use `Neo4jExtensionBuilder.withFixture(Path)`.

`org.neo4j.harness.junit.rule.Neo4jRule`

The following methods are removed:

Method	Change	Action
`Neo4jRule(File)`	Removed	Use `Neo4jRule(Path)`.
`Neo4jRule copyFrom(File)`	Removed	Use `Neo4jRule.copyFrom(Path)`.
`Neo4jRule withExtensionFactories(Iterable<ExtensionFactory<?>>)`	Removed	Removed.
`Neo4jRule withFixture(File)`	Removed	Use `Neo4jRule.withFixture(Path)`.

`org.neo4j.harness.Neo4jBuilders`

The following method is removed:

Method	Change	Action
`Neo4jBuilder newInProcessBuilder(File)`	Removed	Use `Neo4jBuilders.newInProcessBuilder(Path)`.

`org.neo4j.harness.Neo4jConfigurator<T>`

The following methods are removed:

Method	Change	Action
`T copyFrom(File)`	Removed	Use `Neo4jConfigurator.copyFrom(Path)`.
`T withExtensionFactories(Iterable<ExtensionFactory<?>>)`	Removed	Removed.
`T withFixture(File)`	Removed	Use `Neo4jConfigurator.withFixture(Path)`.
`T withWorkingDir(File)`	Removed	Use `Neo4jConfigurator.withWorkingDir(Path)`.

`com.neo4j.harness.junit.rule.EnterpriseNeo4jRule`

The deprecated constructor that took a `File` is removed:

Method	Change	Action
`EnterpriseNeo4jRule(File)`	Removed	Use `EnterpriseNeo4jRule(Path)`.

`org.neo4j.kernel.api.exceptions.Status`

The following status codes are added:

Status code	Change	Description
`Database.HomeDatabaseNotFound`	New	The home database provided does not currently exist.
`Request.DeprecatedFormat`	New	The requested format is deprecated.
`Schema.HintedIndexNotFound`	New	The hinted index does not exist, please check the schema.
`Statement.ParameterNotProvided`	New	Did not supply query with enough parameters.
`Statement.UnsupportedOperationError`	New	

The following status codes are removed:

Statement.MissingAlias	Removed	
`Transaction.TransientTransactionFailure`	Removed	

Some status codes are renamed to indicate changed severity from `WARNING` to `INFORMATION`:

Old name	New name
`Statement.CartesianProductWarning`	`Statement.CartesianProduct`
`Statement.DynamicPropertyWarning`	`Statement.DynamicProperty`
`Statement.EagerOperatorWarning`	`Statement.EagerOperator`
`Statement.ExhaustiveShortestPathWarning`	`Statement.ExhaustiveShortestPath`
`Statement.ExperimentalFeature`	`Statement.RuntimeExperimental`
`Statement.NoApplicableIndexWarning`	`Statement.NoApplicableIndex`
`Statement.SubqueryVariableShadowingWarning`	`Statement.SubqueryVariableShadowing`
`Statement.UnboundedVariableLengthPatternWarning`	`Statement.UnboundedVariableLengthPattern`

Renamed classes, interfaces, and enums

Old name	New name
`com.neo4j.configuration.ServerGroupName`	`class com.neo4j.configuration.ServerTag`
`com.neo4j.configuration.CausalClusteringSettings`	`com.neo4j.configuration.ClusterSettings`
`com.neo4j.configuration.FabricEnterpriseSettings`	`com.neo4j.fabric.configuration.FabricEnterpriseSettings`

`org.neo4j.graphdb.event.DatabaseEventListener`

The following two methods do not have a default implementation anymore and need to be implemented by listeners:

- `void databaseCreate(DatabaseEventContext)`
- `void databaseDrop(DatabaseEventContext)`

`org.neo4j.graphdb.GraphDatabaseService`

A new method is added, which is a non-blocking variant of the already existing method with the same name:

- `boolean isAvailable()`

`org.neo4j.graphdb.Node`

The following methods are changed to return a `ResourceIterable` instead of an `Iterable`:

Method	Change
`ResourceIterable<Relationship> getRelationships()`	New
`ResourceIterable<Relationship> getRelationships(Direction)`	New
`ResourceIterable<Relationship> getRelationships(RelationshipType…)`	New
`ResourceIterable<Relationship> getRelationships(Direction, RelationshipType…)`	New
`Iterable<Relationship> getRelationships()`	Removed
`Iterable<Relationship> getRelationships(Direction)`	Removed
`Iterable<Relationship> getRelationships(RelationshipType…)`	Removed

Method	Change
`Iterable<Relationship> getRelationships(Direction, RelationshipType…)`	Removed

`org.neo4j.dbms.api.DatabaseManagementServiceBuilder`

This class now implements `Neo4jDatabaseManagementServiceBuilder`, which is the new preferred interface to use. As part of this change, the following fields and methods are removed since they exposed internal implementations:

- `Config.Builder config`
- `List<DatabaseEventListener> databaseEventListeners`
- `DependencyResolver dependencies`
- `List<ExtensionFactory<?>> extensions`
- `Path homeDirectory`
- `Monitors monitors`
- `Map<String,URLAccessRule> urlAccessRules`
- `LogProvider userLogProvider`
- `DatabaseManagementServiceBuilder(File)`
- `DatabaseManagementServiceBuilder(Path, Predicate<Class<? extends ExtensionFactory>>)`
- `DatabaseManagementServiceBuilder loadPropertiesFromFile(String)`
- `DatabaseManagementServiceBuilder setConfigRaw(Map<String,String>)`
- `DatabaseManagementServiceBuilder setExternalDependencies(DependencyResolver)`
- `DatabaseManagementServiceBuilder setMonitors(Monitors)`
- `DatabaseManagementServiceBuilder addURLAccessRule(String, URLAccessRule)`
- `DatabaseManagementService newDatabaseManagementService(Config, ExternalDependencies)`
- `DbmsInfo getDbmsInfo(Config)`
- `String getEdition()`
- `Function<GlobalModule,AbstractEditionModule> getEditionFactory(Config)`
- `Config augmentConfig(Config)`
- `ExternalDependencies databaseDependencies()`

Changes to Neo4j settings

`com.neo4j.configuration.OnlineBackupSettings`

Setting	Change
`Setting<TransactionStreamingStrategy> incremental_backup_strategy`	Removed
`Setting<Duration> store_copy_max_retry_time_per_request`	New

`com.neo4j.configuration.MetricsSettings`

All settings to enable and disable a metric type are replaced by `metrics_filter`, which takes a regex of what metrics to enable:

Setting	Change
`Setting<List<GlobbingPattern>> metrics_filter`	New
`Setting<Boolean> bolt_messages_enabled`	Removed
`Setting<Boolean> causal_clustering_enabled`	Removed
`Setting<Boolean> cypher_planning_enabled`	Removed
`Setting<Boolean> database_counts_enabled`	Removed
`Setting<Boolean> database_operation_count_enabled`	Removed
`Setting<Boolean> jvm_buffers_enabled`	Removed
`Setting<Boolean> jvm_file_descriptors_enabled`	Removed
`Setting<Boolean> jvm_gc_enabled`	Removed
`Setting<Boolean> jvm_heap_enabled`	Removed
`Setting<Boolean> jvm_memory_enabled`	Removed
`Setting<Boolean> jvm_pause_time_enabled`	Removed
`Setting<Boolean> jvm_threads_enabled`	Removed
`Setting<Boolean> metrics_namespaces_enabled`	Removed

Setting	Change
`Setting<Boolean> neo_check_pointing_enabled`	Removed
`Setting<Boolean> neo_counts_enabled`	Removed
`Setting<Boolean> neo_memory_pools_enabled`	Removed
`Setting<Boolean> neo_page_cache_enabled`	Removed
`Setting<Boolean> neo_server_enabled`	Removed
`Setting<Boolean> neo_store_size_enabled`	Removed
`Setting<Boolean> neo_transaction_logs_enabled`	Removed
`Setting<Boolean> neo_tx_enabled`	Removed

`com.neo4j.configuration.OnlineBackupSettings`

Setting	Change
`Setting<Duration> store_copy_max_retry_time_per_request`	New
`Setting<TransactionStreamingStrategy> incremental_backup_strategy`	Removed

`com.neo4j.configuration.SecuritySettings`

Setting	Change
`Setting<String> key_name`	New
`Setting<SecureString> keystore_password`	New
`Setting<Path> keystore_path`	New
`Setting<String> ldap_authorization_nested_groups_search_filter`	New
`Setting<Boolean> ldap_authorization_nested_groups_enabled`	New

Setting	Change
`Setting<Boolean> ldap_authentication_use_samaccountname`	Removed

As part of the logging changes, the following settings are removed and moved to the Log4j 2 configuration file:

Setting	Change
`Setting<Path> security_log_filename`	Removed
`Setting<FormattedLogFormat> security_log_format`	Removed
`Setting<Level> security_log_level`	Removed
`Setting<Integer> store_security_log_max_archives`	Removed
`Setting<Duration> store_security_log_rotation_delay`	Removed
`Setting<Long> store_security_log_rotation_threshold`	Removed

`com.neo4j.configuration.SecuritySettings.OIDCSetting`

Setting	Change
`String name()`	New
`Setting<URI> redirect_uri`	Removed

`com.neo4j.configuration.EnterpriseEditionSettings`

Setting	Change
`int MAX_PRIMARIES = 11`	New
`int MAX_SECONDARIES = 20`	New
`int MIN_PRIMARIES = 1`	New
`int MIN_SECONDARIES = 0`	New

Setting	Change
`Setting<Set<String>> initial_allowed_databases`	New
`Setting<String> initial_database_allocator`	New
`Setting<Integer> initial_default_primaries_count`	New
`Setting<Integer> initial_default_secondaries_count`	New
`Setting<Set<String>> initial_denied_databases`	New
`Setting<InstanceModeConstraint> initial_mode_constraint`	New
`Setting<List<ServerTag>> server_groups`	New
`Setting<SystemDatabaseMode> system_database_mode`	New
`Setting<Boolean> enable_clustering_in_standalone`	Removed
`Setting<List<String>> dynamic_setting_whitelist`	Removed

`com.neo4j.fabric.configuration.FabricEnterpriseSettings`

Setting	Change
`Setting<DriverApi> driver_api`	Removed

`com.neo4j.configuration.FabricEnterpriseSettings.GraphSetting`

Setting	Change
`Setting<String> driver_policy`	New
`Setting<DriverApi> driver_api`	Removed

Changes to Neo4j cluster settings

com.neo4j.configuration.ClusterNetworkSettings

Setting	Change
`Setting<Duration> catchup_client_inactivity_timeout`	New
`Setting<Duration> network_handshake_timeout`	New
`Setting<Integer> network_max_chunk_size`	New
`Setting<List<String>> network_supported_compression_algos`	New
`Setting<Boolean> network_use_native_transport`	New

com.neo4j.configuration.KubernetesSettings

Setting	Change
`Setting<SocketAddress> kubernetes_address`	New
`Setting<Path> kubernetes_ca_crt`	New
`Setting<String> kubernetes_cluster_domain`	New
`Setting<String> kubernetes_label_selector`	New
`Setting<Path> kubernetes_namespace`	New
`Setting<String> kubernetes_service_port_name`	New
`Setting<Path> kubernetes_token`	New

com.neo4j.configuration.ClusterSettings

A few constants are added:

Setting	Change
`int DEFAULT_DISCOVERY_PORT = 5000`	New
`int DEFAULT_RAFT_PORT = 7000`	New

Setting	Change
`int DEFAULT_TRANSACTION_PORT = 6000`	New

The following settings are moved to the `ClusterNetworkSettings` class and renamed:

Setting	Change
`Setting<Duration> catch_up_client_inactivity_timeout`	Removed
`Setting<Duration> handshake_timeout`	Removed
`Setting<Integer> store_copy_chunk_size`	Removed
`Setting<List<String>> compression_implementations`	Removed

The following settings are moved to `KubernetesSettings`:

Setting	Change
`Setting<SocketAddress> kubernetes_address .>`	Removed
`Setting<Path> kubernetes_ca_crt`	Removed
`Setting<String> kubernetes_cluster_domain`	Removed
`Setting<String> kubernetes_label_selector`	Removed
`Setting<Path> kubernetes_namespace`	Removed
`Setting<String> kubernetes_service_port_name`	Removed
`Setting<Path> kubernetes_token`	Removed

The following settings are added:

Setting	Change
`Setting<Duration> catchup_pull_interval`	New
`Setting<List<String>> catchup_upstream_strategy`	New

Setting	Change
`Setting<String> catchup_user_defined_upstream_strategy`	New
`Setting<SocketAddress> cluster_advertised_address`	New
`Setting<SocketAddress> cluster_listen_address`	New
`Setting<List<SocketAddress>> discovery_initial_members`	New
`Setting<Level> discovery_log_level`	New
`Setting<Long> raft_apply_buffer_max_bytes`	New
`Setting<Integer> raft_apply_buffer_max_entries`	New
`Setting<Duration> raft_binding_timeout`	New
`Setting<Integer> raft_client_max_channels`	New
`Setting<DurationRange> raft_election_failure_detection_window`	New
`Setting<Long> raft_in_queue_batch_max_bytes`	New
`Setting<DurationRange> raft_leader_failure_detection_window`	New
`Setting<SelectionStrategies> raft_leader_transfer_balancing_strategy`	New
`Setting<ServerTag> raft_leader_transfer_priority_group`	New
`Setting<Integer> minimum_initial_members`	New
`Setting<String> raft_log_prune_strategy`	New
`Setting<Long> raft_log_shipping_buffer_max_bytes`	New
`Setting<Integer> raft_log_shipping_buffer_max_entries`	New
`Setting<Duration> raft_membership_join_max_lag`	New

Setting	Change
`Setting<Duration> raft_membership_join_timeout`	New
`Setting<String> routing_load_balancing_plugin`	New
`Setting<Boolean> routing_load_balancing_shuffle_enabled`	New
`Setting<Boolean> routing_reads_on_primaries_enabled`	New
`Setting<Boolean> routing_reads_on_writers_enabled`	New
`Setting<Boolean> security_cluster_status_auth_enabled`	New
`Setting<List<String>> seed_from_uri_providers`	New
`Setting<List<ServerTag>> catchup_connect_randomly_to_server_group`	New

The following settings are removed:

Setting	Change
`Setting<Integer> catchup_batch_size`	Removed
`Setting<List<ApplicationProtocolVersion>> catchup_implementations`	Removed
`Setting<Boolean> cluster_allow_reads_on_followers`	Removed
`Setting<Boolean> cluster_allow_reads_on_leader`	Removed
`Setting<Duration> cluster_binding_timeout`	Removed
`Setting<Duration> cluster_topology_refresh`	Removed
`Setting<Integer> command_applier_parallelism`	Removed
`Setting<List<ServerGroupName>> connect_randomly_to_server_group_strategy`	Removed
`Setting<ServerGroupName> default_leadership_priority_group`	Removed
`Setting<DurationRange> election_failure_detection_window`	Removed

Setting	Change
`Setting<Boolean> enable_pre_voting`	Removed
`Setting<Integer> global_session_tracker_state_size`	Removed
`Setting<Long> in_flight_cache_max_bytes`	Removed
`Setting<Integer> in_flight_cache_max_entries`	Removed
`Setting<InFlightCacheType in_flight_cache_type`	Removed
`Setting<List<SocketAddress>> initial_discovery_members`	Removed
`Setting<Duration> join_catch_up_max_lag`	Removed
`Setting<Duration> join_catch_up_timeout`	Removed
`Setting<Integer> last_flushed_state_size`	Removed
`Setting<SelectionStrategies> leader_balancing`	Removed
`Setting<Duration> leader_election_timeout`	Removed
`Setting<DurationRange> leader_failure_detection_window`	Removed
`Setting<String> load_balancing_plugin`	Removed
`Setting<Boolean> load_balancing_shuffle`	Removed
`Setting<Integer> log_shipping_max_lag`	Removed
`Setting<Duration> log_shipping_retry_timeout`	Removed
`Setting<Integer> max_raft_channels`	Removed
`Setting<Level> middleware_logging_level`	Removed
`Setting<Integer> minimum_core_cluster_size_at_formation`	Removed

Setting	Change
`Setting<Integer> minimum_core_cluster_size_at_runtime`	Removed
`Setting<Boolean> multi_dc_license`	Removed
`Setting<Duration> pull_interval`	Removed
`Setting<Integer> raft_handler_parallelism`	Removed
`Setting<List<ApplicationProtocolVersion>> raft_implementations`	Removed
`Setting<Long> raft_in_queue_max_batch_bytes`	Removed
`Setting<Integer> raft_log_entry_prefetch_buffer_max_entries`	Removed
`Setting<String> raft_log_implementation`	Removed
`Setting<String> raft_log_pruning_strategy`	Removed
`Setting<Integer> raft_membership_state_size`	Removed
`Setting<Boolean> refuse_to_be_leader`	Removed
`Setting<Integer> replicated_lease_state_size`	Removed
`Setting<Duration> replication_leader_await_timeout`	Removed
`Setting<Duration> replication_retry_timeout_base`	Removed
`Setting<Duration> replication_retry_timeout_limit`	Removed
`Setting<List<ServerGroupName>> server_groups`	Removed
`Setting<Integer> state_machine_apply_max_batch_size`	Removed
`Setting<Integer> state_machine_flush_window_size`	Removed
`Setting<Boolean> status_auth_enabled`	Removed

Setting	Change
`Setting<Duration> status_throughput_window`	Removed
`Setting<Integer> store_copy_parallelism`	Removed
`Setting<Integer> term_state_size`	Removed
`Setting<SocketAddress> transaction_advertised_address`	Removed
`Setting<SocketAddress> transaction_listen_address`	Removed
`Setting<Duration> unknown_address_logging_throttle`	Removed
`Setting<List<String>> upstream_selection_strategy`	Removed
`Setting<String> user_defined_upstream_selection_strategy`	Removed
`Setting<Integer> vote_state_size`	Removed

`org.neo4j.configuration.connectors.BoltConnector`

Setting	Change
`Setting<Duration> thread_pool_shutdown_wait_time`	Removed

`org.neo4j.configuration.GraphDatabaseSettings`

The following settings are added:

Setting	Change
`Setting<String> initial_default_database`	New
`Setting<Boolean> cypher_render_plan_descriptions`	New

The following settings are removed:

Setting	Change
`Setting<String> default_allowed`	Removed

Setting	Change
`Setting<String> default_database`	Removed
`Setting<String> default_schema_provider`	Removed
`Setting<Boolean> enable_clustering_in_standalone`	Removed
`Setting<Boolean> allow_upgrade`	Removed
`Setting<CypherParserVersion> cypher_parser_version`	Removed
`Setting<Integer> lucene_searcher_cache_size`	Removed
`Setting<Mode> mode`	Removed
`Setting<String> pagecache_swapper`	Removed
`Setting<String> pagecache_warmup_prefetch_whitelist`	Removed
`Setting<DriverApi> routing_driver_api`	Removed
`Setting<String> procedure_roles`	Removed
`Setting<String> procedure_whitelist`	Removed
`Setting<Boolean> read_only`	Removed
`Setting<Duration> reconciler_maximum_backoff`	Removed
`Setting<Integer> reconciler_maximum_parallelism`	Removed
`Setting<Boolean> reconciler_may_retry`	Removed
`Setting<Duration> reconciler_minimum_backoff`	Removed
`Setting<Integer> upgrade_processors`	Removed

The setting `record_format` is renamed to `db_format` to allow future, non-record formats, to be selected:

Setting	Change
`Setting<String> db_format`	New
`Setting<String> record_format`	Removed

The following settings are removed and their default values are now permanent:

Setting	Change
`Setting<Boolean> log_queries_allocation_logging_enabled`	Removed
`Setting<Boolean> log_queries_detailed_time_logging_enabled`	Removed
`Setting<Integer> log_queries_max_archives`	Removed
`Setting<Boolean> log_queries_page_detail_logging_enabled`	Removed
`Setting<Boolean> log_queries_parameter_full_entities`	Removed
`Setting<Boolean> log_queries_runtime_logging_enabled`	Removed
`Setting<Boolean> log_queries_transaction_id`	Removed
`Setting<Boolean> track_query_allocation`	Removed

The setting `pagecache_memory` has a changed type:

Setting	Change
`Setting<Long> pagecache_memory`	New
`Setting<String> pagecache_memory`	Removed

As part of the logging changes two new settings are added that will point to the Log4j 2 configuration files. All old logging settings are removed:

Setting	Change
`Setting<Path> server_logging_config_path`	New
`Setting<Path> user_logging_config_path`	New

Setting	Change
`Setting<Boolean> debug_log_enabled`	New
`Setting<FormattedLogFormat> default_log_format`	Removed
`Setting<FormattedLogFormat> store_internal_log_format`	Removed
`Setting<FormattedLogFormat> log_query_format`	Removed
`Setting<FormattedLogFormat> store_user_log_format`	Removed
`Setting<Path> store_internal_log_path`	Removed
`Setting<Level> store_internal_log_level`	Removed
`Setting<Integer> store_internal_log_max_archives`	Removed
`Setting<Duration> store_internal_log_rotation_delay`	Removed
`Setting<Long> store_internal_log_rotation_threshold`	Removed
`Setting<Path> store_user_log_path`	Removed
`Setting<Integer> store_user_log_max_archives`	Removed
`Setting<Duration> store_user_log_rotation_delay`	Removed
`Setting<Long> store_user_log_rotation_threshold`	Removed
`Setting<Boolean> store_user_log_to_stdout`	Removed
`Setting<Path> log_queries_filename`	Removed
`Setting<Long> log_queries_rotation_threshold`	Removed

Changes to the Traversal API

The traversal API can be accessed through `Transaction.traversalDescription()` and `Transaction.bidirectionalTraversalDescription()`. For the full documentation, see Java Reference → The traversal framework .

Added classes, interfaces, and enums as part of the Traversal API

interface	org.neo4j.graphdb.traversal.BidirectionalTraversalDescription
interface	org.neo4j.graphdb.traversal.BranchCollisionDetector
enum	org.neo4j.graphdb.traversal.BranchOrderingPolicies
interface	org.neo4j.graphdb.traversal.BranchOrderingPolicy
interface	org.neo4j.graphdb.traversal.BranchState<STATE>
enum	org.neo4j.graphdb.traversal.Evaluation
interface	org.neo4j.graphdb.traversal.Evaluator
class	org.neo4j.graphdb.traversal.Evaluators
interface	org.neo4j.graphdb.traversal.InitialBranchState<STATE>
interface	org.neo4j.graphdb.traversal.PathEvaluator<STATE>
class	org.neo4j.graphdb.traversal.Paths
enum	org.neo4j.graphdb.traversal.SideSelectorPolicies
interface	org.neo4j.graphdb.traversal.TraversalBranch
interface	org.neo4j.graphdb.traversal.TraversalDescription
interface	org.neo4j.graphdb.traversal.TraversalMetadata
interface	org.neo4j.graphdb.traversal.Traverser
enum	org.neo4j.graphdb.traversal.Uniqueness
interface	org.neo4j.graphdb.traversal.UniquenessFactory
interface	org.neo4j.graphdb.traversal.UniquenessFilter

org.neo4j.graphdb.PathExpanders

A few commonly used `PathExpander`s are added:

Setting	Change
allTypesAndDirections()	New
forConstantDirectionWithTypes(RelationshipType…)	New
forDirection(Direction)	New
forType(RelationshipType)	New
forTypeAndDirection(RelationshipType, Direction)	New
forTypesAndDirections(RelationshipType, Direction, RelationshipType, Direction, Object…)	New

Setting	Change
printingWrapper(…)	New

Changes to configuration settings in Neo4j 5

The following are all changes to Neo4j 4.4 configuration settings in Neo4j 5.

- Complete list of 4.4 configuration settings.
- Complete list of 5 (latest) configuration settings.

Changed default or values

Status	4.4 Name	5 Name	Notes
Renamed Warning	causal_clustering.leader_election_timeout	dbms.cluster.raft.leader_failure_detection_window	Default changed from 7s to 7s-12s.
Removed Warning	causal_clustering.store_copy_chunk_size	dbms.cluster.network.max_chunk_size	Replaced by the setting dbms.cluster.network.max_chunk_size.
Renamed Warning	dbms.http_enabled_modules	server.http_enabled_modules	Default changed from TRANSACTIONAL_ENDPOINTS, UNMANAGED_EXTENSIONS, BROWSER, ENTERPRISE_MANAGEMENT_ENDPOINTS to server.http_enabled_modules,TRANSACTIONAL_ENDPOINTS, UNMANAGED_EXTENSIONS, BROWSER, ENTERPRISE_MANAGEMENT_ENDPOINTS.
Removed Warning	dbms.logs.query.page_logging_enabled		Query page logging is enabled by default when using the setting db.logs.query.enabled=INFO.
Removed Warning	dbms.logs.query.time_logging_enabled		Query time logging is enabled by default when using the setting db.logs.query.enabled=INFO.
Renamed Warning	dbms.mode	initial.server.mode_constraint	Default changed from SINGLE to PRIMARY.
Renamed Warning	dbms.mode	server.cluster.system_database_mode	Default changed from SINGLE to PRIMARY.
Removed Warning	dbms.record_format	db.format	setting has different valid values between 4.4 and 5.

Removed

Status	4.4 Name
Removed	`causal_clustering.cluster_topology_refresh`
Removed	`causal_clustering.delete_store_before_store_copy`
Removed	`causal_clustering.enable_pre_voting`
Removed	`causal_clustering.enable_seed_validation`
Removed	`causal_clustering.middleware.akka.cluster.min_nr_of_members`
Removed	`causal_clustering.multi_dc_license`
Removed	`causal_clustering.protocol_implementations.catchup`
Removed	`causal_clustering.protocol_implementations.raft`
Removed	`causal_clustering.raft_log_implementation`
Removed	`causal_clustering.refuse_to_be_leader`
Removed	`causal_clustering.store_copy_backoff_max_wait`
Removed	`causal_clustering.temporary_database.extension_package_names`
Removed	`cypher.default_language_version`
Removed	`dbms.allow_single_automatic_upgrade`
Removed	`dbms.allow_upgrade`
Removed	`dbms.backup.incremental.strategy`
Removed	`dbms.capabilities.blocked`
Removed	`dbms.clustering.enable`
Removed	`dbms.directories.tx_log`
Removed	`dbms.index.default_schema_provider`
Removed	`dbms.index_searcher_cache_size`
Removed	`dbms.logs.debug.format`
Removed	`dbms.logs.debug.level`
Removed	`dbms.logs.debug.path`
Removed	`dbms.logs.debug.rotation.delay`
Removed	`dbms.logs.debug.rotation.keep_number`
Removed	`dbms.logs.debug.rotation.size`
Removed	`dbms.logs.default_format`
Removed	`dbms.logs.http.format`
Removed	`dbms.logs.http.path`
Removed	`dbms.logs.http.rotation.keep_number`
Removed	`dbms.logs.http.rotation.size`

Status	4.4 Name
Removed	dbms.logs.query.allocation_logging_enabled
Removed	dbms.logs.query.format
Removed	dbms.logs.query.parameter_full_entities
Removed	dbms.logs.query.path
Removed	dbms.logs.query.rotation.keep_number
Removed	dbms.logs.query.rotation.size
Removed	dbms.logs.query.runtime_logging_enabled
Removed	dbms.logs.query.time_logging_enabled
Removed	dbms.logs.query.transaction_id.enabled
Removed	dbms.logs.security.format
Removed	dbms.logs.security.level
Removed	dbms.logs.security.path
Removed	dbms.logs.security.rotation.delay
Removed	dbms.logs.security.rotation.keep_number
Removed	dbms.logs.security.rotation.size
Removed	dbms.logs.user.format
Removed	dbms.logs.user.path
Removed	dbms.logs.user.rotation.delay
Removed	dbms.logs.user.rotation.keep_number
Removed	dbms.logs.user.rotation.size
Removed	dbms.logs.user.stdout_enabled
Removed	dbms.memory.pagecache.swapper
Removed	dbms.rest.transaction.idle_timeout
Removed	dbms.routing.driver.api
Removed	dbms.security.ldap.authentication.use_samaccountname
Removed	dbms.security.procedures.default_allowed
Removed	dbms.security.procedures.roles
Removed	dbms.security.property_level.blacklist
Removed	dbms.security.property_level.enabled
Removed	dbms.track_query_allocation
Removed	fabric.database.name
Removed	fabric.driver.api

Status	4.4 Name
Removed	`fabric.driver.connection.connect_timeout`
Removed	`fabric.driver.connection.max_lifetime`
Removed	`fabric.driver.connection.pool.acquisition_timeout`
Removed	`fabric.driver.connection.pool.idle_test`
Removed	`fabric.driver.connection.pool.max_size`
Removed	`fabric.driver.event_loop_count`
Removed	`fabric.driver.idle_check_interval`
Removed	`fabric.driver.logging.leaked_sessions`
Removed	`fabric.driver.logging.level`
Removed	`fabric.driver.timeout`
Removed	`fabric.enabled_by_default`
Removed	`fabric.graph.mega.database`
Removed	`fabric.graph.mega.driver.api`
Removed	`fabric.graph.mega.driver.connection.connect_timeout`
Removed	`fabric.graph.mega.driver.connection.max_lifetime`
Removed	`fabric.graph.mega.driver.connection.pool.acquisition_timeout`
Removed	`fabric.graph.mega.driver.connection.pool.idle_test`
Removed	`fabric.graph.mega.driver.connection.pool.max_size`
Removed	`fabric.graph.mega.driver.logging.leaked_sessions`
Removed	`fabric.graph.mega.driver.logging.level`
Removed	`fabric.graph.mega.driver.ssl_enabled`
Removed	`fabric.graph.mega.name`
Removed	`fabric.graph.mega.uri`
Removed	`fabric.routing.servers`
Removed	`fabric.routing.ttl`
Removed	`fabric.stream.batch_size`
Removed	`fabric.stream.buffer.low_watermark`
Removed	`fabric.stream.buffer.size`
Removed	`fabric.stream.concurrency`
Removed	`metrics.bolt.messages.enabled`
Removed	`metrics.cypher.replanning.enabled`
Removed	`metrics.jvm.buffers.enabled`

Status	4.4 Name
Removed	`metrics.jvm.file.descriptors.enabled`
Removed	`metrics.jvm.gc.enabled`
Removed	`metrics.jvm.heap.enabled`
Removed	`metrics.jvm.memory.enabled`
Removed	`metrics.jvm.pause_time.enabled`
Removed	`metrics.jvm.threads.enabled`
Removed	`metrics.namespaces.enabled`
Removed	`metrics.neo4j.causal_clustering.enabled`
Removed	`metrics.neo4j.checkpointing.enabled`
Removed	`metrics.neo4j.counts.enabled`
Removed	`metrics.neo4j.data.counts.enabled`
Removed	`metrics.neo4j.database_operation_count.enabled`
Removed	`metrics.neo4j.logs.enabled`
Removed	`metrics.neo4j.pagecache.enabled`
Removed	`metrics.neo4j.pools.enabled`
Removed	`metrics.neo4j.server.enabled`
Removed	`metrics.neo4j.size.enabled`
Removed	`metrics.neo4j.tx.enabled`
Removed	`unsupported.cypher.enable_extra_semantic_features`
Removed	`unsupported.cypher.parser`
Removed	`unsupported.dbms.block_remote_alias`
Removed	`unsupported.dbms.cypher_ip_blocklist`
Removed	`unsupported.dbms.discoverable_bolt_address`
Removed	`unsupported.dbms.discoverable_bolt_routing_address`
Removed	`unsupported.dbms.http_paths_blacklist`
Removed	`unsupported.dbms.large_cluster.enable`
Removed	`unsupported.dbms.lucene.ephemeral`
Removed	`unsupported.dbms.memory.pagecache.warmup.legacy_profile_loader`
Removed	`unsupported.dbms.recovery.ignore_store_id_validation`
Removed	`unsupported.dbms.reserved.page.header.bytes`
Removed	`unsupported.dbms.storage_engine`
Removed	`unsupported.dbms.tokenscan.log.enabled`

Status	4.4 Name
Removed	`unsupported.dbms.tokenscan.log.prune_threshold`
Removed	`unsupported.dbms.tokenscan.log.rotation_threshold`
Removed	`unsupported.dbms.topology_graph.enable`
Removed	`unsupported.dbms.topology_graph_updater.enable`
Removed	`unsupported.dbms.uris.rest`

Renamed

As some of the 4.x configuration settings are now only internal, no documentation is associated. Neo4j does not support nor recommend the use of internal settings.

Status	4.4 Name	5 Name
Renamed	`causal_clustering.akka_actor_system_restarter.initial_delay`	`internal.dbms.cluster.discovery.akka.actor_system_restarter.initial_delay`
Renamed	`causal_clustering.akka_actor_system_restarter.max_acceptable_failures`	`internal.dbms.cluster.discovery.akka.actor_system_restarter.max_acceptable_failures`
Renamed	`causal_clustering.akka_actor_system_restarter.max_delay`	`internal.dbms.cluster.discovery.akka.actor_system_restarter.max_delay`
Renamed	`causal_clustering.catch_up_client_inactivity_timeout`	`dbms.cluster.catchup.client_inactivity_timeout`
Renamed	`causal_clustering.catchup_batch_size`	`internal.dbms.cluster.raft.log_shipping.batch_size`
Renamed	`causal_clustering.cluster_allow_reads_on_followers`	`dbms.routing.reads_on_primaries_enabled`
Renamed	`causal_clustering.cluster_allow_reads_on_leader`	`dbms.routing.reads_on_writers_enabled`
Renamed	`causal_clustering.cluster_binding_retry_timeout`	`internal.dbms.cluster.discovery.akka.binding_retry_timeout`
Renamed	`causal_clustering.cluster_binding_timeout`	`dbms.cluster.raft.binding_timeout`
Renamed	`causal_clustering.cluster_id_publish_timeout`	`internal.dbms.cluster.discovery.akka.bootstrap_publish_timeout`
Renamed	`causal_clustering.cluster_info_polling_max_wait`	`internal.dbms.cluster.info_service_cache_timeout`
Renamed	`causal_clustering.command_applier_parallelism`	`internal.server.cluster.raft.apply.parallelism`
Renamed	`causal_clustering.connect_randomly_to_server_group`	`server.cluster.catchup.connect_randomly_to_server_group`
Renamed	`causal_clustering.discovery_advertised_address`	`server.discovery.advertised_address`
Renamed	`causal_clustering.discovery_listen_address`	`server.discovery.listen_address`
Renamed	`causal_clustering.discovery_resolution_retry_interval`	`internal.dbms.cluster.discovery.resolution_retry_interval`
Renamed	`causal_clustering.discovery_resolution_timeout`	`internal.dbms.cluster.discovery.resolution_timeout`

Status	4.4 Name	5 Name
Renamed	causal_clustering.discovery_type	dbms.cluster.discovery.type
Renamed	causal_clustering.election_failure_detection_window	dbms.cluster.raft.election_failure_detection_window
Renamed	causal_clustering.global_session_tracker_state_size	internal.dbms.cluster.raft.state_size.global_session_tracker
Renamed	causal_clustering.handshake_timeout	dbms.cluster.network.handshake_timeout
Renamed	causal_clustering.in_flight_cache.max_bytes	db.cluster.raft.log_shipping.buffer.max_bytes
Renamed	causal_clustering.in_flight_cache.max_entries	db.cluster.raft.log_shipping.buffer.max_entries
Renamed	causal_clustering.in_flight_cache.type	internal.db.cluster.raft.log_shipping.buffer.type
Renamed	causal_clustering.initial_discovery_members	dbms.cluster.discovery.endpoints
Renamed	causal_clustering.join_catch_up_max_lag	dbms.cluster.raft.membership.join_max_lag
Renamed	causal_clustering.join_catch_up_timeout	dbms.cluster.raft.membership.join_timeout
Renamed	causal_clustering.kubernetes.address	dbms.kubernetes.address
Renamed	causal_clustering.kubernetes.ca_crt	dbms.kubernetes.ca_crt
Renamed	causal_clustering.kubernetes.cluster_domain	dbms.kubernetes.cluster_domain
Renamed	causal_clustering.kubernetes.label_selector	dbms.kubernetes.label_selector
Renamed	causal_clustering.kubernetes.namespace	dbms.kubernetes.namespace
Renamed	causal_clustering.kubernetes.service_port_name	dbms.kubernetes.service_port_name
Renamed	causal_clustering.kubernetes.token	dbms.kubernetes.token
Renamed	causal_clustering.last_applied_state_size	internal.dbms.cluster.raft.state_size.last_applied
Renamed	causal_clustering.leader_failure_detection_window	dbms.cluster.raft.leader_failure_detection_window
Renamed	causal_clustering.leader_transfer_interval	internal.dbms.cluster.raft.leader_transfer.interval
Renamed	causal_clustering.leader_transfer_member_backoff	internal.dbms.cluster.raft.leader_transfer.member_backoff
Renamed	causal_clustering.leader_transfer_timeout	internal.dbms.cluster.raft.leader_transfer.timeout
Renamed	causal_clustering.leadership_balancing	dbms.cluster.raft.leader_transfer.balancing_strategy
Renamed	causal_clustering.leadership_priority_group	db.cluster.raft.leader_transfer.priority_group
Renamed	causal_clustering.leadership_priority_group.test	db.cluster.raft.leader_transfer.priority_group.test
Renamed	causal_clustering.load_balancing.config.server_policies.test	dbms.routing.load_balancing.config.server_policies.test
Renamed	causal_clustering.load_balancing.plugin	dbms.routing.load_balancing.plugin
Renamed	causal_clustering.load_balancing.shuffle	dbms.routing.load_balancing.shuffle_enabled

Status	4.4 Name	5 Name
Renamed	causal_clustering.log_shipping_max_lag	internal.db.cluster.raft.log_shipping.max_lag
Renamed	causal_clustering.log_shipping_retry_timeout	internal.db.cluster.raft.log_shipping.retry_timeout
Renamed	causal_clustering.max_commits_delay_id_reuse	internal.dbms.cluster.raft.id_reuse.max_commits
Renamed	causal_clustering.max_raft_channels	dbms.cluster.raft.client.max_channels
Renamed	causal_clustering.max_time_delay_id_reuse	internal.dbms.cluster.raft.id_reuse.max_time
Renamed	causal_clustering.middleware.akka.allow_any_core_to_bootstrap	internal.dbms.cluster.discovery.akka.any_core_to_bootstrap_enabled
Renamed	causal_clustering.middleware.akka.bind_timeout	internal.dbms.cluster.discovery.akka.bind_timeout
Renamed	causal_clustering.middleware.akka.cluster.seed_node_timeout	internal.dbms.cluster.discovery.akka.seed_node_timeout
Renamed	causal_clustering.middleware.akka.cluster.seed_node_timeout_on_first_start	internal.dbms.cluster.discovery.akka.seed_node_timeout_on_first_start
Renamed	causal_clustering.middleware.akka.connection_timeout	internal.dbms.cluster.discovery.akka.connection_timeout
Renamed	causal_clustering.middleware.akka.default_parallelism	internal.dbms.cluster.discovery.akka.default_parallelism
Renamed	causal_clustering.middleware.akka.down_unreachable_on_new_joiner	internal.dbms.cluster.discovery.akka.down_unreachable_on_new_joiner
Renamed	causal_clustering.middleware.akka.external_config	internal.dbms.cluster.discovery.akka.external_config_path
Renamed	causal_clustering.middleware.akka.failure_detector.acceptable_heartbeat_pause	internal.dbms.cluster.discovery.akka.failure_detector.acceptable_heartbeat_pause
Renamed	causal_clustering.middleware.akka.failure_detector.expected_response_after	internal.dbms.cluster.discovery.akka.failure_detector.expected_response_after
Renamed	causal_clustering.middleware.akka.failure_detector.heartbeat_interval	internal.dbms.cluster.discovery.akka.failure_detector.heartbeat_interval
Renamed	causal_clustering.middleware.akka.failure_detector.max_sample_size	internal.dbms.cluster.discovery.akka.failure_detector.max_sample_size
Renamed	causal_clustering.middleware.akka.failure_detector.min_std_deviation	internal.dbms.cluster.discovery.akka.failure_detector.min_std_deviation
Renamed	causal_clustering.middleware.akka.failure_detector.monitored_by_nr_of_members	internal.dbms.cluster.discovery.akka.failure_detector.monitored_by_nr_of_members
Renamed	causal_clustering.middleware.akka.failure_detector.threshold	internal.dbms.cluster.discovery.akka.failure_detector.threshold
Renamed	causal_clustering.middleware.akka.handshake_timeout	internal.dbms.cluster.discovery.akka.handshake_timeout
Renamed	causal_clustering.middleware.akka.shutdown_timeout	internal.dbms.cluster.discovery.akka.shutdown_timeout
Renamed	causal_clustering.middleware.akka.sink_parallelism	internal.dbms.cluster.discovery.akka.sink_parallelism
Renamed	causal_clustering.middleware.logging.level	dbms.cluster.discovery.log_level
Renamed	causal_clustering.min_time_delay_id_reuse	internal.dbms.cluster.raft.id_reuse.min_time

Status	4.4 Name	5 Name
Renamed	causal_clustering.minimum_core_cluster_size_at_formation	dbms.cluster.minimum_initial_system_primaries_count
Renamed	causal_clustering.minimum_core_cluster_size_at_runtime	internal.db.cluster.raft.minimum_voting_members
Renamed	causal_clustering.protocol_implementations.compression	dbms.cluster.network.supported_compression_algos
Renamed	causal_clustering.pull_interval	db.cluster.catchup.pull_interval
Renamed	causal_clustering.raft_advertised_address	server.cluster.raft.advertised_address
Renamed	causal_clustering.raft_group_graveyard_state_size	internal.dbms.cluster.raft.state_size.group_graveyard
Renamed	causal_clustering.raft_handler_parallelism	internal.server.cluster.raft.message_handler.parallelism
Renamed	causal_clustering.raft_in_queue_max_batch	internal.db.cluster.raft.in_queue.batch.max_entries
Renamed	causal_clustering.raft_in_queue_max_batch_bytes	db.cluster.raft.in_queue.batch.max_bytes
Renamed	causal_clustering.raft_in_queue_max_bytes	db.cluster.raft.in_queue.max_bytes
Renamed	causal_clustering.raft_in_queue_size	internal.db.cluster.raft.in_queue.max_entries
Renamed	causal_clustering.raft_listen_address	server.cluster.raft.listen_address
Renamed	causal_clustering.raft_log_entry_prefetch_buffer.max_entries	db.cluster.raft.apply.buffer.max_entries
Renamed	causal_clustering.raft_log_prune_strategy	db.cluster.raft.log.prune_strategy
Renamed	causal_clustering.raft_log_pruning_frequency	dbms.cluster.raft.log.pruning_frequency
Renamed	causal_clustering.raft_log_reader_pool_size	dbms.cluster.raft.log.reader_pool_size
Renamed	causal_clustering.raft_log_rotation_size	dbms.cluster.raft.log.rotation_size
Renamed	causal_clustering.raft_membership_state_size	internal.dbms.cluster.raft.state_size.membership
Renamed	causal_clustering.raft_messages_log_enable	internal.dbms.cluster.raft.messages_log.enabled
Renamed	causal_clustering.raft_messages_log_path	internal.dbms.cluster.raft.messages_log.path
Renamed	causal_clustering.raft_term_state_size	internal.dbms.cluster.raft.state_size.term
Renamed	causal_clustering.raft_vote_state_size	internal.dbms.cluster.raft.state_size.vote
Renamed	causal_clustering.read_replica_transaction_applier_batch_size	internal.db.cluster.catchup.in_queue.batch.max_entries
Renamed	causal_clustering.read_replica_transaction_applier_max_queue_size	internal.db.cluster.catchup.in_queue.batch.max_size
Renamed	causal_clustering.replicated_lease_state_size	internal.dbms.cluster.raft.state_size.replicated_lease
Renamed	causal_clustering.replication_leader_await_timeout	internal.dbms.cluster.raft.replication.leader_await_timeout
Renamed	causal_clustering.replication_retry_timeout_base	internal.dbms.cluster.raft.replication.retry_timeout_base

Status	4.4 Name	5 Name
Renamed	causal_clustering.replication_retry_timeout_limit	internal.dbms.cluster.raft.replication.retry_timeout_limit
Renamed	causal_clustering.seed_validation_timeout	internal.dbms.cluster.seed_validation_timeout
Renamed	causal_clustering.server_groups	server.groups
Renamed	causal_clustering.state_machine_apply_max_batch_size	internal.dbms.cluster.raft.apply.max_batch_size
Renamed	causal_clustering.state_machine_flush_window_size	internal.dbms.cluster.raft.apply.flush_window_size
Renamed	causal_clustering.status_throughput_window	internal.dbms.status.throughput_window
Renamed	causal_clustering.store_copy_max_retry_time_per_request	dbms.cluster.store_copy.max_retry_time_per_request
Renamed	causal_clustering.store_copy_parallelism	internal.server.cluster.store_copy.parallelism
Renamed	causal_clustering.store_size_service_cache_timeout	internal.dbms.cluster.store_size_service_cache_timeout
Renamed	causal_clustering.topology_graph.default_num_primaries	initial.dbms.default_primaries_count
Renamed	causal_clustering.topology_graph.default_num_secondaries	initial.dbms.default_secondaries_count
Renamed	causal_clustering.transaction_advertised_address	server.cluster.advertised_address
Renamed	causal_clustering.transaction_listen_address	server.cluster.listen_address
Renamed	causal_clustering.unknown_address_logging_throttle	internal.dbms.cluster.raft.unknown_address_logging_throttle
Renamed	causal_clustering.upstream_selection_strategy	server.cluster.catchup.upstream_strategy
Renamed	causal_clustering.use_native_transport	server.cluster.network.native_transport_enabled
Renamed	causal_clustering.user_defined_upstream_strategy	server.cluster.catchup.user_defined_upstream_strategy
Renamed	clients.allow_telemetry	client.allow_telemetry
Renamed	cypher.forbid_exhaustive_shortestpath	dbms.cypher.forbid_exhaustive_shortestpath
Renamed	cypher.forbid_shortestpath_common_nodes	dbms.cypher.forbid_shortestpath_common_nodes
Renamed	cypher.hints_error	dbms.cypher.hints_error
Renamed	cypher.lenient_create_relationship	dbms.cypher.lenient_create_relationship
Renamed	cypher.min_replan_interval	dbms.cypher.min_replan_interval
Renamed	cypher.planner	dbms.cypher.planner
Renamed	cypher.statistics_divergence_threshold	dbms.cypher.statistics_divergence_threshold
Renamed	dbms.backup.enabled	server.backup.enabled
Renamed	dbms.backup.listen_address	server.backup.listen_address
Renamed	dbms.checkpoint	db.checkpoint

Status	4.4 Name	5 Name
Renamed	`dbms.checkpoint.interval.time`	`db.checkpoint.interval.time`
Renamed	`dbms.checkpoint.interval.tx`	`db.checkpoint.interval.tx`
Renamed	`dbms.checkpoint.interval.volume`	`db.checkpoint.interval.volume`
Renamed	`dbms.checkpoint.iops.limit`	`db.checkpoint.iops.limit`
Renamed	`dbms.config.strict_validation`	`server.config.strict_validation.enabled`
Renamed	`dbms.connector.bolt.advertised_address`	`server.bolt.advertised_address`
Renamed	`dbms.connector.bolt.connection_keep_alive`	`server.bolt.connection_keep_alive`
Renamed	`dbms.connector.bolt.connection_keep_alive_for_requests`	`server.bolt.connection_keep_alive_for_requests`
Renamed	`dbms.connector.bolt.connection_keep_alive_probes`	`server.bolt.connection_keep_alive_probes`
Renamed	`dbms.connector.bolt.connection_keep_alive_streaming_scheduling_interval`	`server.bolt.connection_keep_alive_streaming_scheduling_interval`
Renamed	`dbms.connector.bolt.enabled`	`server.bolt.enabled`
Renamed	`dbms.connector.bolt.listen_address`	`server.bolt.listen_address`
Renamed	`dbms.connector.bolt.ocsp_stapling_enabled`	`server.bolt.ocsp_stapling_enabled`
Renamed	`dbms.connector.bolt.tcp_keep_alive`	`internal.server.bolt.tcp_keep_alive`
Renamed	`dbms.connector.bolt.thread_pool_keep_alive`	`server.bolt.thread_pool_keep_alive`
Renamed	`dbms.connector.bolt.thread_pool_max_size`	`server.bolt.thread_pool_max_size`
Renamed	`dbms.connector.bolt.thread_pool_min_size`	`server.bolt.thread_pool_min_size`
Renamed	`dbms.connector.bolt.tls_level`	`server.bolt.tls_level`
Renamed	`dbms.connector.bolt.unsupported_thread_pool_queue_size`	`internal.server.bolt.thread_pool_queue_size`
Renamed	`dbms.connector.bolt.unsupported_thread_pool_shutdown_wait_time`	`internal.server.bolt.thread_pool_shutdown_wait_time`
Renamed	`dbms.connector.bolt.unsupported_unauth_connection_timeout`	`internal.server.bolt.unauth_connection_timeout`
Renamed	`dbms.connector.bolt.unsupported_unauth_max_inbound_bytes`	`internal.server.bolt.unauth_max_inbound_bytes`
Renamed	`dbms.connector.http.advertised_address`	`server.http.advertised_address`
Renamed	`dbms.connector.http.enabled`	`server.http.enabled`
Renamed	`dbms.connector.http.listen_address`	`server.http.listen_address`
Renamed	`dbms.connector.https.advertised_address`	`server.https.advertised_address`
Renamed	`dbms.connector.https.enabled`	`server.https.enabled`
Renamed	`dbms.connector.https.listen_address`	`server.https.listen_address`
Renamed	`dbms.databases.default_to_read_only`	`server.databases.default_to_read_only`
Renamed	`dbms.databases.read_only`	`server.databases.read_only`

Status	4.4 Name	5 Name
Renamed	dbms.databases.writable	server.databases.writable
Renamed	dbms.default_advertised_address	server.default_advertised_address
Renamed	dbms.default_database	initial.dbms.default_database
Renamed	dbms.default_listen_address	server.default_listen_address
Renamed	dbms.directories.cluster_state	server.directories.cluster_state
Renamed	dbms.directories.data	server.directories.data
Renamed	dbms.directories.dumps.root	server.directories.dumps.root
Renamed	dbms.directories.import	server.directories.import
Renamed	dbms.directories.lib	server.directories.lib
Renamed	dbms.directories.licenses	server.directories.licenses
Renamed	dbms.directories.logs	server.directories.logs
Renamed	dbms.directories.metrics	server.directories.metrics
Renamed	dbms.directories.neo4j_home	server.directories.neo4j_home
Renamed	dbms.directories.plugins	server.directories.plugins
Renamed	dbms.directories.run	server.directories.run
Renamed	dbms.directories.script.root	server.directories.script.root
Renamed	dbms.directories.transaction.logs.root	server.directories.transaction.logs.root
Renamed	dbms.dynamic.setting.allowlist	server.dynamic.setting.allowlist
Renamed	dbms.dynamic.setting.whitelist	server.dynamic.setting.allowlist
Renamed	dbms.filewatcher.enabled	db.filewatcher.enabled
Renamed	dbms.import.csv.buffer_size	db.import.csv.buffer_size
Renamed	dbms.import.csv.legacy_quote_escaping	db.import.csv.legacy_quote_escaping
Renamed	dbms.index.fulltext.default_analyzer	db.index.fulltext.default_analyzer
Renamed	dbms.index.fulltext.eventually_consistent	db.index.fulltext.eventually_consistent
Renamed	dbms.index.fulltext.eventually_consistent_index_update_queue_max_length	db.index.fulltext.eventually_consistent_index_update_queue_max_length
Renamed	dbms.index_sampling.background_enabled	db.index_sampling.background_enabled
Renamed	dbms.index_sampling.sample_size_limit	db.index_sampling.sample_size_limit
Renamed	dbms.index_sampling.update_percentage	db.index_sampling.update_percentage
Renamed	dbms.init_file	internal.dbms.init_file
Renamed	dbms.jvm.additional	server.jvm.additional
Renamed	dbms.lock.acquisition.timeout	db.lock.acquisition.timeout
Renamed	dbms.log_inconsistent_data_deletion	internal.dbms.log_inconsistent_data_deletion

Status	4.4 Name	5 Name
Renamed	dbms.logs.gc.enabled	server.logs.gc.enabled
Renamed	dbms.logs.gc.options	server.logs.gc.options
Renamed	dbms.logs.gc.rotation.keep_number	server.logs.gc.rotation.keep_number
Renamed	dbms.logs.gc.rotation.size	server.logs.gc.rotation.size
Renamed	dbms.logs.query.early_raw_logging_enabled	db.logs.query.early_raw_logging_enabled
Renamed	dbms.logs.query.enabled	db.logs.query.enabled
Renamed	dbms.logs.query.max_parameter_length	db.logs.query.max_parameter_length
Renamed	dbms.logs.query.obfuscate_literals	db.logs.query.obfuscate_literals
Renamed	dbms.logs.query.parameter_logging_enabled	db.logs.query.parameter_logging_enabled
Renamed	dbms.logs.query.plan_description_enabled	db.logs.query.plan_description_enabled
Renamed	dbms.logs.query.threshold	db.logs.query.threshold
Renamed	dbms.logs.query.transaction.enabled	db.logs.query.transaction.enabled
Renamed	dbms.logs.query.transaction.threshold	db.logs.query.transaction.threshold
Renamed	dbms.memory.heap.initial_size	server.memory.heap.initial_size
Renamed	dbms.memory.heap.max_size	server.memory.heap.max_size
Renamed	dbms.memory.off_heap.block_cache_size	server.memory.off_heap.block_cache_size
Renamed	dbms.memory.off_heap.max_cacheable_block_size	server.memory.off_heap.max_cacheable_block_size
Renamed	dbms.memory.off_heap.max_size	server.memory.off_heap.max_size
Renamed	dbms.memory.pagecache.directio	server.memory.pagecache.directio
Renamed	dbms.memory.pagecache.flush.buffer.enabled	server.memory.pagecache.flush.buffer.enabled
Renamed	dbms.memory.pagecache.flush.buffer.size_in_pages	server.memory.pagecache.flush.buffer.size_in_pages
Renamed	dbms.memory.pagecache.scan.prefetchers	server.memory.pagecache.scan.prefetchers
Renamed	dbms.memory.pagecache.size	server.memory.pagecache.size
Renamed	dbms.memory.pagecache.warmup.enable	db.memory.pagecache.warmup.enable
Renamed	dbms.memory.pagecache.warmup.preload	db.memory.pagecache.warmup.preload
Renamed	dbms.memory.pagecache.warmup.preload.allowlist	db.memory.pagecache.warmup.preload.allowlist
Renamed	dbms.memory.pagecache.warmup.preload.whitelist	db.memory.pagecache.warmup.preload.allowlist
Renamed	dbms.memory.pagecache.warmup.profile.interval	db.memory.pagecache.warmup.profile.interval
Renamed	dbms.memory.transaction.database_max_size	db.memory.transaction.total.max
Renamed	dbms.memory.transaction.global_max_size	dbms.memory.transaction.total.max
Renamed	dbms.memory.transaction.max_size	db.memory.transaction.max
Renamed	dbms.panic.shutdown_on_panic	server.panic.shutdown_on_panic

Status	4.4 Name	5 Name
Renamed	`dbms.query_cache_size`	`server.db.query_cache_size`
Renamed	`dbms.read_only`	`server.databases.default_to_read_only`
Renamed	`dbms.reconciler.max_backoff`	`internal.dbms.reconciler.max_backoff`
Renamed	`dbms.reconciler.max_parallelism`	`internal.dbms.reconciler.max_parallelism`
Renamed	`dbms.reconciler.may_retry`	`internal.dbms.reconciler.retry_enabled`
Renamed	`dbms.reconciler.min_backoff`	`internal.dbms.reconciler.min_backoff`
Renamed	`dbms.recovery.fail_on_missing_files`	`db.recovery.fail_on_missing_files`
Renamed	`dbms.relationship_grouping_threshold`	`db.relationship_grouping_threshold`
Renamed	`dbms.routing.advertised_address`	`server.routing.advertised_address`
Renamed	`dbms.routing.driver.event_loop_count`	`internal.dbms.routing.driver.event_loop_count`
Renamed	`dbms.routing.driver.idle_check_interval`	`internal.dbms.routing.driver.idle_check_interval`
Renamed	`dbms.routing.driver.logging.leaked_sessions`	`internal.dbms.routing.driver.logging.leaked_sessions`
Renamed	`dbms.routing.driver.timeout`	`internal.dbms.routing.driver.timeout`
Renamed	`dbms.routing.listen_address`	`server.routing.listen_address`
Renamed	`dbms.security.causal_clustering_status_auth_enabled`	`dbms.security.cluster_status_auth_enabled`
Renamed	`dbms.security.http_auth_whitelist`	`dbms.security.http_auth_allowlist`
Renamed	`dbms.security.procedures.whitelist`	`dbms.security.procedures.allowlist`
Renamed	`dbms.shutdown_transaction_end_timeout`	`db.shutdown_transaction_end_timeout`
Renamed	`dbms.store.files.preallocate`	`db.store.files.preallocate`
Renamed	`dbms.threads.worker_count`	`server.threads.worker_count`
Renamed	`dbms.track_query_cpu_time`	`db.track_query_cpu_time`
Renamed	`dbms.transaction.bookmark_ready_timeout`	`db.transaction.bookmark_ready_timeout`
Renamed	`dbms.transaction.concurrent.maximum`	`db.transaction.concurrent.maximum`
Renamed	`dbms.transaction.monitor.check.interval`	`db.transaction.monitor.check.interval`
Renamed	`dbms.transaction.sampling.percentage`	`db.transaction.sampling.percentage`
Renamed	`dbms.transaction.timeout`	`db.transaction.timeout`
Renamed	`dbms.transaction.tracing.level`	`db.transaction.tracing.level`
Renamed	`dbms.tx_log.buffer.size`	`db.tx_log.buffer.size`
Renamed	`dbms.tx_log.preallocate`	`db.tx_log.preallocate`
Renamed	`dbms.tx_log.rotation.retention_policy`	`db.tx_log.rotation.retention_policy`
Renamed	`dbms.tx_log.rotation.size`	`db.tx_log.rotation.size`

Status	4.4 Name	5 Name
Renamed	dbms.tx_state.memory_allocation	db.tx_state.memory_allocation
Renamed	dbms.unmanaged_extension_classes	server.unmanaged_extension_classes
Renamed	dbms.upgrade_max_processors	internal.dbms.upgrade_max_processors
Renamed	dbms.windows_service_name	server.windows_service_name
Renamed	metrics.csv.enabled	server.metrics.csv.enabled
Renamed	metrics.csv.interval	server.metrics.csv.interval
Renamed	metrics.csv.rotation.compression	server.metrics.csv.rotation.compression
Renamed	metrics.csv.rotation.keep_number	server.metrics.csv.rotation.keep_number
Renamed	metrics.csv.rotation.size	server.metrics.csv.rotation.size
Renamed	metrics.enabled	server.metrics.enabled
Renamed	metrics.filter	server.metrics.filter
Renamed	metrics.graphite.enabled	server.metrics.graphite.enabled
Renamed	metrics.graphite.interval	server.metrics.graphite.interval
Renamed	metrics.graphite.server	server.metrics.graphite.server
Renamed	metrics.jmx.enabled	server.metrics.jmx.enabled
Renamed	metrics.prefix	server.metrics.prefix
Renamed	metrics.prometheus.enabled	server.metrics.prometheus.enabled
Renamed	metrics.prometheus.endpoint	server.metrics.prometheus.endpoint
Renamed	systemdb.secrets.key.name	dbms.security.key.name
Renamed	systemdb.secrets.keystore.password	dbms.security.keystore.password
Renamed	systemdb.secrets.keystore.path	dbms.security.keystore.path
Renamed	unsupported.causal_clustering.cluster_status_request_maximum_wait	internal.dbms.cluster.raft.status_check_max_response_wait
Renamed	unsupported.causal_clustering.experimental_catchup_protocol_enabled	internal.dbms.cluster.experimental_protocol_version.catchup_enabled
Renamed	unsupported.causal_clustering.experimental_raft_protocol_enabled	internal.dbms.cluster.experimental_protocol_version.raft_enabled
Renamed	unsupported.causal_clustering.inbound_connection_initialization_logging_enabled	internal.dbms.cluster.network.inbound_connection_initialization_logging_enabled
Renamed	unsupported.consistency_checker.fail_fast_threshold	internal.consistency_checker.fail_fast_threshold
Renamed	unsupported.consistency_checker.memory_limit_factor	internal.consistency_checker.memory_limit_factor
Renamed	unsupported.cypher.compiler_tracing	internal.cypher.compiler_tracing
Renamed	unsupported.cypher.enable_runtime_monitors	internal.cypher.enable_runtime_monitors
Renamed	unsupported.cypher.expression_engine	internal.cypher.expression_engine

Status	4.4 Name	5 Name
Renamed	unsupported.cypher.expression_recompilation_limit	internal.cypher.expression_recompilation_limit
Renamed	unsupported.cypher.idp_solver_duration_threshold	internal.cypher.idp_solver_duration_threshold
Renamed	unsupported.cypher.idp_solver_table_threshold	internal.cypher.idp_solver_table_threshold
Renamed	unsupported.cypher.non_indexed_label_warning_threshold	internal.cypher.non_indexed_label_warning_threshold
Renamed	unsupported.cypher.number_of_workers	internal.cypher.number_of_workers
Renamed	unsupported.cypher.pipelined.batch_size_big	internal.cypher.pipelined.batch_size_big
Renamed	unsupported.cypher.pipelined.batch_size_small	internal.cypher.pipelined.batch_size_small
Renamed	unsupported.cypher.pipelined.enable_runtime_trace	internal.cypher.pipelined.enable_runtime_trace
Renamed	unsupported.cypher.pipelined.operator_engine	internal.cypher.pipelined.operator_engine
Renamed	unsupported.cypher.pipelined.operator_fusion_over_pipeline_limit	internal.cypher.pipelined.operator_fusion_over_pipeline_limit
Renamed	unsupported.cypher.pipelined.runtime_trace_path	internal.cypher.pipelined.runtime_trace_path
Renamed	unsupported.cypher.pipelined_interpreted_pipes_fallback	internal.cypher.pipelined_interpreted_pipes_fallback
Renamed	unsupported.cypher.planning_point_indexes_enabled	internal.cypher.planning_point_indexes_enabled
Renamed	unsupported.cypher.planning_range_indexes_enabled	internal.cypher.planning_range_indexes_enabled
Renamed	unsupported.cypher.planning_text_indexes_enabled	internal.cypher.planning_text_indexes_enabled
Renamed	unsupported.cypher.replan_algorithm	internal.cypher.replan_algorithm
Renamed	unsupported.cypher.runtime	internal.cypher.runtime
Renamed	unsupported.cypher.splitting_top_behavior	internal.cypher.splitting_top_behavior
Renamed	unsupported.cypher.statistics_divergence_target	internal.cypher.statistics_divergence_target
Renamed	unsupported.cypher.target_replan_interval	internal.cypher.target_replan_interval
Renamed	unsupported.datacollector.max_query_text_size	internal.datacollector.max_query_text_size
Renamed	unsupported.datacollector.max_recent_query_count	internal.datacollector.max_recent_query_count
Renamed	unsupported.dbms.block_alter_database	internal.dbms.block_alter_database
Renamed	unsupported.dbms.block_create_drop_database	internal.dbms.block_create_drop_database
Renamed	unsupported.dbms.block_size.array_properties	internal.dbms.block_size.array_properties
Renamed	unsupported.dbms.block_size.labels	internal.dbms.block_size.labels
Renamed	unsupported.dbms.block_size.strings	internal.dbms.block_size.strings
Renamed	unsupported.dbms.block_start_stop_database	internal.dbms.block_start_stop_database

Status	4.4 Name	5 Name
Renamed	unsupported.dbms.bolt.inbound_message_throttle.high_watermark	internal.dbms.bolt.inbound_message_throttle.high_watermark
Renamed	unsupported.dbms.bolt.inbound_message_throttle.low_watermark	internal.dbms.bolt.inbound_message_throttle.low_watermark
Renamed	unsupported.dbms.bolt.netty_message_merge_cumulator	internal.dbms.bolt.netty_message_merge_cumulator
Renamed	unsupported.dbms.bolt.netty_server_shutdown_quiet_period	internal.dbms.bolt.netty_server_shutdown_quiet_period
Renamed	unsupported.dbms.bolt.netty_server_shutdown_timeout	internal.dbms.bolt.netty_server_shutdown_timeout
Renamed	unsupported.dbms.bolt.netty_server_use_epoll	internal.dbms.bolt.netty_server_use_native_transport
Renamed	unsupported.dbms.bolt.outbound_buffer_throttle	internal.dbms.bolt.outbound_buffer_throttle
Renamed	unsupported.dbms.bolt.outbound_buffer_throttle.high_watermark	internal.dbms.bolt.outbound_buffer_throttle.high_watermark
Renamed	unsupported.dbms.bolt.outbound_buffer_throttle.low_watermark	internal.dbms.bolt.outbound_buffer_throttle.low_watermark
Renamed	unsupported.dbms.bolt.outbound_buffer_throttle.max_duration	internal.dbms.bolt.outbound_buffer_throttle.max_duration
Renamed	unsupported.dbms.checkpoint_log.rotation.keep.files	internal.db.checkpoint_log.rotation.keep.files
Renamed	unsupported.dbms.checkpoint_log.rotation.size	internal.db.checkpoint_log.rotation.size
Renamed	unsupported.dbms.config.command_evaluation_timeout	internal.dbms.config.command_evaluation_timeout
Renamed	unsupported.dbms.counts_store_rotation_timeout	internal.dbms.counts_store_rotation_timeout
Renamed	unsupported.dbms.db.spatial.crs.wgs-84.max	internal.dbms.db.spatial.crs.wgs-84.max
Renamed	unsupported.dbms.db.spatial.crs.wgs-84.min	internal.dbms.db.spatial.crs.wgs-84.min
Renamed	unsupported.dbms.debug.page_cache_tracer_speed_reporting_threshold	internal.dbms.debug.page_cache_tracer_speed_reporting_threshold
Renamed	unsupported.dbms.debug.print_page_buffer_allocation_trace	internal.dbms.debug.print_page_buffer_allocation_trace
Renamed	unsupported.dbms.debug.trace_cursors	internal.dbms.debug.trace_cursors
Renamed	unsupported.dbms.debug.trace_tx_statement	internal.dbms.debug.trace_tx_statement
Renamed	unsupported.dbms.debug.track_cursor_close	internal.dbms.debug.track_cursor_close
Renamed	unsupported.dbms.debug.track_tx_statement_close	internal.dbms.debug.track_tx_statement_close
Renamed	unsupported.dbms.directories.auth	internal.server.directories.auth
Renamed	unsupported.dbms.directories.databases.root	internal.server.directories.databases.root
Renamed	unsupported.dbms.directories.pid_file	internal.server.directories.pid_file
Renamed	unsupported.dbms.directories.scripts	internal.server.directories.scripts
Renamed	unsupported.dbms.directories.windows_tools	internal.server.directories.windows_tools

Status	4.4 Name	5 Name
Renamed	unsupported.dbms.dump_diagnostics	internal.dbms.dump_diagnostics
Renamed	unsupported.dbms.enable_transaction_heap_allocation_tracking	internal.dbms.enable_transaction_heap_allocation_tracking
Renamed	unsupported.dbms.executiontime_limit.time	internal.dbms.executiontime_limit.time
Renamed	unsupported.dbms.extra_lock_verification	internal.dbms.extra_lock_verification
Renamed	unsupported.dbms.force_small_id_cache	internal.dbms.force_small_id_cache
Renamed	unsupported.dbms.idgenerator.log.enabled	internal.dbms.idgenerator.log.enabled
Renamed	unsupported.dbms.idgenerator.log.prune_threshold	internal.dbms.idgenerator.log.prune_threshold
Renamed	unsupported.dbms.idgenerator.log.rotation_threshold	internal.dbms.idgenerator.log.rotation_threshold
Renamed	unsupported.dbms.index.archive_failed	internal.dbms.index.archive_failed
Renamed	unsupported.dbms.index.default_fulltext_provider	internal.dbms.index.default_fulltext_provider
Renamed	unsupported.dbms.index.lucene.merge_factor	internal.dbms.index.lucene.merge_factor
Renamed	unsupported.dbms.index.lucene.min_merge	internal.dbms.index.lucene.min_merge
Renamed	unsupported.dbms.index.lucene.nocfs.ratio	internal.dbms.index.lucene.nocfs.ratio
Renamed	unsupported.dbms.index.lucene.population_max_buffered_docs	internal.dbms.index.lucene.population_max_buffered_docs
Renamed	unsupported.dbms.index.lucene.population_ram_buffer_size	internal.dbms.index.lucene.population_ram_buffer_size
Renamed	unsupported.dbms.index.lucene.standard_ram_buffer_size	internal.dbms.index.lucene.standard_ram_buffer_size
Renamed	unsupported.dbms.index.lucene.writer_max_buffered_docs	internal.dbms.index.lucene.writer_max_buffered_docs
Renamed	unsupported.dbms.index.population_batch_max_byte_size	internal.dbms.index.population_batch_max_byte_size
Renamed	unsupported.dbms.index.population_print_debug	internal.dbms.index.population_print_debug
Renamed	unsupported.dbms.index.population_queue_threshold	internal.dbms.index.population_queue_threshold
Renamed	unsupported.dbms.index.populator_block_size	internal.dbms.index.populator_block_size
Renamed	unsupported.dbms.index.populator_merge_factor	internal.dbms.index.populator_merge_factor
Renamed	unsupported.dbms.index.sampling.async_recovery	internal.dbms.index.sampling.async_recovery
Renamed	unsupported.dbms.index.sampling.async_recovery_wait	internal.dbms.index.sampling.async_recovery_wait
Renamed	unsupported.dbms.index.sampling.log_recovered_samples	internal.dbms.index.sampling.log_recovered_samples
Renamed	unsupported.dbms.index.skip_default_indexes_on_creation	internal.dbms.index.skip_default_indexes_on_creation
Renamed	unsupported.dbms.index.spatial.curve.bottom_threshold	internal.dbms.index.spatial.curve.bottom_threshold

Status	4.4 Name	5 Name
Renamed	unsupported.dbms.index.spatial.curve.extra_levels	internal.dbms.index.spatial.curve.extra_levels
Renamed	unsupported.dbms.index.spatial.curve.top_threshold	internal.dbms.index.spatial.curve.top_threshold
Renamed	unsupported.dbms.index_population.parallelism	internal.dbms.index_population.parallelism
Renamed	unsupported.dbms.index_population.workers	internal.dbms.index_population.workers
Renamed	unsupported.dbms.index_sampling.parallelism	internal.dbms.index_sampling.parallelism
Renamed	unsupported.dbms.initial_transaction_heap_grab_size	internal.dbms.initial_transaction_heap_grab_size
Renamed	unsupported.dbms.io.controller.consider.external.enabled	internal.dbms.io.controller.consider.external.enabled
Renamed	unsupported.dbms.kernel_id	internal.dbms.kernel_id
Renamed	unsupported.dbms.lock_manager	internal.dbms.lock_manager
Renamed	unsupported.dbms.lock_manager.verbose_deadlocks	internal.dbms.lock_manager.verbose_deadlocks
Renamed	unsupported.dbms.logs.query.heap_dump_enabled	internal.dbms.logs.query.heap_dump_enabled
Renamed	unsupported.dbms.loopback_delete	internal.dbms.loopback_delete
Renamed	unsupported.dbms.loopback_enabled	internal.dbms.loopback_enabled
Renamed	unsupported.dbms.loopback_file	internal.dbms.loopback_file
Renamed	unsupported.dbms.max_http_request_header_size	internal.dbms.max_http_request_header_size
Renamed	unsupported.dbms.max_http_response_header_size	internal.dbms.max_http_response_header_size
Renamed	unsupported.dbms.memory.counts_store_max_cached_entries	internal.dbms.memory.counts_store_max_cached_entries
Renamed	unsupported.dbms.memory.managed_network_buffers	internal.dbms.memory.managed_network_buffers
Renamed	unsupported.dbms.page.file.tracer	internal.dbms.page.file.tracer
Renamed	unsupported.dbms.query.snapshot	internal.dbms.query.snapshot
Renamed	unsupported.dbms.query.snapshot.retries	internal.dbms.query.snapshot.retries
Renamed	unsupported.dbms.query_execution_plan_cache_size	internal.dbms.query_execution_plan_cache_size
Renamed	unsupported.dbms.readonly.failover	internal.dbms.readonly.failover
Renamed	unsupported.dbms.recovery.enable_parallelism	internal.dbms.recovery.enable_parallelism
Renamed	unsupported.dbms.report_configuration	internal.dbms.report_configuration
Renamed	unsupported.dbms.security.ldap.authorization.connection_pooling	internal.dbms.security.ldap.authorization.connection_pooling
Renamed	unsupported.dbms.ssl.system.ignore_dot_files	internal.dbms.ssl.system.ignore_dot_files
Renamed	unsupported.dbms.storage.consistency_check_on_apply	internal.dbms.storage.consistency_check_on_apply

Status	4.4 Name	5 Name
Renamed	unsupported.dbms.strictly_prioritize_id_freelist	internal.dbms.strictly_prioritize_id_freelist
Renamed	unsupported.dbms.tracer	internal.dbms.tracer
Renamed	unsupported.dbms.transaction_start_timeout	internal.dbms.transaction_start_timeout
Renamed	unsupported.dbms.tx.logs.dedicated.appender	internal.dbms.tx.logs.dedicated.appender
Renamed	unsupported.dbms.tx_log.fail_on_corrupted_log_files	internal.dbms.tx_log.fail_on_corrupted_log_files
Renamed	unsupported.dbms.tx_log.presketch	internal.dbms.tx_log.presketch
Renamed	unsupported.dbms.upgrade_restriction_enabled	internal.dbms.upgrade_restriction_enabled
Renamed	unsupported.dbms.uris.browser	internal.dbms.uris.browser
Renamed	unsupported.dbms.uris.db	internal.dbms.uris.db
Renamed	unsupported.dbms.uris.dbms	internal.dbms.uris.dbms
Renamed	unsupported.dbms.uris.management	internal.dbms.uris.management
Renamed	unsupported.dbms.wadl_generation_enabled	internal.dbms.wadl_generation_enabled
Renamed	unsupported.tools.batch_inserter.batch_size	internal.tools.batch_inserter.batch_size
Renamed	unsupported.vm_pause_monitor.measurement_duration	internal.vm_pause_monitor.measurement_duration
Renamed	unsupported.vm_pause_monitor.stall_alert_threshold	internal.vm_pause_monitor.stall_alert_threshold

Migrate a standalone server (Helm)

This example shows how to migrate a standalone server deployed on Kubernetes with the *neo4j/neo4j-standalone* Helm chart from version 4.4 to 5.x.

Migration to Neo4j 5 is only supported from 4.4.

It is recommended to read the following pages before continuing:

- Introduction
- Migrate from 4.4
- Planning
- Breaking changes

The following example steps assume that Neo4j DBMS 4.4 is installed with the Helm release name *standalone-4*.

Prepare the 4.4 standalone server for migration

Prepare the 4.4 standalone server for migration by recreating the indexes and the index-backed constraints to match the new index types, and by backing up each of your databases.

Recreate indexes and index-backed constraints

In 5.0, the BTREE index type is no longer available. Therefore, it is recommended to recreate all your BTREE indexes and index-backed constraints with index providers `native-btree-1.0` or `lucene+native-3.0` before switching to 5.x. During the migration, 5.x checks whether each BTREE index and index-backed constraint has an equivalent type of index and provider, and drops them.

What type of index to use instead of BTREE?

In most cases, RANGE indexes can replace BTREE. However, there might be occasions when a different index type is more suitable, such as:

- Use POINT indexes if the property value type is `point` and `distance` or `bounding box` queries are used for the property.
- Use TEXT indexes if the property value type is `text` and the values can be larger than 8Kb.
- Use TEXT indexes if the property value type is `text` and `CONTAINS` and `ENDS WITH` are used in queries for the property.

If more than one of the conditions is true, it is possible to have multiple indexes of different index types on the same schema. For more information on each index type, see Operations Manual 5.0 → Index configuration.

Steps

1. Recreate each of your BTREE indexes on the same schema but using the new type (RANGE, POINT, or TEXT) as per your use case. The following example creates a range index on a single property for all nodes with a particular label:

   ```
   CREATE RANGE INDEX range_index_name FOR (n:Label) ON (n.prop1)
   ```

2. Recreate each of your index-backed constraints with index providers `native-btree-1.0` or `lucene+native-3.0` on the same schema but with the new provider. The following example creates a unique node property constraint on a single property for all nodes with a particular label. The backing index is of type range with `range-1.0` index provider.

   ```
   CREATE CONSTRAINT constraint_with_provider FOR (n:Label) REQUIRE (n.prop1) IS UNIQUE OPTIONS
   {indexProvider: 'range-1.0'}
   ```

3. Run `SHOW INDEXES` to verify that the indexes have been populated and constraints have been created with the correct index provider.

For more information about creating indexes, see Cypher Manual → Creating indexes.

Backup each of your databases

1. To ensure the databases do not get updated during the backup, put them into read-only mode using the Cypher command `ALTER DATABASE <databasename> SET ACCESS READ ONLY`.
2. Create a directory to store backups. This tutorial uses */migration-backups*.
3. Backup each of your databases by choosing one of the options:
   - Enterprise Run the `neo4j-admin backup` command to back up all your databases.

     - All databases that you want to back up must be online.
     - Use the option `--include-metadata=all` to include all roles and users associated with each of your databases:

     ```
     kubectl exec -t -i standalone-4-0 -- neo4j-admin backup --database=*  --backup-dir=/backups
     --include-metadata=all
     ```

     Now copy the backups from the Kubernetes Pod to an intermediate to the `/migration-backups` folder on your local filesystem:

     ```
     kubectl cp standalone-4-0:/backups /path/to/migration-backups
     ```

     The result is a folder for each database, called *<databasename>* and located in the */tmp//migration-backups* folder, and a metadata script for each database, located in */migration-backups/databasename/tools/metadata_script.cypher*. For more information about the `neo4j-admin backup` command, see Operations Manual → Backup an online database.

   - Use the `neo4j-admin dump` command to create an offline `.dump` file of the `system` and `neo4j` database.

     First place the Neo4j standalone server in offline maintenance mode:

     ```
     helm upgrade standalone-4 neo4j/neo4j-standalone --reuse-values --set
     neo4j.offlineMaintenanceModeEnabled=true
     ```

     Perform an offline dump of the databases:

     ```
     kubectl exec -t -i standalone-4-0 -- neo4j-admin dump --database=system --to=/backups
     ```

     ```
     kubectl exec -t -i standalone-4-0 -- neo4j-admin dump --database=neo4j --to=/backups
     ```

     Now copy the backups from the Kubernetes Pod to an intermediate to the `/migration-backups` folder on your local filesystem:

     ```
     kubectl cp standalone-4-0:/backups /path/to/migration-backups
     ```

     The result is two files called *system.dump* and *neo4j.dump* located in the */migration-backups* folder. For more information about the `neo4j-admin dump` command, see Operations Manual →

Backup an offline database.

⚠	The file system copy-and-paste of databases is not supported and may result in unwanted behavior.

Set up 5.x standalone server

ℹ	Follow the Quickstart guide for detailed information about installing a 5.x standalone server using the *neo4j/neo4j* Helm chart.

Prepare the 5.x standalone server for the migration

1. Update the `neo4j` helm repository:

```
helm repo update neo4j
```

2. Make a note of any user-supplied values that need to be migrated from the 4.4 Helm installation. The user-supplied values can be printed out using the command:

```
helm get values standalone-4
```

3. After reading the Quickstart guide, install a Neo4j 5.x standalone server, for example:

```
helm install standalone-5 neo4j/neo4j --set volumes.data.mode=defaultStorageClass --set neo4j.name
=standalone-5 --set neo4j.password=password
```

ℹ	If you are using TLS and want to reuse the keys from the 4.4 installation, install 5.x in the same namespace as 4.4.

Restore the database backups

Migrate your databases using one of the following options depending on the Neo4j edition:

Enterprise

1. Copy the backup files from the `/migration-backups` to the 5.x standalone server:

```
kubectl cp /path/to/migration-backups/<databasename> standalone-5-0:/backups/<databasename>
```

2. If restoring the default `neo4j` database or another database that already exists, it must first be dropped:

```
kubectl exec -t -i standalone-5-0 -- cypher-shell -u neo4j -p password "DROP DATABASE neo4j
IF EXISTS"
```

3. Use the `neo4j-admin restore` command to restore each of your databases except the `system` database:

```
kubectl exec -t -i standalone-5-0 -- neo4j-admin database restore <databasename> --from-
path=/backups/<databasename>
```

4. Migrate each of your restored databases:

```
kubectl exec -t -i standalone-5-0 -- neo4j-admin database migrate <databasename>
```

5. Recreate each of your migrated databases:

```
kubectl exec -t -i standalone-5-0 -- cypher-shell -d system -u neo4j -p password "CREATE
DATABASE <databasename>"
```

6. (Optional) Restore the roles and privileges associated with each of your databases by running the respective metadata script *data/scripts/databasename/restore_metadata.cypher*, which the `neo4j-admin restore` command output, using Cypher Shell:

```
kubectl exec -t -i standalone-5-0 -- cypher-shell -u neo4j -p password -d system --param
"database => 'neo4j'" -f /data/scripts/neo4j/restore_metadata.cypher
```

7. If you have kept your 4.4 server running while setting up the new one, you can uninstall it now.

1. Place the service in offline maintenance mode, so the Neo4j process is not running:

   ```
   helm upgrade standalone-5 neo4j/neo4j --reuse-values --set
   neo4j.offlineMaintenanceModeEnabled=true
   ```

2. Copy the backup files from the `/migration-backups` to the 5.x standalone server:

   ```
   kubectl cp /path/to/migration-backups/<databasename>.dump standalone-5-
   0:/backups/<databasename>.dump
   ```

3. Use the `neo4j-admin database load` command to move the dump files to the new installation:

   ```
   kubectl exec -t -i standalone-5-0 -- neo4j-admin database load --overwrite-destination
   --from-path=/backups/<databasename>.dump "*"
   ```

4. Migrate all your databases:

   ```
   kubectl exec -t -i standalone-5-0 -- neo4j-admin database migrate "*"
   ```

5. Place the Neo4j DBMS in online mode:

   ```
   helm upgrade standalone-5 neo4j/neo4j --reuse-values --set
   neo4j.offlineMaintenanceModeEnabled=false
   ```

6. If you have kept your 4.4 server running while setting up the new one, you can uninstall it now.

Chapter 7. Neo4j 4 upgrades and migration

It is recommended to read the Introduction section before continuing.

For critical systems, it is recommended to:

- Perform the upgrade in a test environment.
- Back up your current deployment, including databases, configuration, and encryption keys to avoid losing data in case of failure. If you are upgrading a cluster, back up each of your cluster members.

To gain a better understanding of the upgrade and migration in Neo4j 4, read the following pages in order:

- Understanding upgrades and migration
- Supported paths
- Planning and preparation

Then, follow the instructions in the folder for the upgrade/migration you want to perform.

7.1. Understanding upgrades and migration in 4.x

This section describes the Neo4j DBMS components that might need to be upgraded or migrated as well as the difference between an upgrade, a migration, and a store copy.

7.1.1. DBMS components you should be aware of

It is useful to be aware of the following DBMS components beforehand:

Database

A database is a single database for storing data (e.g. Nodes and Relationships) within a Neo4j DBMS. Its physical structure is organized in files within a directory, which has the same name as the database. The default Neo4j installation contains one *default* database, called `neo4j`. A different name can be configured before starting Neo4j for the first time.

Neo4j 3.5 supports only one database. Neo4j 4.0 introduced the support for multiple databases.

Store format

A store format defines how the data of a database is stored on the file system. Each database has its own store format, which can be auto-upgraded at server startup if `dbms.allow_upgrade=true` is set.

For more information on the available store formats per Neo4j version, how to configure them, and their limits, see Operations Manual → Display store information.

The `system` database

Neo4j 4.0 and higher versions support the management of multiple databases within the same DBMS. All these databases are controlled through a shared database, called the `system` database. The role of the `system` database is to define the configuration for the other databases. There could be various types of configuration for the databases, such as operational configuration, database existence, status (online/offline), security configuration (RBAC).
The structure of the graph contained in the `system` database changes with each new `MAJOR` or `MINOR` version of Neo4j. Therefore, each time a Neo4j deployment is upgraded to a new `MAJOR` or `MINOR` version, the contents of the `system` database must be transformed as well.

7.1.2. Upgrade

Neo4j upgrade is the process of upgrading an existing single or clustered deployment to a newer `PATCH` or `MINOR` version.

Such upgrades **do not require** changes to the Neo4j configurations or the applications that use Neo4j. For example, upgrades between:

- adjacent and non-adjacent `PATCH` releases within the same `MAJOR` and `MINOR` version, e.g., `4.2.1` to `4.2.2` or `4.3.0` to `4.3.4`.
- adjacent `MINOR` versions, e.g., `4.0.0` to `4.1.3`.

What to upgrade:

- The Neo4j product.
- The store format, only if a new format has been introduced or you want to change to another format, for example, from `standard` to `aligned` or to `high_limit`. The opposite is not possible.
- The `system` database schema, which restructures the contents of that database.
  From 4.1 onwards, this is done automatically on a standalone server and in offline cluster upgrades (where you temporarily set each server as standalone). However, for rolling upgrades, you have to manually upgrade the `system` database after the other two upgrades (product and stores) finish.
- Configuration settings. In `PATCH` and `MINOR` upgrades, you do not need to change the configuration settings. If you leave the *neo4j.conf* file unchanged, everything should work as in the old version. However, some versions may include new features, so it is always good to review the changes to the Configuration settings and Procedures and update the settings if needed.

7.1.3. Migration

Neo4j migration is the process of migrating an existing single or clustered deployment to a newer `MAJOR` version. Such migrations **require** a review of the Neo4j configurations and the applications that use Neo4j. For example, migration between:

- `MAJOR` versions, e.g., from 3.5.13 to 4.0.10.

What to migrate:

- The Neo4j product.

- The store format of the database (which is auto-upgraded at server startup).
  For more information on the available store formats per Neo4j version, how to configure them, and their limits, see Operations Manual → Display store information.
- The `system` database schema, which restructures the contents of that database (automatically created on the first startup of Neo4j 4.x).
- Configuration settings — the configuration changes from the newer version must be applied to the old configuration file.
- Application code — the source code of your application(s) must be updated as per the new version in order to work properly.
- Plugins (including custom plugins) - all plugins must be compatible with the new version of Neo4j.

7.1.4. Store copy Enterprise edition

Neo4j store copy is the process of migrating the *store format* of a database from Neo4j Community or Enterprise Edition to Neo4j Enterprise Edition. It uses the `neo4j-admin copy` command, which does not copy the *schema* store, and therefore, you can migrate a database directly into a Neo4j DBMS on the latest version. If a schema is defined, you have to recreate it by running the commands that the `neo4j-admin copy` operation outputs. Note that to be able to copy a database, it has to be **offline**.

7.1.5. Limitations

- **Neo4j does not support downgrades.** If the upgrade or migration is not successful, you have to do a full rollback, including restoring a pre-upgrade or a pre-migration backup.
- A Neo4j upgrade and migration must be performed as an isolated operation.
- Migration requires Neo4j DBMS downtime.

7.2. Supported upgrade and migration paths in 4.x

This section explains the supported upgrade and migration paths.

Which path you choose depends on your current version and the version you want to go to; if you are going to upgrade or migrate the whole Neo4j DBMS or a single database; and your backup and restore strategy.

- Sequential path (Neo4j DBMS) from 3.5.latest → 4.0.latest → 4.1.latest → 4.2.latest → 4.3.latest → 4.4.latest
  It includes migration and follow-up upgrades of the whole Neo4j DBMS (both the databases and `system` database).
- Copy/Migrate a single database from Neo4j Community or Enterprise Edition 3.5.latest or 4.x to Neo4j Enterprise Edition 4.4.latest
  It makes use of the `neo4j-admin copy` command to migrate the *data store* of the single database.

The `neo4j-admin copy` command is a Neo4j Enterprise Edition feature. You can use it to migrate from Neo4j Community Edition to Neo4j Enterprise Edition.

- Restore a single database into a running Neo4j DBMS on a later version.

7.2.1. Sequential path (Neo4j DBMS)

Following the sequential path, you first migrate your Neo4j deployment from 3.5.latest to 4.0.latest and then sequentially upgrade it to 4.1.latest, 4.2.latest, 4.3.latest, and 4.4.latest. It makes use of `neo4-admin load` or `neo4j-admin restore` to copy both the databases and the `system` database or backups to the new installation, and then it upgrades the *data* and *schema* stores to the new version during the Neo4j startup. The `neo4j-admin` commands can be run from an online, as well as from an offline Neo4j DBMS, and must be invoked as the `neo4j` user to ensure the appropriate file permissions.

Documentation for completing your upgrade/migration

Starting version	Target version	Supported path	Operations	Documentation
3.4.latest	3.5.latest	3.4.latest → 3.5.latest	Upgrade	Operations Manual 3.5 → Upgrade
3.5.latest	4.0.latest	3.5.latest → 4.0.latest	Migration	Neo4j 3.5 → Neo4j 4.0
3.5.latest	4.1.latest	3.5.latest → 4.0.latest → 4.1.latest	Migration and upgrade	Neo4j 3.5 → Neo4j 4.0 Neo4j 4.0.latest → Neo4j 4.1.latest
3.5.latest	4.2.latest	3.5.latest → 4.0.latest → 4.1.latest → 4.2.latest	Migration and upgrades	Neo4j 3.5 → Neo4j 4.0 Neo4j 4.0.latest → Neo4j 4.1.latest Neo4j 4.1.latest → Neo4j 4.2.latest
3.5.latest	4.3.latest	3.5.latest → 4.0.latest → 4.1.latest → 4.2.latest → 4.3.latest	Migration and upgrades	Neo4j 3.5 → Neo4j 4.0 Neo4j 4.0.latest → Neo4j 4.1.latest Neo4j 4.1.latest → Neo4j 4.2.latest Neo4j 4.2.latest → Neo4j 4.3.latest
3.5.latest	4.4.latest	3.5.latest → 4.0.latest → 4.1.latest → 4.2.latest → 4.3.latest → 4.4.latest	Migration and upgrades	Neo4j 3.5 → Neo4j 4.0 Neo4j 4.0.latest → Neo4j 4.1.latest Neo4j 4.1.latest → Neo4j 4.2.latest Neo4j 4.2.latest → Neo4j 4.3.latest Neo4j 4.3.latest → Neo4j 4.4.latest

Starting version	Target version	Supported path	Operations	Documentation
4.0.latest	4.1.latest	4.0.latest → 4.1.latest	Upgrade	Neo4j 4.0.latest → Neo4j 4.1.latest
4.0.latest	4.2.latest	4.0.latest → 4.1.latest → 4.2.latest	Upgrades	Neo4j 4.0.latest → Neo4j 4.1.latest Neo4j 4.1.latest → Neo4j 4.2.latest
4.0.latest	4.3.latest	4.0.latest → 4.1.latest → 4.2.latest → 4.3.latest	Upgrades	Neo4j 4.0.latest → Neo4j 4.1.latest Neo4j 4.1.latest → Neo4j 4.2.latest Neo4j 4.2.latest → Neo4j 4.3.latest
4.0.latest	4.4.latest	4.0.latest → 4.1.latest → 4.2.latest → 4.3.latest → 4.4.latest	Upgrades	Neo4j 4.0.latest → Neo4j 4.1.latest Neo4j 4.1.latest → Neo4j 4.2.latest Neo4j 4.2.latest → Neo4j 4.3.latest Neo4j 4.3.latest → Neo4j 4.4.latest
4.1.latest	4.2.latest	4.1.latest → 4.2.latest	Upgrade	Neo4j 4.1.latest → Neo4j 4.2.latest
4.1.latest	4.3.latest	4.1.latest → 4.2.latest → 4.3.latest	Upgrade	Neo4j 4.1.latest → Neo4j 4.2.latest Neo4j 4.2.latest → Neo4j 4.3.latest
4.1.latest	4.4.latest	4.1.latest → 4.2.latest → 4.3.latest → 4.4.latest	Upgrade	Neo4j 4.1.latest → Neo4j 4.2.latest Neo4j 4.2.latest → Neo4j 4.3.latest Neo4j 4.3.latest → Neo4j 4.4.latest
4.2.latest	4.3.latest	4.2.latest → 4.3.latest	Upgrade	Neo4j 4.2.latest → Neo4j 4.3.latest
4.2.latest	4.4.latest	4.2.latest → 4.3.latest → 4.4.latest	Upgrade	Neo4j 4.2.latest → Neo4j 4.3.latest Neo4j 4.3.latest → Neo4j 4.4.latest
4.3.latest	4.4.latest	4.3.latest → 4.4.latest	Upgrade	Neo4j 4.3.latest → Neo4j 4.4.latest

7.2.2. Copy/Migrate a single database Enterprise edition

You can use the `neo4j-admin copy` command to copy or migrate a single database, **except for your `system` database**, into a running Neo4j DBMS on any later version. The copy command copies and, if needed, migrates the *data store* of a database. However, it does not copy the *schema store*. Therefore, if a schema is defined, you must manually run the commands that the `neo4j-admin copy` operation outputs to recreate the schema.

This path is suitable for:

- Cleaning up database inconsistencies and compacting stores.
- Migrating a single database from a 3.5 Neo4j DBMS (Community or Enterprise) to 4.x Enterprise.
- Migrating the store format of a single database. For more information on the store formats versions and compatibility, see Operations Manual → Store formats and entity limits.
- Migrating a single database from a 4.x Community Edition to 4.y Enterprise Edition.
- Migrating a single database from a 4.x Enterprise Edition to a later 4.y Enterprise Edition.
- Migrating a single database from a 4.x Enterprise Edition to a later 4.y Enterprise Edition with a different store format.

> The `neo4j-admin copy` command copies the entire data store and is an IOPS-intensive process. It takes up more room and is slower than a migration on startup. However, if you want to move over several of the sequential steps at once or for some reason want to move a few databases at a time, this might be a good option.
>
> Keep in mind that the `neo4j-admin copy` command copies the node IDs, but the relationships get new IDs. Therefore, if you want to preserve the relationship IDs, you should follow the sequential path.

For more information, see Tutorial: Back up and copy a single database in a running single instance and Tutorial: Back up and copy a single database in a running cluster.

7.2.3. Restore a single database Enterprise edition

You can use the `neo4j-admin restore` command to restore a backup of a single database into a running Neo4j DBMS on a later version. However, it **does not migrate the store_format** of the database. Therefore, if the restored database is not on the same store format as the DBMS, you will have to set `dbms.allow_upgrade=true` to enable the migration. For versions earlier than 4.1, you have to restart the DBMS for the configuration to take effect.

For more information, see Tutorial: Back up and restore a single database in a single instance and Tutorial: Back up and restore a single database in a cluster.

7.3. Considerations and planning

This chapter describes what to consider when planning your Neo4j DBMS upgrade or migration.

Planning an upgrade or migration is crucial to ensure that everything goes smoothly, and you are covered in case of failure. This section aims to highlight some important items to keep an eye on.

7.3.1. General considerations

Duration

It is impossible to understand beforehand how long an upgrade or a migration will take, due to the number of moving parts involved, such as (but not restricted to):

- The size of your data store.
- Do you need to migrate the store format?
- Do indexes need to be migrated?
- How big are your indexes?
- How fast is your hardware?

Because of this, it is recommended to prepare an environment where you can test the upgrade or migration process back-to-back. This will give you more accurate timings than you can ever estimate as well as an idea of what to expect in terms of duration for the process. As a rule of thumb, the *data* migration part of an upgrade or a migration process (not the entire process) takes around 50\~55 minutes for a 600GB store.

Configuration changes

In `PATCH` and `MINOR` upgrades, you do not need to change the configuration settings. If you leave the *neo4j.conf* file unchanged, everything should work as in the old version. However, some versions may include new features, so it is always good to review the changes to the Configuration settings and Procedures and update the settings if needed.

New `MAJOR` versions introduce changes to configuration settings, such as new configuration names, deprecated and removed features. It is advised to check the changes to the configuration settings to understand all changes required.

At this stage, a good practice is to go through your current *neo4j.conf* and capture any non-default configurations you may have. You need to add these configurations to the *neo4j.conf* file of your Neo4j 4.x.

You can use the following to strip all comments/empty lines of neo4j.conf file:

```
grep -v "^#" neo4j.conf | sed -e '/^$/d' | sort
```

Logs

All log formats/layouts have changes in Neo4j 4.x. Some are minor, others more significant, so take your time to familiarize yourself with the new formats and information they contain. If you are consuming these logs and generating alarms based on them, it is worth checking the impact of the log format changes.

For more information about the available logs and how to configure them, including log rotation and retention, see the Operations Manual → Logging of the version you are upgrading to.

Metrics

There are three important things to note on the metrics front:

- The metrics that are enabled by default have been changed in the 4.2 version. Any specific metrics that you want to be enabled must be specified in the configuration.
- From 4.0 onwards, the rules for naming metrics have changed to include the database name prefix.
- Neo4j 4.0 version also introduces namespaces for metrics, which are disabled by default.

If you are migrating from 3.5 to 4.x and you have set up a monitoring dashboard, either built in-house or from a 3rd party vendor, you should understand how these changes will impact the dashboards after the migration.

The Operations Manual contains two useful sections on this topic:

- Expose Metrics
- Full metrics reference (includes all metric names)

Embedded deployments

For users with embedded deployments (when you include Neo4j in your project), from version 4.2 onwards, Neo4j supports running a Causal Cluster in embedded mode. However, coming from Neo4j HA embedded, there are a small number of changes required. For more information, see Java Reference → Using Neo4j embedded in Java applications and Operations Manual → Embedded usage.

7.3.2. Migration considerations

Downtime

When migrating to a newer Neo4j `MAJOR` version, only offline migrations are supported. Therefore, it requires planning for some downtime. Plan for this accordingly. Because each case is unique, it is recommended to run one or several test migrations to understand how long the process will take back-to-back. That information will allow you to better plan for the downtime window you will need.

Disk space considerations

A migration requires substantial free disk space, as it makes an entire copy of the database and creates temporary files. It is also recommended taking a backup of your production store in case something goes wrong. Because of the reasons listed above, it is good to reserve two times the size of the database directory (three times in case you take a backup beforehand) for migration purposes.

Java version

Neo4j 4.x runs on Java 11. The following table shows the compatible Java Virtual Machine (JVM) to run a Neo4j instance.

Neo4j version and JVM requirements

Neo4j version	Java version
4.x	Java SE 11
3.x	Java SE 8

ℹ If you have other Java applications running on the machine hosting Neo4j, make sure those applications are compatible with the Java version your Neo4j is running on. Alternatively, you should configure to run multiple JDKs on the same machine.

Database naming rules

With the introduction of multiple databases, the rules for naming a database have changed. For example, it is no longer possible to use an underscore in a database name. For a full list of naming rules, see Operations Manual → Administrative commands.

Application code

Depending on how your application is interacting with Neo4j, you should be prepared to review your application code. Neo4j 4.x has changes that may impact your application such as (but not restricted to):

- Removal of REST API
- Changes to HTTP API endpoints
- Procedure refactoring
- Changes in cluster REST endpoints
- Core Java API

All breaking changes that may affect your application and Neo4j can be found in Breaking changes between Neo4j 3.5 and Neo4j 4.x.

Drivers

Neo4j's official drivers have some significant and breaking changes you need to be aware of. The Breaking changes between Neo4j 1.7 drivers and Neo4j 4.x drivers section contains all information and lists all the breaking changes.

The following are some key changes that may cause confusion:

- Starting with Neo4j 4.0, the versioning scheme for the database, driver, and protocol are all aligned. For supported drivers, this means that the version number goes from 1.7 to 4.0. This is merely a cosmetic change and version 4.0 of the drivers is in fact only one release ahead of 1.7.

The driver version was version bumped from 1.7 to 4.0 for each official Neo4j driver. There are no driver versions 1.8, 2.x, and 3.x.

- The driver's default configuration for encrypted is now `false`. A 4.x driver only attempts plain text connections by default.
- When encryption is explicitly enabled, connections with holding self-signed certificates will fail on certificate verification by default. On Neo4j 4.x the default trust mode is to trust the CAs that are trusted by the operating system.
- `v1` is removed from drivers' package name.
- The `neo4j://` scheme replaces `bolt+routing://` and can be used for both clustered and single-instance configurations.
- With 4.0 servers, session instances should now be acquired against a specific database.
- Bookmark has changed from a string, and a list of strings, to a Bookmark object.
- Several language specific driver changes.

The list above does not reflect the entirety of changes in the drivers, so it is imperative to read through the Breaking changes between Neo4j 1.7 drivers and Neo4j 4.x drivers section. Failing do to so may result in unwanted problems when your application tries to connect to Neo4j after the migration. As a tip, you can also check the Neo4j Drivers documentation (for all officially supported languages), which will guide you through all the drivers features configurations, such as deeper dive on Authentication, Asynchronous sessions, and Reactive sessions (client-side back-pressure).

Plugins (including custom plugins)

Take note of the plugins you are using and make sure they are compatible with Neo4j 4.x.

The following are the most commonly used plugins/plugin types:

- One of the most used plugins is APOC, a procedure library that extends the Neo4j functionality. If you are using APOC, check the Version Compatibility Matrix and make sure you plan to upgrade APOC alongside your Neo4j migration.

  To quickly check your APOC version, you can run `RETURN apoc.version();`

- If you are using Neo4j Bloom or Graph Data Science Library (GDSL), you can find the most recent versions for these products in the Neo4j Download Center. Be aware that some previous versions of Bloom and GDSL were not initially compatible with Neo4j 4.0. This has been fully rectified, and all of the product suite is compatible with the 4.x series. Therefore, it is recommended to simply upgrade to the latest versions of these products.
- If you have developed any custom plugins, you should review them as you would your application code. The several changes in Neo4j 4.x can impact the behaviour of these custom plugins, therefore, it is highly-advised to plan some time for this as well.

Other 3rd-party software and tools

Be mindful of any other 3rd-party software and tools you are using alongside Neo4j. Maybe you have leveraged operational scripts to install, manage, backup, or monitor your Neo4j deployment. You might also have set alarms and built complete monitoring dashboards. You need to revise these as there have been changes in metrics (as mentioned in section Metrics) and the Neo4j operational tools now account for multiple databases. Therefore, it is recommended to review all scripts/tools/3rd party software and make sure they are prepared and compatible with Neo4j 4.x.

7.4. Upgrade to a later 4.x `PATCH` release

This chapter describes how to upgrade the Neo4j DBMS to a newer `PATCH` release within the same `MAJOR` and `MINOR` version.

It covers the following topics:

- Upgrade checklist
- Upgrade a single instance
- Upgrade a Causal Cluster

7.4.1. Upgrade checklist for 4.x patch release

This chapter describes how to prepare to upgrade the Neo4j DBMS to a newer `PATCH` release within the same `MAJOR` and `MINOR` version.

Before you start preparing for the upgrade, it is very important to read the information in Understanding upgrades and migration and Supported upgrade and migration paths.

Follow the checklist to prepare for upgrading your Neo4j deployment:

- ☐ Complete all prerequisites for the upgrade.
- ☐ Back up your current deployment to avoid losing data in case of failure.

  Neo4j does not support downgrades. If the upgrade is not successful, you have to do a full rollback, including restoring a pre-upgrade backup.

- ☐ Download the new version of Neo4j.
- ☐ Prepare a new *neo4j.conf* file to be used by the new installation.
- ☐ Perform a test upgrade as per your Neo4j version and deployment type (Single instance or Causal Cluster).
- ☐ Monitor the logs.
- ☐ Perform the upgrade.

If you are upgrading a Causal Cluster, complete the checklist for each cluster member.

Prerequisites

1. Verify that you have installed Java 11.
2. Review the improvements and fixes that have been carried out in the version that you want to upgrade to.
   - Neo4j changelog
   - Release Notes
   - Changes to configuration settings in Neo4j 4
   - Changes to procedures in Neo4j 4

Back up your current deployment

Depending on your particular deployment and environment, it is important to design an appropriate backup and restore strategy. For more information about what to consider when deciding on your backup and restore strategy, what needs to be backed up, and the different backup modes and options, see the Operations Manual → Backup and restore planning page of your version.

Prepare a new *neo4j.conf* file to be used by the new installation

Prepare a *neo4j.conf* file to be used by the new installation if you have any custom values configured in your old *neo4j.conf* file. Look for:

- Any non-default settings.
- Any custom values of the settings `dbms.directories.*` and `dbms.default_database`.
- (In cluster installations) Cluster-specific configuration settings, which might be different for the different cluster members.

Perform a test upgrade

Based on the findings in this chapter, allocate a staging test environment for the upgrade and do a test upgrade. The test upgrade will give you valuable information about the time required for the production upgrade. Follow the steps as per your Neo4j version and deployment type.

Monitor the logs

The *neo4j.log* file contains information on how many steps the upgrade will involve and how far it has progressed. It is a good idea to monitor this log. For more information, see Operations Manual → Log files.

7.4.2. Upgrade a single instance to a 4.x patch release

This section describes how to upgrade a single Neo4j instance to a newer `PATCH` release within the same `MAJOR` and `MINOR` version.

Prerequisites

Ensure that you have completed all the tasks on the Upgrade checklist.

Prepare for the upgrade

1. Install the *Neo4j* version that you want to upgrade to. For more information on how to install the distribution that you are using, see the Operations Manual → Installation section of the version that you want to install.
2. Replace the *neo4j.conf* file with the one that you have prepared in section Prepare a new *neo4j.conf* file to be used by the new installation.
3. Copy all the files used for encryption, such as private key, public certificate, and the contents of the trusted and revoked directories (located in *<neo4j-home>/certificates/*).
4. Restore each of the databases and transactions that you have backed up in the new installation, including the `system` database, by either using `neo4j-admin restore` (online) or `neo4j-admin load` (offline), depending on your backup approach. If you are running a Debian/RPM distribution, you can skip this step.

   If your old installation has modified configurations starting with `dbms.directories.*` or the setting `dbms.default_database`, verify that the new `neo4j.conf` file is configured properly to find these directories.

Upgrade your instance

1. Start the instance by running the following command from `<neo4j-home>`:

   ```
   bin/neo4j start
   ```

   The upgrade takes place during startup.
2. Monitor the neo4j.log file for information on how many steps the upgrade involves and how far it has progressed.

Post-upgrade tasks

It is good practice to make a full backup immediately after the upgrade.

7.4.3. Upgrade a Causal Cluster to a 4.x patch release

This section describes how to upgrade a Neo4j cluster to a newer `PATCH` release within the same `MAJOR` and `MINOR` version.

You can upgrade your existing Neo4j Causal Cluster by either performing a rolling upgrade, or by upgrading it offline.

The prerequisites and the upgrade steps must be completed for each cluster member.

When upgrading from an earlier patch release, you might need to upgrade the `system` database as well. Use `call dbms.upgradeStatus` to check whether this is necessary or see Upgrade the system database for more information.

Offline upgrade

This variant is suitable for cases where a rolling upgrade is not possible.

It is recommended to perform a test upgrade on a production-like environment to get information on the duration of the downtime.

Prerequisites

Ensure that you have completed all the tasks on the Upgrade checklist for each cluster member.

Prepare for the upgrade

1. Shut down all the cluster members (Cores and Read Replicas).
2. Perform `neo4j-admin unbind` on each cluster member to remove cluster state data.
3. Install the Neo4j version that you want to upgrade to on each instance. For more information on how to install the distribution that you are using, see the Operations Manual → Installation section of the version that you want to install.
4. Replace the *neo4j.conf* file with the one that you have prepared for each instance in section Prepare a new *neo4j.conf* file to be used by the new installation.
5. Copy all the files used for encryption, such as private key, public certificate, and the contents of the trusted and revoked directories (located in *<neo4j-home>/certificates/*).
6. Restore each of your backed-up databases and transactions on each cluster member, including the `system` database, by either using `neo4j-admin restore` (online) or `neo4j-admin load` (offline), depending on your backup approach. If you are running a Debian/RPM distribution, you can skip this step.

   If your old installation has modified configurations starting with `dbms.directories.*` or the setting `dbms.default_database`, verify that the new `neo4j.conf` file is configured properly to find these directories.

Upgrade your cluster

1. Start Neo4j by running the following command from `<neo4j-home>`:

```
bin/neo4j start
```

The upgrade takes place during startup.

2. Monitor the neo4j.log file for information on how many steps the upgrade will involve and how far it has progressed.
3. Verify that the cluster forms and the upgraded Neo4j DBMS deployment comes online.
4. Start each Read Replica and wait for it to catch up with the rest of the cluster members.
5. Verify that the Read Replicas join the cluster.

Post-upgrade

It is recommended to perform a *full backup*, using an empty target directory.

Rolling upgrade

Rolling upgrade is a zero-downtime method for upgrading a Causal Cluster. You upgrade one member at a time, while the rest of the members are running. However, if during a rolling upgrade the cluster loses quorum and cannot be recovered, then downtime may be required to do a disaster recovery.

Recommendations

- The critical point during the upgrade is knowing when it is safe to switch off the original member. It is highly recommended to monitor the status endpoint before each removal, in order to decide which member to switch off and when it is safe to do so.
- To reduce the risk of failure during a rolling upgrade, make sure the cluster is not under any heavy load during the upgrade. If possible, the safest would be to disable writes entirely.
- There should be no changes to database administration during a rolling upgrade. For more information, see Operations Manual → Manage databases.

Rolling upgrade for a fixed number of servers

This variant is suitable for deployments where there is a fixed number of servers and they have to be updated in-place.

When performing a rolling upgrade for a fixed number of servers, it is not possible to increase the cluster size. Therefore, the cluster fault tolerance level will be reduced while replacing the members.

Prerequisites

1. Ensure that you have completed all the tasks on the Upgrade checklist for each cluster member.
2. Verify that **all databases are online** by running `SHOW DATABASES` in Cypher Shell or Neo4j Browser.

Offline databases can be started using `START DATABASE [database-name]`.

All databases must be started before you start a rolling upgrade. If you have to keep a database inaccessible during the rolling upgrade, you can disable access to it by using one of following ways:

> You must never run `DENY ACCESS ON DATABASE system TO PUBLIC` or `DENY ACCESS ON DATABASE * TO PUBLIC` because you will lock yourself out of the `system` database. If you do lock yourself out, follow the disable authentication steps in the Operations Manual to recover and prevent outside access to the instance or cluster.

> *For `PATCH` upgrades within 4.0 version*
>
> 1. Deny all roles access to each database except the `system` database. You can query all available roles with `SHOW ROLES`.
>
> ```
> DENY ACCESS ON DATABASE [database-name] TO [role1],[role2]
> ```
>
> 2. Ensure that the databases cannot be stopped during the rolling upgrade by using the following command:
>
> ```
> DENY STOP ON DATABASE * TO admin
> ```
>
> This must be done for the `admin` role and all other roles that have the privilege to stop databases. For more information about listing privileges, see Cypher Manual → Managing privileges.

> *For `PATCH` upgrades within a 4.x version::*
>
> 1. Deny the `PUBLIC` role access to each database except the `system` database.
>
> ```
> DENY ACCESS ON DATABASE [database-name] TO PUBLIC
> ```
>
> 2. Ensure that the databases cannot be stopped, created, or dropped during the rolling upgrade by using the following command:
>
> ```
> DENY STOP ON DATABASE * TO PUBLIC
> DENY DATABASE MANAGEMENT ON DBMS TO PUBLIC
> ```

Upgrade the cluster

You upgrade **one cluster member at a time**, while the rest of the members are running.

> If during a rolling upgrade the cluster loses quorum and cannot be recovered, then downtime may be required to do disaster recovery.

For each cluster member

1. (Recommended) Use the process described in the status endpoint to evaluate whether it is safe to remove the old instance.
2. Shut down the instance.
3. Install the *Neo4j* version that you want to upgrade to. For more information on how to install the distribution that you are using, see the Operations Manual → Installation section of the version that you want to install.
4. Replace the *neo4j.conf* file with the one that you have prepared for this instance in section Prepare a new *neo4j.conf* file to be used by the new installation.
5. Start the new instance and wait for it to catch up with the rest of the cluster members.
6. Verify that the new instance has successfully joined the cluster and caught up with the rest of the members, by using the status endpoint.

Because Read Replicas are not part of the cluster consensus group, their replacement during an upgrade does not affect the cluster availability and fault tolerance level. However, it is still recommended to incrementally add Read Replicas for a structured and maintainable upgrade process.

Post-upgrade steps

The following steps must be performed after a rolling upgrade.

For `PATCH` upgrades within 4.0 version

1. Restore the privilege of the `admin` role to stop databases.

   ```
   REVOKE DENY STOP ON DATABASE * FROM admin
   ```

   This must be done for all roles for which the privilege to stop databases has been denied (see step 6 of Rolling upgrade for a fixed number of servers). For more information about listing privileges, see Cypher Manual 4.0 → Graph and sub-graph access control.

2. (Optional) If you have started offline databases and denied some access rights during the preparation phase for a rolling upgrade, you should also restore them to the original state:

   a. Stop each of the databases by running the following command:

      ```
      STOP DATABASE [database-name]
      ```

   b. Re-enable access to the databases by running the following command:

      ```
      REVOKE DENY ACCESS ON DATABASE [database-name] FROM [role1],[role2]
      ```

For `PATCH` upgrades within a 4.x version

1. Restore the privilege of the `PUBLIC` role to stop databases:

```
REVOKE DENY STOP ON DATABASE * FROM PUBLIC
```

2. Restore the privilege of the `PUBLIC` role to create and drop databases:

```
REVOKE DENY DATABASE MANAGEMENT ON DBMS FROM PUBLIC
```

3. (Optional) If you have started offline databases during the preparation phase for a rolling upgrade, you stop each of them to restore them to the original state:

```
STOP DATABASE [database-name]
```

Rolling upgrade for cloud infrastructure

This variant is suitable for deployments that use replaceable cloud or container resources. It follows the same steps as for the fixed number of servers, but **you can add the new members before you shut down the old ones**, thus preserving the cluster fault tolerance level. Because Read Replicas are not part of the cluster consensus group, their replacement during the upgrade will not affect the cluster availability and fault tolerance level. However, it is still recommended to incrementally add Read Replicas for a structured and maintainable upgrade process.

7.5. Upgrade 4.3 to 4.4 `MINOR` version

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

It covers the following topics:

- Upgrading Neo4j 4.3 to 4.4 `MINOR` version.
  - Upgrade checklist
  - Upgrade a single instance
  - Upgrade a Causal Cluster
  - Troubleshooting

7.5.1. Upgrade checklist

This chapter describes how to prepare to upgrade Neo4j from an earlier minor or patch version.

Before you start preparing for the upgrade, it is very important to read the information in Understanding upgrades and migration and Supported upgrade and migration paths.

Follow the checklist to prepare for upgrading your Neo4j deployment:

- ☐ Complete all prerequisites for the upgrade.
- ☐ Back up your current deployment to avoid losing data in case of failure.
- ☐ Download the new version of Neo4j. Make sure the upgrade path is supported.
- ☐ Prepare a new *neo4j.conf* file to be used by the new installation.
- ☐ Perform a test upgrade as per your Neo4j version and deployment type (Single instance or Causal Cluster).
- ☐ Monitor the logs.
- ☐ Perform the upgrade.

If you are upgrading a Causal Cluster, complete the checklist for each cluster member.

Prerequisites

1. Verify that you have installed Java 11.
2. Review the improvements and fixes that have been carried out in the version that you want to upgrade to.
   - Neo4j 4.4 Change log
   - Release Notes
   - Changes to configuration settings in Neo4j 4
   - Changes to procedures in Neo4j 4

Back up your current deployment

Depending on your particular deployment and environment, it is important to design an appropriate backup and restore strategy. For more information about what to consider when deciding on your backup and restore strategy, what needs to be backed up, and the different backup modes and options, see Operations Manual → Backup and restore planning.

Prepare a new *neo4j.conf* file to be used by the new installation

Prepare a *neo4j.conf* file to be used by the new installation if you have any custom values configured in your old *neo4j.conf* file. Look for:

- Any non-default settings.
- Any custom values of the settings `dbms.directories.*` and `dbms.default_database`.
- (In cluster installations) Cluster-specific configuration settings, which might be different for the different cluster members.

Perform a test upgrade

Based on the findings in this chapter, allocate a staging test environment for the upgrade and do a test upgrade. The test upgrade will give you valuable information about the time required for the production upgrade. Follow the steps as per your Neo4j version and deployment type.

Monitor the logs

The neo4j.log file contains information on how many steps the upgrade will involve and how far it has progressed. It is a good idea to monitor this log.

7.5.2. Upgrade a single instance to 4.4

This section describes how to upgrade a single Neo4j instance.

Prerequisites

Ensure that you have completed all tasks on the Upgrade checklist.

Prepare for the upgrade

1. Install the Neo4j version that you want to upgrade to. For more information on how to install the distribution that you are using, see Operations Manual → Installation.
2. Replace the *neo4j.conf* file with the one that you have prepared in section Prepare a new neo4j.conf file to be used by the new installation.
3. Set `dbms.allow_upgrade=true` to allow automatic store upgrade.
4. Set `dbms.mode=SINGLE`. This enables the automatic upgrade of the `system` database schema, as the setting `dbms.allow_single_automatic_upgrade` is `true` by default when `dbms.mode=SINGLE`.
5. Copy all the files used for encryption, such as private key, public certificate, and the contents of the trusted and revoked directories (located in *<neo4j-home>/certificates/*).
6. Restore each of your databases and transactions in the new installation, including the `system` database, by either using `neo4j-admin restore` (online) or `neo4j-admin load` (offline), depending on your backup approach. If you are running a Debian/RPM distribution, you can skip this step.

   If your old installation has modified configurations starting with `dbms.directories.*` or the setting `dbms.default_database`, verify that the new `neo4j.conf` file is configured properly to find these directories.

7. If you are using custom plugins, make sure they are updated and compatible with the new version, and place them in the */plugins* directory.

Upgrade your instance

1. Start the instance by running the following command from `<neo4j-home>`:

   ```
   bin/neo4j start
   ```

   The upgrade takes place during startup.

2. Monitor the neo4j.log file for information on how many steps the upgrade involves and how far it has progressed.

Post-upgrade tasks

1. When the upgrade finishes, open the *neo4j.conf* file and set `dbms.allow_upgrade=false`. If, for some reason, you have forgotten to enable the automatic upgrade of the `system` database schema, use the following commands to manually upgrade it:
   - `CALL dbms.upgradeStatus()` to determine whether an upgrade is necessary or not.
   - `CALL dbms.upgrade()` on the `system` database to perform the upgrade of the system schema. For more details, see Operations Manual 4.4 → Procedures.
2. Restart the instance by running the following command from `<neo4j-home>`:

   ```
   bin/neo4j restart
   ```
3. It is good practice to make a full backup immediately after the upgrade.
4. Check the default settings on all metrics. Any specific metrics that you want to be enabled **must** be specified in the `metrics.filter`. For more information, see Operations Manual → Enable metrics logging.

7.5.3. Upgrade a Causal Cluster to 4.4

This section describes how to upgrade a Neo4j Causal Cluster.

You can upgrade your existing Neo4j Causal Cluster by either performing a rolling upgrade, or by upgrading it offline.

The prerequisites and the upgrade steps must be completed for each cluster member.

Offline upgrade

This variant is suitable for cases where a rolling upgrade is not possible.

It is recommended to perform a test upgrade on a production-like environment to get information on the duration of the downtime.

Prerequisites

1. Ensure that you have completed all tasks on the Upgrade checklist.

When performing a rolling upgrade onto Neo4j 4.4, ensure that the version upgrading from is the latest release of 4.3, or the upgrade may fail.

Prepare for the upgrade

1. Shut down all the cluster members (Cores and Read Replicas).
2. Perform `neo4j-admin unbind` on each cluster member to remove cluster state data.
3. Install the Neo4j version that you want to upgrade to on each instance. For more information on how to install the distribution that you are using, see Operations Manual 4.4 → Installation.
4. Replace the *neo4j.conf* file with the one that you have prepared for each instance in section Prepare a new neo4j.conf file to be used by the new installation.
5. Copy all the files used for encryption, such as private key, public certificate, and the contents of the trusted and revoked directories (located in *<neo4j-home>/certificates/*).
6. Restore each of your databases and transactions in the new installation, including the `system` database, by either using `neo4j-admin restore` (online) or `neo4j-admin load` (offline), depending on your backup approach. If you are running a Debian/RPM distribution, you can skip this step.

   If your old installation has modified configurations starting with `dbms.directories.*` or the setting `dbms.default_database`, verify that the new `neo4j.conf` file is configured properly to find these directories.

7. If you are using custom plugins, make sure they are updated and compatible with the new version, and place them in the */plugins* directory.

Upgrade your cluster

*On **one** of the Cores*

1. Open the *neo4j.conf* file of the new installation and configure the following settings:
   - Uncomment `dbms.allow_upgrade=true` to allow automatic store upgrade.
   - Set `dbms.mode=SINGLE`. This enables the automatic upgrade of the `system` database schema, as the setting `dbms.allow_single_automatic_upgrade` is `true` by default when `dbms.mode=SINGLE`.
2. Start Neo4j by running the following command from `<neo4j-home>`:

   ```
   bin/neo4j start
   ```

   The upgrade takes place during startup.
3. Monitor the neo4j.log file for information on how many steps the upgrade will involve and how far it has progressed.
4. When the upgrade finishes, stop the server.
5. Open the *neo4j.conf* file and configure the following settings:
   - Set `dbms.allow_upgrade=false`.
   - Set `dbms.mode=CORE` to re-enable Causal Clustering in the configuration.
6. Use `neo4j-admin dump` to make a dump of each of your databases and transactions, including the `system` database.

7. Do **not** yet start the server.

On each of the other Cores

1. Copy the database dumps you created on the first Core server to each of the other Cores.
2. Use `neo4j-admin load --from=<archive-path> --database=<database> --force` to replace each of your databases, including the `system` database, with the ones you upgraded on the first Core server.
3. Start each of the core servers, including the first one, and verify that they join in a cluster.

For each Read Replica

Start the Read Replica and wait for it to catch up with the rest of the cluster members.

(Optional) Whilst an empty read replica will eventually get a full copy of all data from the other members of your cluster, this catching up may take some time. To speed up the process, you can load the data first by using `neo4j-admin load --from=<archive -path> --database=<database> --force` to replace each of your databases with the upgraded ones, including the `system` database.

Verify that the Read Replicas join the cluster.

Post-upgrade

It is recommended to perform a *full backup*, using an empty target directory.

Rolling upgrade

Rolling upgrade is a zero-downtime method for upgrading a Causal Cluster. You upgrade one member at a time, while the rest of the members are running. However, if during a rolling upgrade the cluster loses quorum and cannot be recovered, then downtime may be required to do a disaster recovery.

Recommendations

- The critical point during the upgrade is knowing when it is safe to switch off the original member. It is highly recommended to monitor the status endpoint before each removal, in order to decide which member to switch off and when it is safe to do so.
- To reduce the risk of failure during a rolling upgrade, make sure the cluster is not under any heavy load during the upgrade. If possible, the safest would be to disable writes entirely.
- There should be no changes to database administration during a rolling upgrade. For more information, see Operations Manual 4.4 → Manage databases.
- Remove `dbms.record_format` from *neo4j.conf* to avoid any accidental cross-format migration.

Rolling upgrade for a fixed number of servers

This variant is suitable for deployments where there is a fixed number of servers and they have to be updated in-place.

When performing a rolling upgrade for a fixed number of servers, it is not possible to increase the cluster size. Therefore, the cluster fault tolerance level will be reduced while replacing the members.

Prerequisites

1. Ensure that you have completed all tasks on the Upgrade checklist.

   When performing a rolling upgrade onto Neo4j 4.4, ensure that the version upgrading from is the latest release of 4.3, or the upgrade may fail.

2. Verify that **all databases are online** by running `SHOW DATABASES` in Cypher Shell or Neo4j Browser. Offline databases can be started using `START DATABASE [database-name]`.

   All databases must be started before you start a rolling upgrade. If you have to keep a database inaccessible during the rolling upgrade, you can disable access to it by using the following command:

   ```
   DENY ACCESS ON DATABASE [database-name] TO PUBLIC
   ```

   You must never run `DENY ACCESS ON DATABASE system TO PUBLIC` or `DENY ACCESS ON DATABASE * TO PUBLIC` because you will lock yourself out of the `system` database. If you do lock yourself out, follow the disable authentication steps in the Operations Manual to recover and prevent outside access to the instance or cluster.

3. Ensure that the databases cannot be stopped, created, or dropped during the rolling upgrade by using the following command:

   ```
   DENY STOP ON DATABASE * TO PUBLIC
   DENY DATABASE MANAGEMENT ON DBMS TO PUBLIC
   ```

Upgrade the cluster

You upgrade **one cluster member at a time**, while the rest of the members are running.

If during a rolling upgrade the cluster loses quorum and cannot be recovered, then downtime may be required to do a disaster recovery.

For each cluster member

1. (Recommended) Use the process described in the status endpoint to evaluate whether it is safe to remove the old instance.

2. Shut down the instance.

3. Install the *Neo4j* version that you want to upgrade to. For more information on how to install the distribution that you are using, see Operations Manual 4.4 → Installation.
4. Replace the *neo4j.conf* file with the one that you have prepared for this instance in Prepare a new *neo4j.conf* file to be used by the new installation and set `dbms.allow_upgrade=true` to allow automatic store upgrade.
5. Start the new instance and wait for it to catch up with the rest of the cluster members.
6. Verify that the new instance has successfully joined the cluster and caught up with the rest of the members, by using the status endpoint.
7. In the *neo4j.conf* file, configure the following settings:
   a. Set `dbms.allow_upgrade=false` to disable the automatic store upgrade.
   b. Restore any custom value of `dbms.record_format` if it was previously removed.

Because Read Replicas are not part of the cluster consensus group, their replacement during an upgrade does not affect the cluster availability and fault tolerance level. However, it is still recommended to incrementally add Read Replicas for a structured and maintainable upgrade process.

Upgrade the `system` database

In 4.x versions, Neo4j uses a shared `system` database, which includes complex information, such as the security configuration for users, roles, and their privileges. The structure of the graph contained in the `system` database changes with each new version of Neo4j as the capabilities of the DBMS grow. Therefore, each time a Neo4j deployment is upgraded, the contents of the `system` database, or schema, must be transformed as well. When performing an offline upgrade of a single deployment or a Causal cluster, these changes happen automatically, as a consequence of configuring `dbms.mode=SINGLE` (See Prepare for the upgrade and Upgrade your cluster). However, when performing a rolling upgrade, you never start instances with the configuration value `dbms.mode=SINGLE`, i.e., updating the `system` database automatically is not possible.

Any specific metrics that you want to be enabled **must** be specified in the `metrics.filter`.
For more information, see Operations Manual 4.4 → Enable metrics logging.

Compatibility and synchronization

With a causal cluster of many instances, while upgrading each instance in turn, there is a period during which the cluster is composed of some old and some new instances. A single `system` database is consistently replicated across the entire cluster. As a result, it is not possible to have its schema structured according to the needs of the new Neo4j version on some instances and the old version on others.

When the `system` database is not up-to-date with the Neo4j version of a given instance, that instance runs in *compatibility mode*. This means that capabilities common to both Neo4j versions will continue to work, but the features that require the new schema, will be disabled. For example, if you attempt to grant a new privilege not supported in the old schema, you will receive an error and the grant will fail. Therefore, when the rolling upgrade finishes, you must manually upgrade the `system` database schema in order to access all new features.

> If the `system` database's schema is too old to allow *compatibility mode*, the server will not start. For more information, see Troubleshooting.

Manually trigger an upgrade of the `system` database

1. On one of the cluster members, call the procedure `dbms.upgradeStatus()` to determine whether an upgrade is necessary:

```
CALL dbms.upgradeStatus();
```

```
+-----------------------------------------------------------------------------------------------------
--------------------+
| status              | description                                                               |
resolution           |
+-----------------------------------------------------------------------------------------------------
--------------------+
| "REQUIRES_UPGRADE" | "The sub-graph is supported, but is an older version and requires upgrade" |
"CALL dbms.upgrade()" |
+-----------------------------------------------------------------------------------------------------
--------------------+
```

> For the full list of possible status values, see Status codes for `dbms.upgradeStatus`.

2. On one of the cluster members, perform the upgrade by calling the procedure `dbms.upgrade()` against the `system` database:

```
CALL dbms.upgrade();
```

```
+----------------------------+
| status    | upgradeResult |
+----------------------------+
| "CURRENT" | "Success"     |
+----------------------------+
```

Since Neo4j uses a shared `system` database, the upgraded `system` database will replicate across the entire cluster. If the upgrade fails for some reason, the status will not change, and the `upgradeResult` field will describe which components have failed to upgrade.

Post-upgrade steps

The following steps must be performed after a rolling upgrade.

1. Restore the privilege of the `PUBLIC` role to stop databases:

```
REVOKE DENY STOP ON DATABASE * FROM PUBLIC
```

2. Restore the privilege of the `PUBLIC` role to create and drop databases:

```
REVOKE DENY DATABASE MANAGEMENT ON DBMS FROM PUBLIC
```

3. (Optional) If you have started offline databases during the preparation phase for a rolling upgrade, you stop each of them to restore them to the original state:

```
STOP DATABASE [database-name]
```

4. (Recommended) Perform a *full backup*, using an empty target directory.

Rolling upgrade for cloud infrastructure

This variant is suitable for deployments that use replaceable cloud or container resources. It follows the same steps as for the fixed number of servers, but **you can add the new members before you shut down the old ones**, thus preserving the cluster fault tolerance level. Because Read Replicas are not part of the cluster consensus group, their replacement during the upgrade will not affect the cluster availability and fault tolerance level. However, it is still recommended to incrementally add Read Replicas for a structured and maintainable upgrade process.

7.5.4. Troubleshooting

Troubleshooting information that can help you diagnose and correct a problem.

You can review the Operations Manual → Neo4j log files to monitor the upgrade process and verify that no problems occur as it progresses. If a problem occurs, the following known troubleshooting information can help you diagnose and correct the problem.

Solutions to common problems

Calling `dbms.upgrade` results in a failure

The `upgradeResult` field describes which components have failed to upgrade.

Solution

To troubleshoot the root cause there are two things to try:

- Call the more verbose version of the command — `dbms.upgradeDetails`, which returns a message with the underlying exception thrown. This might not be the root cause, but can help narrow it down.
- Look for descriptive warnings and errors in the *security.log* and the *debug.log*. Stack traces should help track down the root cause of the problem.

Trying to upgrade between incompatible versions, for example migrating from Neo4j 3.5 to Neo4j 4.1

When the detected version of the `system` database is too old, the status code is one of the following:

- `UNSUPPORTED_BUT_CAN_UPGRADE` - the server shuts down.

  Solution

  Configure `dbms.mode=SINGLE`. This enables the automatic upgrade of the `system` database schema as the setting `dbms.allow_single_automatic_upgrade` is `true` by default when `dbms.mode=SINGLE`.

- `UNSUPPORTED` - the server shuts down.

  Solution

  No direct upgrade is possible, upgrade first to a supported intermediate version.

Calling `dbms.upgrade` when not all instances have been upgraded to the new version of Neo4j

- Calling `dbms.upgradeStatus` on any instances still running the older version returns `UNSUPPORTED_FUTURE`.
- These older instances are no longer able to authorize users.
- All queries against these instances result in an error.

  Solution

  Shut down, upgrade, and restart the instance in order to regain service.

Status codes for `dbms.upgradeStatus`

System component status

Status	Description	Resolution
`UNINITIALIZED`	No sub-graph detected for this component.	Requires initialization.
`CURRENT`	The sub-graph is already at the current version.	No action required.
`REQUIRES_UPGRADE`	The sub-graph is supported but is an older version and requires upgrade.	Run `CALL dbms.upgrade()`.
`UNSUPPORTED_BUT_CAN_UPGRADE`	The sub-graph is unsupported; this component cannot function.	Restart Neo4j in a single-instance mode to upgrade this component at startup.
`UNSUPPORTED`	The sub-graph is unsupported because it is too old; this component cannot function.	Downgrade Neo4j and then upgrade this component before upgrading Neo4j again.
`UNSUPPORTED_FUTURE`	The sub-graph is unsupported because it is a newer version; this component cannot function.	Upgrade Neo4j.

7.6. Upgrade 4.2 to 4.3 `MINOR` version

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

It covers the following topics:

- Upgrading Neo4j 4.2 to 4.3 `MINOR` version.
  - Upgrade checklist
  - Upgrade a single instance
  - Upgrade a Causal Cluster
  - Troubleshooting

7.6.1. Upgrade checklist

This chapter describes how to prepare to upgrade Neo4j from an earlier minor or patch version.

ℹ	Before you start preparing for the upgrade, it is very important to read the information in Understanding upgrades and migration and Supported upgrade and migration paths.

Follow the checklist to prepare for upgrading your Neo4j deployment:

- ☐ Complete all prerequisites for the upgrade.
- ☐ Back up your current deployment to avoid losing data in case of failure.
- ☐ Download the new version of Neo4j. Make sure the upgrade path is supported.
- ☐ Prepare a new neo4j.conf file to be used by the new installation.
- ☐ Perform a test upgrade as per your Neo4j version and deployment type (Single instance or Causal Cluster).
- ☐ Monitor the logs.
- ☐ Perform the upgrade.

ℹ	If you are upgrading a Causal Cluster, complete the checklist for each cluster member.

Prerequisites

1. Verify that you have installed Java 11.
2. Review the improvements and fixes that have been carried out in the version that you want to upgrade to.
   - Neo4j 4.3 Change log
   - Release Notes

    - Changes to configuration settings in Neo4j 4
    - Changes to procedures in Neo4j 4

Back up your current deployment

Depending on your particular deployment and environment, it is important to design an appropriate backup and restore strategy. For more information about what to consider when deciding on your backup and restore strategy, what needs to be backed up, and the different backup modes and options, see Operations Manual → Backup and restore planning.

Prepare a new *neo4j.conf* file to be used by the new installation

Prepare a *neo4j.conf* file to be used by the new installation if you have any custom values configured in your old *neo4j.conf* file. Look for:

- Any non-default settings from your old installation.
- Any custom values of the settings `dbms.directories.*` and `dbms.default_database`.
- (In cluster installations) Cluster-specific configuration settings, which might be different for the different cluster members.

Perform a test upgrade

Based on the findings in this chapter, allocate a staging test environment for the upgrade and do a test upgrade. The test upgrade will give you valuable information about the time required for the production upgrade. Follow the steps as per your Neo4j version and deployment type.

Monitor the logs

The neo4j.log file contains information on how many steps the upgrade will involve and how far it has progressed. It is a good idea to monitor this log.

7.6.2. Upgrade a single instance to 4.3

This section describes how to upgrade a single Neo4j instance.

Prerequisites

Ensure that you have completed all tasks on the Upgrade checklist.

Prepare for the upgrade

1. Install the Neo4j version that you want to upgrade to. For more information on how to install the distribution that you are using, see Operations Manual → Installation.
2. Replace the *neo4j.conf* file with the one that you have prepared in section Prepare a new *neo4j.conf* file to be used by the new installation.

3. Set `dbms.allow_upgrade=true` to allow automatic store upgrade.
4. Set `dbms.mode=SINGLE`. This enables the automatic upgrade of the `system` database schema, as the setting `dbms.allow_single_automatic_upgrade` is `true` by default when `dbms.mode=SINGLE`.
5. Copy all the files used for encryption, such as private key, public certificate, and the contents of the trusted and revoked directories (located in *<neo4j-home>/certificates/*).
6. Restore each of your databases and transactions in the new installation, including the `system` database, by either using `neo4j-admin restore` (online) or `neo4j-admin load` (offline), depending on your backup approach. If you are running a Debian/RPM distribution, you can skip this step.

   If your old installation has modified configurations starting with `dbms.directories.*` or the setting `dbms.default_database`, verify that the new `neo4j.conf` file is configured properly to find these directories.

7. If you are using custom plugins, make sure they are updated and compatible with the new version, and place them in the */plugins* directory.

Upgrade your instance

1. Start the instance by running the following command from `<neo4j-home>`:

   ```
   bin/neo4j start
   ```

   The upgrade takes place during startup.

2. Monitor the neo4j.log file for information on how many steps the upgrade involves and how far it has progressed.

Post-upgrade tasks

1. When the upgrade finishes, open the *neo4j.conf* file and set `dbms.allow_upgrade=false`. If, for some reason, you have forgotten to enable the automatic upgrade of the `system` database schema, use the following commands to manually upgrade it:
   - `CALL dbms.upgradeStatus()` to determine whether an upgrade is necessary or not.
   - `CALL dbms.upgrade()` on the `system` database to perform the upgrade of the system schema. For more details, see Operations Manual 4.3 → Procedures.
2. Restart the instance by running the following command from `<neo4j-home>`:

   ```
   bin/neo4j restart
   ```

3. It is good practice to make a full backup immediately after the upgrade.
4. Check the default settings on all metrics. Any specific metrics that you want to be enabled **must** be specified in the `metrics.filter`. For more information, see Operations Manual → Enable metrics logging.

7.6.3. Upgrade a Causal Cluster to 4.3

This section describes how to upgrade a Neo4j Causal Cluster.

You can upgrade your existing Neo4j Causal Cluster by either performing a rolling upgrade, or by upgrading it offline.

ℹ	The prerequisites and the upgrade steps must be completed for each cluster member.

Offline upgrade

This variant is suitable for cases where a rolling upgrade is not possible.

ℹ	It is recommended to perform a test upgrade on a production-like environment to get information on the duration of the downtime.

Prerequisites

Ensure that you have completed all tasks on the Upgrade checklist.

⚠	When performing a rolling upgrade onto Neo4j 4.3, ensure that the version upgrading from is the latest release of 4.2, or the upgrade may fail.

Prepare for the upgrade

1. Shut down all the cluster members (Cores and Read Replicas).
2. Perform `neo4j-admin unbind` on each cluster member to remove cluster state data.
3. Install the Neo4j version that you want to upgrade to on each instance. For more information on how to install the distribution that you are using, see Operations Manual 4.3 → Installation.
4. Replace the *neo4j.conf* file with the one that you have prepared for each instance in section Prepare a new *neo4j.conf* file to be used by the new installation.
5. Copy all the files used for encryption, such as private key, public certificate, and the contents of the trusted and revoked directories (located in *<neo4j-home>/certificates/*).
6. Restore each of your databases and transactions in the new installation, including the `system` database, by either using `neo4j-admin restore` (online) or `neo4j-admin load` (offline), depending on your backup approach.

ℹ	If your old installation has modified configurations starting with `dbms.directories.*` or the setting `dbms.default_database`, verify that the new `neo4j.conf` file is configured properly to find these directories.

7. If you are using custom plugins, make sure they are updated and compatible with the new version, and

place them in the */plugins* directory.

Upgrade your cluster

*On **one** of the Cores*

1. Open the *neo4j.conf* file of the new installation and configure the following settings:
   - Uncomment `dbms.allow_upgrade=true` to allow automatic store upgrade.
   - Set `dbms.mode=SINGLE`. This enables the automatic upgrade of the `system` database schema, as the setting `dbms.allow_single_automatic_upgrade` is `true` by default when `dbms.mode=SINGLE`.
2. Start Neo4j by running the following command from `<neo4j-home>`:

   ```
   bin/neo4j start
   ```

   The upgrade takes place during startup.
3. Monitor the *neo4j.log* file for information on how many steps the upgrade will involve and how far it has progressed.
4. When the upgrade finishes, stop the server.
5. Open the *neo4j.conf* file and configure the following settings:
   - Set `dbms.allow_upgrade=false`.
   - Set `dbms.mode=CORE` to re-enable Causal Clustering in the configuration.
6. Use `neo4j-admin dump` to make a dump of each of your databases and transactions, including the `system` database.
7. Do **not** yet start the server.

On each of the other Cores

1. Copy the database dumps you created on the first Core server to each of the other Cores.
2. Use `neo4j-admin load --from=<archive-path> --database=<database> --force` to replace each of your databases, including the `system` database, with the ones you upgraded on the first Core server.
3. Start each of the core servers, including the first one, and verify that they join in a cluster.

For each Read Replica

Start the Read Replica and wait for it to catch up with the rest of the cluster members.

(Optional) Whilst an empty read replica will eventually get a full copy of all data from the other members of your cluster, this catching up may take some time. In order to speed up the process, you can load the data first by using `neo4j-admin load --from=<archive-path> --database=<database> --force` to replace each of your databases with the upgraded ones, including the `system` database.

Verify that the Read Replicas join the cluster.

Post-upgrade

It is recommended to perform a *full backup*, using an empty target directory.

Rolling upgrade

Rolling upgrade is a zero-downtime method for upgrading a Causal Cluster. You upgrade one member at a time, while the rest of the members are running. However, if during a rolling upgrade the cluster loses quorum and cannot be recovered, then downtime may be required to do a disaster recovery.

Recommendations

- The critical point during the upgrade is knowing when it is safe to switch off the original member. It is highly recommended to monitor the status endpoint before each removal, in order to decide which member to switch off and when it is safe to do so.
- To reduce the risk of failure during a rolling upgrade, make sure the cluster is not under any heavy load during the upgrade. If possible, the safest would be to disable writes entirely.
- There should be no changes to database administration during a rolling upgrade. For more information, see Operations Manual 4.3 → Manage databases.
- Remove `dbms.record_format` from *neo4j.conf* to avoid any accidental cross-format migration.

Rolling upgrade for a fixed number of servers

This variant is suitable for deployments where there is a fixed number of servers and they have to be updated in-place.

> When performing a rolling upgrade for a fixed number of servers, it is not possible to increase the cluster size. Therefore, the cluster fault tolerance level will be reduced while replacing the members.

Prerequisites

1. Ensure that you have completed all tasks on the Upgrade checklist.

   > When performing a rolling upgrade onto Neo4j 4.3, ensure that the version upgrading from is the latest release of 4.2, or the upgrade may fail.

2. Verify that **all databases are online** by running `SHOW DATABASES` in Cypher Shell or Neo4j Browser. Offline databases can be started using `START DATABASE [database-name]`.

All databases must be started before you start a rolling upgrade. If you have to keep a database inaccessible during the rolling upgrade, you can disable access to it by using the command:

```
DENY ACCESS ON DATABASE [database-name] TO PUBLIC
```

You must never run `DENY ACCESS ON DATABASE system TO PUBLIC` or `DENY ACCESS ON DATABASE * TO PUBLIC` because you will lock yourself out of the `system` database. If you do lock yourself out, follow the disable authentication steps in the Operations Manual to recover and prevent outside access to the instance or cluster.

3. Ensure that the databases cannot be stopped, created, or dropped during the rolling upgrade by using the following command:

```
DENY STOP ON DATABASE * TO PUBLIC
DENY DATABASE MANAGEMENT ON DBMS TO PUBLIC
```

Upgrade the cluster

You upgrade **one cluster member at a time**, while the rest of the members are running.

If during a rolling upgrade the cluster loses quorum and cannot be recovered, then downtime may be required to do a disaster recovery.

For each cluster member

1. (Recommended) Use the process described in the status endpoint to evaluate whether it is safe to remove the old instance.
2. Shut down the instance.
3. Install the *Neo4j* version that you want to upgrade to. For more information on how to install the distribution that you are using, see Operations Manual 4.3 → Installation.
4. Replace the *neo4j.conf* file with the one that you have prepared for this instance in Prepare a new *neo4j.conf* file to be used by the new installation and set `dbms.allow_upgrade=true` to allow automatic store upgrade.
5. Start the new instance and wait for it to catch up with the rest of the cluster members.
6. Verify that the new instance has successfully joined the cluster and caught up with the rest of the members, by using the status endpoint.
7. In the *neo4j.conf* file, configure the following settings:
   a. Set `dbms.allow_upgrade=false` to disable the automatic store upgrade.
   b. Restore any custom value of `dbms.record_format` if it was previously removed.

Because Read Replicas are not part of the cluster consensus group, their replacement during an upgrade does not affect the cluster availability and fault tolerance level. However, it is still recommended to incrementally add Read Replicas for a structured and maintainable upgrade process.

Upgrade the `system` database

In 4.x versions, Neo4j uses a shared `system` database, which includes complex information, such as the security configuration for users, roles, and their privileges. The structure of the graph contained in the `system` database changes with each new version of Neo4j as the capabilities of the DBMS grow. Therefore, each time a Neo4j deployment is upgraded, the contents of the `system` database, or schema, must be transformed as well. When performing an offline upgrade of a single deployment or a Causal cluster, these changes happen automatically, as a consequence of configuring `dbms.mode=SINGLE` (See Prepare for the upgrade and Upgrade your cluster). However, when performing a rolling upgrade, you never start instances with the configuration value `dbms.mode=SINGLE`, i.e., updating the `system` database automatically is not possible.

Any specific metrics that you want to be enabled **must** be specified in the `metrics.filter`.
For more information, see Operations Manual 4.3 → Enable metrics logging.

Compatibility and synchronization

With a causal cluster of many instances, while upgrading each instance in turn, there is a period during which the cluster is composed of some old and some new instances. A single `system` database is consistently replicated across the entire cluster. As a result, it is not possible to have its schema structured according to the needs of the new Neo4j version on some instances and the old version on others.

When the `system` database is not up-to-date with the Neo4j version of a given instance, that instance runs in *compatibility mode*. This means that capabilities common to both Neo4j versions will continue to work, but the features that require the new schema, will be disabled. For example, if you attempt to grant a new privilege not supported in the old schema, you will receive an error and the grant will fail. Therefore, when the rolling upgrade finishes, you must manually upgrade the `system` database schema in order to access all new features.

If the `system` database's schema is too old to allow *compatibility mode*, the server will not start. For more information, see Troubleshooting.

Manually trigger an upgrade of the `system` database

1. On one of the cluster members, call the procedure `dbms.upgradeStatus()` to determine whether an upgrade is necessary:

```
CALL dbms.upgradeStatus();
```

```
+---------------------------------------------------------------------------------------------------
--------------------+
| status             | description                                                              |
resolution           |
+---------------------------------------------------------------------------------------------------
--------------------+
| "REQUIRES_UPGRADE" | "The sub-graph is supported, but is an older version and requires upgrade" |
"CALL dbms.upgrade()" |
+---------------------------------------------------------------------------------------------------
--------------------+
```

For the full list of possible status values, see Status codes for `dbms.upgradeStatus`.

2. On one of the cluster members, perform the upgrade by calling the procedure `dbms.upgrade()` against the `system` database:

```
CALL dbms.upgrade();
```

```
+----------------------------+
| status    | upgradeResult |
+----------------------------+
| "CURRENT" | "Success"     |
+----------------------------+
```

Since Neo4j uses a shared `system` database, the upgraded `system` database will replicate across the entire cluster. If the upgrade fails for some reason, the status will not change, and the `upgradeResult` field will describe which components have failed to upgrade.

Post-upgrade steps

The following steps must be performed after a rolling upgrade.

1. Restore the privilege of the `PUBLIC` role to stop databases:

```
REVOKE DENY STOP ON DATABASE * FROM PUBLIC
```

2. Restore the privilege of the `PUBLIC` role to create and drop databases:

```
REVOKE DENY DATABASE MANAGEMENT ON DBMS FROM PUBLIC
```

3. (Optional) If you have started offline databases during the preparation phase for a rolling upgrade, you stop each of them to restore them to the original state:

```
STOP DATABASE [database-name]
```

4. (Recommended) Perform a *full backup*, using an empty target directory.

Rolling upgrade for cloud infrastructure

This variant is suitable for deployments that use replaceable cloud or container resources. It follows the same steps as for the fixed number of servers, but **you can add the new members before you shut down the old ones**, thus preserving the cluster fault tolerance level. Because Read Replicas are not part of the cluster consensus group, their replacement during the upgrade will not affect the cluster availability and fault tolerance level. However, it is still recommended to incrementally add Read Replicas for a structured and maintainable upgrade process.

7.6.4. Troubleshooting

Troubleshooting information that can help you diagnose and correct a problem.

You can review the Operations Manual → Neo4j log files to monitor the upgrade process and verify that no problems occur as it progresses. If a problem occurs, the following known troubleshooting information can help you diagnose and correct the problem.

Solutions to common problems

Calling `dbms.upgrade` results in a failure

The `upgradeResult` field describes which components have failed to upgrade.

Solution

To troubleshoot the root cause there are two things to try:

- Call the more verbose version of the command — `dbms.upgradeDetails`, which returns a message with the underlying exception thrown. This might not be the root cause, but can help narrow it down.
- Look for descriptive warnings and errors in the *security.log* and the *debug.log*. Stack traces should help track down the root cause of the problem.

Trying to upgrade between incompatible versions, for example migrating from Neo4j 3.5 to Neo4j 4.1

When the detected version of the `system` database is too old, the status code is one of the following:

- `UNSUPPORTED_BUT_CAN_UPGRADE` - the server shuts down.

  Solution

  Configure `dbms.mode=SINGLE`. This enables the automatic upgrade of the `system` database schema as the setting `dbms.allow_single_automatic_upgrade` is `true` by default when `dbms.mode=SINGLE`.

- `UNSUPPORTED` - the server shuts down.

  Solution

  No direct upgrade is possible, upgrade first to a supported intermediate version.

Calling `dbms.upgrade` when not all instances have been upgraded to the new version of Neo4j

- Calling `dbms.upgradeStatus` on any instances still running the older version returns `UNSUPPORTED_FUTURE`.
- These older instances are no longer able to authorize users.
- All queries against these instances result in an error.

  Solution

  Shut down, upgrade, and restart the instance in order to regain service.

Status codes for `dbms.upgradeStatus`

System component status

Status	Description	Resolution
`UNINITIALIZED`	No sub-graph detected for this component.	Requires initialization.
`CURRENT`	The sub-graph is already at the current version.	No action required.
`REQUIRES_UPGRADE`	The sub-graph is supported but is an older version and requires upgrade.	Run `CALL dbms.upgrade()`.
`UNSUPPORTED_BUT_CAN_UPGRADE`	The sub-graph is unsupported; this component cannot function.	Restart Neo4j in a single-instance mode to upgrade this component at startup.
`UNSUPPORTED`	The sub-graph is unsupported because it is too old; this component cannot function.	Downgrade Neo4j and then upgrade this component before upgrading Neo4j again.
`UNSUPPORTED_FUTURE`	The sub-graph is unsupported because it is a newer version; this component cannot function.	Upgrade Neo4j.

7.7. Upgrade 4.1 to 4.2 `MINOR` version

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

It covers the following topics:

- Upgrading Neo4j 4.1 to 4.2.
  - Upgrade checklist
  - Upgrade a single instance
  - Upgrade a Causal Cluster
  - Troubleshooting

7.7.1. Upgrade checklist

This chapter describes how to prepare to upgrade Neo4j from an earlier minor or patch version.

Before you start preparing for the upgrade, it is very important to read the information in Understanding upgrades and migration and Supported upgrade and migration paths.

Follow the checklist to prepare for upgrading your Neo4j deployment:

- ☐ Complete all prerequisites for the upgrade.
- ☐ Back up your current deployment to avoid losing data in case of failure.
- ☐ Download the new version of Neo4j. Make sure the upgrade path is supported.
- ☐ Prepare a new *neo4j.conf* file to be used by the new installation.
- ☐ Perform a test upgrade as per your Neo4j version and deployment type (Single instance or Causal Cluster).
- ☐ Monitor the logs.
- ☐ Perform the upgrade.

If you are upgrading a Causal Cluster, complete the checklist for each cluster member.

Prerequisites

1. Verify that you have installed Java 11.
2. Review the improvements and fixes that have been carried out in the version that you want to upgrade to.
   - Neo4j 4.2 Change log
   - Release Notes
   - Changes to configuration settings in Neo4j 4
   - Changes to procedures in Neo4j 4

Back up your current deployment

Depending on your particular deployment and environment, it is important to design an appropriate backup and restore strategy. For more information about what to consider when deciding on your backup and restore strategy, what needs to be backed up, and the different backup modes and options, see Operations Manual → Backup and restore planning.

Prepare a new *neo4j.conf* file to be used by the new installation

Prepare a *neo4j.conf* file to be used by the new installation if you have any custom values configured in your old *neo4j.conf* file. Look for:

- Any non-default settings from your old installation.
- Any custom values of the settings `dbms.directories.*` and `dbms.default_database`.
- (In cluster installations) Cluster-specific configuration settings, which might be different for the different cluster members.

Perform a test upgrade

Based on the findings in this chapter, allocate a staging test environment for the upgrade and do a test upgrade. The test upgrade will give you valuable information about the time required for the production upgrade. Follow the steps as per your Neo4j version and deployment type.

Monitor the logs

The *neo4j.log* file contains information on how many steps the upgrade will involve and how far it has progressed. It is a good idea to monitor this log.

7.7.2. Upgrade a single instance to 4.2

This section describes how to upgrade a single Neo4j instance.

Prerequisites

1. Ensure that you have completed all tasks on the Upgrade checklist.

Prepare for the upgrade

1. Install the Neo4j version that you want to upgrade to. For more information on how to install the distribution that you are using, see Operations Manual → Installation.
2. Replace the *neo4j.conf* file with the one that you have prepared in section Prepare a new *neo4j.conf* file to be used by the new installation.
3. Set `dbms.allow_upgrade=true` to allow automatic store upgrade.
4. Set `dbms.mode=SINGLE`. This enables the automatic upgrade of the `system` database schema, as the setting `dbms.allow_single_automatic_upgrade` is `true` by default when `dbms.mode=SINGLE`.
5. Copy all the files used for encryption, such as private key, public certificate, and the contents of the trusted and revoked directories (located in *<neo4j-home>/certificates/*).
6. Restore each of your databases and transactions in the new installation, including the `system` database, by either using `neo4j-admin restore` (online) or `neo4j-admin load` (offline), depending on your backup approach. If you are running a Debian/RPM distribution, you can skip this step.

   > If your old installation has modified configurations starting with `dbms.directories.*` or the setting `dbms.default_database`, verify that the new `neo4j.conf` file is configured properly to find these directories.

7. If you are using custom plugins, make sure they are updated and compatible with the new version, and place them in the */plugins* directory.

Upgrade your instance

1. Start the instance by running the following command from `<neo4j-home>`:

```
bin/neo4j start
```

The upgrade takes place during startup.

2. Monitor the neo4j.log file for information on how many steps the upgrade involves and how far it has progressed.

Post-upgrade tasks

1. When the upgrade finishes, open the *neo4j.conf* file and set `dbms.allow_upgrade=false`. If, for some reason, you have forgotten to enable the automatic upgrade of the `system` database schema, use the following commands to manually upgrade it:
   - `CALL dbms.upgradeStatus()` to determine whether an upgrade is necessary or not.
   - `CALL dbms.upgrade()` on the `system` database to perform the upgrade of the system schema. For more details, see Operations Manual 4.2 → Procedures.
2. Restart the instance by running the following command from `<neo4j-home>`:

```
bin/neo4j restart
```

3. It is good practice to make a full backup immediately after the upgrade.
4. Check the default settings on all metrics. Any specific metrics that you want to be enabled **must** be specified in the `metrics.filter`. For more information, see Operations Manual → Enable metrics logging.

7.7.3. Upgrade a Causal Cluster to 4.2

This section describes how to upgrade a Neo4j Causal Cluster.

You can upgrade your existing Neo4j Causal Cluster by either performing a rolling upgrade, or by upgrading it offline.

ℹ	The prerequisites and the upgrade steps must be completed for each cluster member.

Offline upgrade

This variant is suitable for cases where a rolling upgrade is not possible.

ℹ	It is recommended to perform a test upgrade on a production-like environment to get information on the duration of the downtime.

Prerequisites

1. Ensure that you have completed all tasks on the Upgrade checklist.

Prepare for the upgrade

1. Shut down all the cluster members (Cores and Read Replicas).
2. Perform `neo4j-admin unbind` on each cluster member to remove cluster state data.
3. Install the Neo4j version that you want to upgrade to on each instance. For more information on how to install the distribution that you are using, see Operations Manual 4.2 → Installation.
4. Replace the *neo4j.conf* file with the one that you have prepared for each instance in section Prepare a new neo4j.conf file to be used by the new installation.
5. Copy all the files used for encryption, such as private key, public certificate, and the contents of the trusted and revoked directories (located in *<neo4j-home>/certificates/*).
6. Restore each of your databases and transactions in the new installation, including the `system` database, by either using `neo4j-admin restore` (online) or `neo4j-admin load` (offline), depending on your backup approach. If you are running a Debian/RPM distribution, you can skip this step.

   If your old installation has modified configurations starting with `dbms.directories.*` or the setting `dbms.default_database`, verify that the new `neo4j.conf` file is configured properly to find these directories.

7. If you are using custom plugins, make sure they are updated and compatible with the new version, and place them in the */plugins* directory.

Upgrade your cluster

*On **one** of the Cores*

1. Open the *neo4j.conf* file of the new installation and configure the following settings:
   - Uncomment `dbms.allow_upgrade=true` to allow automatic store upgrade.
   - Set `dbms.mode=SINGLE`. This enables the automatic upgrade of the `system` database schema, as the setting `dbms.allow_single_automatic_upgrade` is `true` by default when `dbms.mode=SINGLE`.
2. Start Neo4j by running the following command from `<neo4j-home>`:

   ```
   bin/neo4j start
   ```

   The upgrade takes place during startup.
3. Monitor the neo4j.log file for information on how many steps the upgrade will involve and how far it has progressed.
4. When the upgrade finishes, stop the server.
5. Open the *neo4j.conf* file and configure the following settings:
   - Set `dbms.allow_upgrade=false`.
   - Set `dbms.mode=CORE` to re-enable Causal Clustering in the configuration.
6. Use `neo4j-admin dump` to make a dump of each of your databases and transactions, including the `system` database.

7. Do **not** yet start the server.

On each of the other Cores

1. Copy the database dumps you created on the first Core server to each of the other Cores.
2. Use `neo4j-admin load --from=<archive-path> --database=<database> --force` to replace each of your databases, including the `system` database, with the ones you upgraded on the first Core server.
3. Start each of the core servers, including the first one, and verify that they join in a cluster.

For each Read Replica

Start the Read Replica and wait for it to catch up with the rest of the cluster members.

> (Optional) Whilst an empty read replica will eventually get a full copy of all data from the other members of your cluster, this catching up may take some time. In order to speed up the process, you can load the data first by using `neo4j-admin load --from=<archive-path> --database=<database> --force` to replace each of your databases with the upgraded ones, including the `system` database.

Verify that the Read Replicas join the cluster.

Post-upgrade

It is recommended to perform a *full backup*, using an empty target directory.

Rolling upgrade

Rolling upgrade is a zero-downtime method for upgrading a Causal Cluster. You upgrade one member at a time, while the rest of the members are running. However, if during a rolling upgrade the cluster loses quorum and cannot be recovered, then downtime may be required to do a disaster recovery.

Recommendations

- The critical point during the upgrade is knowing when it is safe to switch off the original member. It is highly recommended to monitor the status endpoint before each removal, in order to decide which member to switch off and when it is safe to do so.
- To reduce the risk of failure during a rolling upgrade, make sure the cluster is not under any heavy load during the upgrade. If possible, the safest would be to disable writes entirely.
- There should be no changes to database administration during a rolling upgrade. For more information, see Operations Manual 4.2 → Manage databases.

Rolling upgrade for a fixed number of servers

This variant is suitable for deployments where there is a fixed number of servers and they have to be updated in-place.

When performing a rolling upgrade for a fixed number of servers, it is not possible to increase the cluster size. Therefore, the cluster fault tolerance level will be reduced while replacing the members.

Prerequisites

1. Ensure that you have completed all tasks on the Upgrade checklist.
2. Verify that **all databases are online** by running `SHOW DATABASES` in Cypher Shell or Neo4j Browser. Offline databases can be started using `START DATABASE [database-name]`.

   All databases must be started before you start a rolling upgrade. If you have to keep a database inaccessible during the rolling upgrade, you can disable access to it by using the following command:

   ```
   DENY ACCESS ON DATABASE [database-name] TO PUBLIC
   ```

   You must never run `DENY ACCESS ON DATABASE system TO PUBLIC` or `DENY ACCESS ON DATABASE * TO PUBLIC` because you will lock yourself out of the `system` database. If you do lock yourself out, follow the disable authentication steps in the Operations Manual to recover and prevent outside access to the instance or cluster.

   When upgrading from Neo4j 4.0.x, you have to disable access to each role that have access to that particular database, as the `PUBLIC` will not yet exist:

   ```
   DENY ACCESS ON DATABASE [database-name] TO [role1],[role2]
   ```

   All available roles can be queried with `SHOW ROLES`.
3. Ensure that the databases cannot be stopped, created, or dropped during the rolling upgrade by using the following command:

   ```
   DENY STOP ON DATABASE * TO PUBLIC
   DENY DATABASE MANAGEMENT ON DBMS TO PUBLIC
   ```

   When upgrading from Neo4j 4.0.x, you can only disable the ability to stop databases.

   ```
   DENY STOP ON DATABASE * TO admin
   ```

   This must be done for the `admin` role and all other roles that have the privilege to stop databases. For more information about listing privileges, see Cypher Manual 4.2 → Graph and sub-graph access control.

Upgrade the cluster

You upgrade **one cluster member at a time**, while the rest of the members are running.

⚠	If during a rolling upgrade the cluster loses quorum and cannot be recovered, then downtime may be required to do a disaster recovery.

For each cluster member

1. (Recommended) Use the process described in the status endpoint to evaluate whether it is safe to remove the old instance.
2. Shut down the instance.
3. Install the *Neo4j* version that you want to upgrade to. For more information on how to install the distribution that you are using, see Operations Manual 4.2 → Installation.
4. Replace the *neo4j.conf* file with the one that you have prepared for this instance in Prepare a new *neo4j.conf* file to be used by the new installation.
5. Start the new instance and wait for it to catch up with the rest of the cluster members.
6. Verify that the new instance has successfully joined the cluster and caught up with the rest of the members, by using the status endpoint.

ℹ	Because Read Replicas are not part of the cluster consensus group, their replacement during an upgrade does not affect the cluster availability and fault tolerance level. However, it is still recommended to incrementally add Read Replicas for a structured and maintainable upgrade process.

Upgrade the `system` database

In 4.x versions, Neo4j uses a shared `system` database, which includes complex information, such as the security configuration for users, roles, and their privileges. The structure of the graph contained in the `system` database changes with each new version of Neo4j as the capabilities of the DBMS grow. Therefore, each time a Neo4j deployment is upgraded, the contents of the `system` database, or schema, must be transformed as well. When performing an offline upgrade of a single deployment or a Causal cluster, these changes happen automatically, as a consequence of configuring `dbms.mode=SINGLE` (See Prepare for the upgrade and Upgrade your cluster). However, when performing a rolling upgrade, you never start instances with the configuration value `dbms.mode=SINGLE`, i.e., updating the `system` database automatically is not possible.

ℹ	The metrics that are enabled by default have been changed in Neo4j 4.2. Any specific metrics that you want to be enabled **must** be specified in the `metrics.filter`. For more information, see Operations Manual 4.2 → Enable metrics logging.

Compatibility and synchronization

With a causal cluster of many instances, while upgrading each instance in turn, there is a period during which the cluster is composed of some old and some new instances. A single `system` database is consistently replicated across the entire cluster. As a result, it is not possible to have its schema structured according to the needs of the new Neo4j version on some instances and the old version on others.

When the `system` database is not up-to-date with the Neo4j version of a given instance, that instance runs in *compatibility mode*. This means that capabilities common to both Neo4j versions will continue to work, but the features that require the new schema, will be disabled. For example, if you attempt to grant a new privilege not supported in the old schema, you will receive an error and the grant will fail. Therefore, when the rolling upgrade finishes, you must manually upgrade the `system` database schema in order to access all new features.

If the `system` database's schema is too old to allow *compatibility mode*, the server will not start. For more information, see Troubleshooting.

Manually trigger an upgrade of the `system` database

1. On one of the cluster members, call the procedure `dbms.upgradeStatus()` to determine whether an upgrade is necessary:

   ```
   CALL dbms.upgradeStatus();
   ```

   ```
   +----------------------------------------------------------------------------------------------------
   --------------------+
   | status              | description                                                              |
   resolution           |
   +----------------------------------------------------------------------------------------------------
   --------------------+
   | "REQUIRES_UPGRADE" | "The sub-graph is supported, but is an older version and requires upgrade" |
   "CALL dbms.upgrade()" |
   +----------------------------------------------------------------------------------------------------
   --------------------+
   ```

   For the full list of possible status values, see Status codes for `dbms.upgradeStatus`.

2. On one of the cluster members, perform the upgrade by calling the procedure `dbms.upgrade()` against the `system` database:

   ```
   CALL dbms.upgrade();
   ```

   ```
   +----------------------------+
   | status    | upgradeResult |
   +----------------------------+
   | "CURRENT" | "Success"     |
   +----------------------------+
   ```

   Since Neo4j uses a shared `system` database, the upgraded `system` database will replicate across the entire cluster. If the upgrade fails for some reason, the status will not change, and the `upgradeResult` field will describe which components have failed to upgrade.

Post-upgrade steps

The following steps must be performed after a rolling upgrade.

1. Restore the privilege of the `PUBLIC` role to stop databases:

   ```
   REVOKE DENY STOP ON DATABASE * FROM PUBLIC
   ```

2. Restore the privilege of the `PUBLIC` role to create and drop databases:

   ```
   REVOKE DENY DATABASE MANAGEMENT ON DBMS FROM PUBLIC
   ```

3. (Optional) If you have started offline databases and denied some access rights during the preparation phase for a rolling upgrade, you should also restore them to the original state:

   a. Stop each of the databases by running the following command:

      ```
      STOP DATABASE [database-name]
      ```

   b. Re-enable access to the databases by running the following command:

      ```
      REVOKE DENY ACCESS ON DATABASE [database-name] FROM [role1],[role2]
      ```

4. (Recommended) Perform a *full backup*, using an empty target directory.

Rolling upgrade for cloud infrastructure

This variant is suitable for deployments that use replaceable cloud or container resources. It follows the same steps as for the fixed number of servers, but **you can add the new members before you shut down the old ones**, thus preserving the cluster fault tolerance level. Because Read Replicas are not part of the cluster consensus group, their replacement during the upgrade will not affect the cluster availability and fault tolerance level. However, it is still recommended to incrementally add Read Replicas for a structured and maintainable upgrade process.

7.7.4. Troubleshooting

Troubleshooting information that can help you diagnose and correct a problem.

You can review the Operations Manual → Neo4j log files to monitor the upgrade process and verify that no problems occur as it progresses. If a problem occurs, the following known troubleshooting information can help you diagnose and correct the problem.

Solutions to common problems

Calling `dbms.upgrade` results in a failure

The `upgradeResult` field describes which components have failed to upgrade.

Solution

To troubleshoot the root cause there are two things to try:

- Call the more verbose version of the command — `dbms.upgradeDetails`, which returns a message with the underlying exception thrown. This might not be the root cause, but can help narrow it down.
- Look for descriptive warnings and errors in the *security.log* and the *debug.log*. Stack traces should help track down the root cause of the problem.

Trying to upgrade between incompatible versions, for example migrating from Neo4j 3.5 to Neo4j 4.1

When the detected version of the `system` database is too old, the status code is one of the following:

- `UNSUPPORTED_BUT_CAN_UPGRADE` - the server shuts down.

  Solution

  Configure `dbms.mode=SINGLE`. This enables the automatic upgrade of the `system` database schema as the setting `dbms.allow_single_automatic_upgrade` is `true` by default when `dbms.mode=SINGLE`.

- `UNSUPPORTED` - the server shuts down.

  Solution

  No direct upgrade is possible, upgrade first to a supported intermediate version.

Calling `dbms.upgrade` when not all instances have been upgraded to the new version of Neo4j

- Calling `dbms.upgradeStatus` on any instances still running the older version returns `UNSUPPORTED_FUTURE`.
- These older instances are no longer able to authorize users.
- All queries against these instances result in an error.

  Solution

  Shut down, upgrade, and restart the instance in order to regain service.

Status codes for `dbms.upgradeStatus`

System component status

Status	Description	Resolution
`UNINITIALIZED`	No sub-graph detected for this component.	Requires initialization.
`CURRENT`	The sub-graph is already at the current version.	No action required.

Status	Description	Resolution
REQUIRES_UPGRADE	The sub-graph is supported but is an older version and requires upgrade.	Run `CALL dbms.upgrade()`.
UNSUPPORTED_BUT_CAN_UPGRADE	The sub-graph is unsupported; this component cannot function.	Restart Neo4j in a single-instance mode to upgrade this component at startup.
UNSUPPORTED	The sub-graph is unsupported because it is too old; this component cannot function.	Downgrade Neo4j and then upgrade this component before upgrading Neo4j again.
UNSUPPORTED_FUTURE	The sub-graph is unsupported because it is a newer version; this component cannot function.	Upgrade Neo4j.

7.8. Upgrade 4.0 to 4.1 `MINOR` version

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

It covers the following topics:

- Upgrading Neo4j 4.0 to 4.1 `MINOR` version.
  - Upgrade checklist
  - Upgrade a single instance
  - Upgrade a Causal Cluster
  - Troubleshooting

7.8.1. Upgrade checklist

This chapter describes how to prepare to upgrade Neo4j from an earlier minor or patch version.

Before you start preparing for the upgrade, it is very important to read the information in Understanding upgrades and migration and Supported upgrade and migration paths.

Follow the checklist to prepare for upgrading your Neo4j deployment:

- ☐ Complete all prerequisites for the upgrade.
- ☐ Back up your current deployment to avoid losing data in case of failure.
- ☐ Download the new version of Neo4j. Make sure the upgrade path is supported.
- ☐ Prepare a new neo4j.conf file to be used by the new installation.
- ☐ Perform a test upgrade as per your Neo4j version and deployment type (Single instance or Causal Cluster).
- ☐ Monitor the logs.

☐ Perform the upgrade.

ℹ	If you are upgrading a Causal Cluster, complete the checklist for each cluster member.

Prerequisites

1. Verify that you have installed Java 11.
2. Review the improvements and fixes that have been carried out in the version that you want to upgrade to.
   - Neo4j 4.1 Change log
   - Release Notes
   - Changes to configuration settings in Neo4j 4
   - Changes to procedures in Neo4j 4

Back up your current deployment

Depending on your particular deployment and environment, it is important to design an appropriate backup and restore strategy. For more information about what to consider when deciding on your backup and restore strategy, what needs to be backed up, and the different backup modes and options, see Operations Manual → Backup and restore planning.

Prepare a new *neo4j.conf* file to be used by the new installation

Prepare a *neo4j.conf* file to be used by the new installation if you have any custom values configured in your old *neo4j.conf* file. Look for:

- Any non-default settings from your old installation.
- Any custom values of the settings `dbms.directories.*` and `dbms.default_database`.
- (In cluster installations) Cluster-specific configuration settings, which might be different for the different cluster members.

Perform a test upgrade

Based on the findings in this chapter, allocate a staging test environment for the upgrade and do a test upgrade. The test upgrade will give you valuable information about the time required for the production upgrade. Follow the steps as per your Neo4j version and deployment type.

Monitor the logs

The neo4j.log file contains information on how many steps the upgrade will involve and how far it has progressed. It is a good idea to monitor this log.

7.8.2. Upgrade a single instance to 4.1

This section describes how to upgrade a single Neo4j instance.

Prerequisites

1. Ensure that you have completed all tasks on the Upgrade checklist.

Prepare for the upgrade

1. Install the *Neo4j* version that you want to upgrade to. For more information on how to install the distribution that you are using, see Operations Manual → Installation.
2. Replace the *neo4j.conf* file with the one that you have prepared in section Prepare a new *neo4j.conf* file to be used by the new installation.
3. Set `dbms.allow_upgrade=true` to allow automatic store upgrade.
4. Set `dbms.mode=SINGLE`. This enables the automatic upgrade of the `system` database schema, as the setting `dbms.allow_single_automatic_upgrade` is `true` by default when `dbms.mode=SINGLE`.
5. Copy all the files used for encryption, such as private key, public certificate, and the contents of the trusted and revoked directories (located in *<neo4j-home>/certificates/*).
6. Restore each of your databases and transactions in the new installation, including the `system` database, by either using `neo4j-admin restore` (online) or `neo4j-admin load` (offline), depending on your backup approach. If you are running a Debian/RPM distribution, you can skip this step.

   If your old installation has modified configurations starting with `dbms.directories.*` or the setting `dbms.default_database`, verify that the new `neo4j.conf` file is configured properly to find these directories.

7. If you are using custom plugins, make sure they are updated and compatible with the new version, and place them in the */plugins* directory.

Upgrade your instance

1. Start the instance by running the following command from `<neo4j-home>`:

   ```
   bin/neo4j start
   ```

   The upgrade takes place during startup.

2. Monitor the neo4j.log file for information on how many steps the upgrade involves and how far it has progressed.

Post-upgrade tasks

1. When the upgrade finishes, open the *neo4j.conf* file and set `dbms.allow_upgrade=false`. If, for some reason, you have forgotten to enable the automatic upgrade of the `system` database schema, use the following commands to manually upgrade it:
   - `CALL dbms.upgradeStatus()` to determine whether an upgrade is necessary or not.
   - `CALL dbms.upgrade()` on the `system` database to perform the upgrade of the system schema. For

more details, see Operations Manual 4.1 → Procedures.

2. Restart the instance by running the following command from <neo4j-home>:

```
bin/neo4j restart
```

3. It is good practice to make a full backup immediately after the upgrade.

7.8.3. Upgrade a Causal Cluster to 4.1

This section describes how to upgrade a Neo4j Causal Cluster.

You can upgrade your existing Neo4j Causal Cluster by either performing a rolling upgrade, or by upgrading it offline.

ℹ	The prerequisites and the upgrade steps must be completed for each cluster member.

Offline upgrade

This variant is suitable for cases where a rolling upgrade is not possible.

ℹ	It is recommended to perform a test upgrade on a production-like environment to get information on the duration of the downtime.

Prerequisites

Ensure that you have completed all tasks on the Upgrade checklist.

Prepare for the upgrade

1. Shut down all the cluster members (Cores and Read Replicas).
2. Perform `neo4j-admin unbind` on each cluster member to remove cluster state data.
3. Install the Neo4j version that you want to upgrade to on each instance. For more information on how to install the distribution that you are using, see Operations Manual 4.1 → Installation.
4. Replace the *neo4j.conf* file with the one that you have prepared for each instance in section Prepare a new neo4j.conf file to be used by the new installation.
5. Copy all the files used for encryption, such as private key, public certificate, and the contents of the trusted and revoked directories (located in *<neo4j-home>/certificates/*).
6. Restore each of your databases and transactions in the new installation, including the `system` database, by either using `neo4j-admin restore` (online) or `neo4j-admin load` (offline), depending on your backup approach. If you are running a Debian/RPM distribution, you can skip this step.

ℹ	If your old installation has modified configurations starting with `dbms.directories.*` or the setting `dbms.default_database`, verify that the new `neo4j.conf` file is configured properly to find these directories.

7. If you are using custom plugins, make sure they are updated and compatible with the new version, and place them in the */plugins* directory.

Upgrade your cluster

*On **one** of the Cores*

1. Open the *neo4j.conf* file of the new installation and configure the following settings:
   - Uncomment `dbms.allow_upgrade=true` to allow automatic store upgrade.
   - Set `dbms.mode=SINGLE`. This enables the automatic upgrade of the `system` database schema, as the setting `dbms.allow_single_automatic_upgrade` is `true` by default when `dbms.mode=SINGLE`.
2. Start Neo4j by running the following command from `<neo4j-home>`:

   ```
   bin/neo4j start
   ```

   The upgrade takes place during startup.
3. Monitor the neo4j.log file for information on how many steps the upgrade will involve and how far it has progressed.
4. When the upgrade finishes, stop the server.
5. Open the *neo4j.conf* file and configure the following settings:
   - Set `dbms.allow_upgrade=false`.
   - Set `dbms.mode=CORE` to re-enable Causal Clustering in the configuration.
6. Use `neo4j-admin dump` to make a dump of each of your databases and transactions, including the `system` database.
7. Do **not** yet start the server.

On each of the other Cores

1. Copy the database dumps you created on the first Core server to each of the other Cores.
2. Use `neo4j-admin load --from=<archive-path> --database=<database> --force` to replace each of your databases, including the `system` database, with the ones you upgraded on the first Core server.
3. Start each of the core servers, including the first one, and verify that they join in a cluster.

For each Read Replica

Start the Read Replica and wait for it to catch up with the rest of the cluster members.

(Optional) Whilst an empty read replica will eventually get a full copy of all data from the other members of your cluster, this catching up may take some time. In order to speed up the process, you can load the data first by using `neo4j-admin load --from=<archive-path> --database=<database> --force` to replace each of your databases with the upgraded ones, including the `system` database.

Verify that the Read Replicas join the cluster.

Post-upgrade

It is recommended to perform a *full backup*, using an empty target directory.

Rolling upgrade

Rolling upgrade is a zero-downtime method for upgrading a Causal Cluster. You upgrade one member at a time, while the rest of the members are running. However, if during a rolling upgrade the cluster loses quorum and cannot be recovered, then downtime may be required to do a disaster recovery.

Recommendations

- The critical point during the upgrade is knowing when it is safe to switch off the original member. It is highly recommended to monitor the status endpoint before each removal, in order to decide which member to switch off and when it is safe to do so.
- To reduce the risk of failure during a rolling upgrade, make sure the cluster is not under any heavy load during the upgrade. If possible, the safest would be to disable writes entirely.
- There should be no changes to database administration during a rolling upgrade. For more information, see Operations Manual 4.1 → Manage databases.

Rolling upgrade for a fixed number of servers

This variant is suitable for deployments where there is a fixed number of servers and they have to be updated in-place.

When performing a rolling upgrade for a fixed number of servers, it is not possible to increase the cluster size. Therefore, the cluster fault tolerance level will be reduced while replacing the members.

Prerequisites

1. Ensure that you have completed all tasks on the Upgrade checklist.
2. Verify that **all databases are online** by running `SHOW DATABASES` in Cypher Shell or Neo4j Browser. Offline databases can be started using `START DATABASE [database-name]`.

All databases must be started before you start a rolling upgrade. If you have to keep a database inaccessible during the rolling upgrade, you can disable access to it by using the following command:

```
DENY ACCESS ON DATABASE [database-name] TO PUBLIC
```

You must never run `DENY ACCESS ON DATABASE system TO PUBLIC` or `DENY ACCESS ON DATABASE * TO PUBLIC` because you will lock yourself out of the `system` database. If you do lock yourself out, follow the disable authentication steps in the Operations Manual to recover and prevent outside access to the instance or cluster.

When upgrading from Neo4j 4.0.x, you have to disable access to each role that have access to that particular database, as the `PUBLIC` will not yet exist:

```
DENY ACCESS ON DATABASE [database-name] TO [role1],[role2]
```

All available roles can be queried with `SHOW ROLES`.

3. Ensure that the databases cannot be stopped, created, or dropped during the rolling upgrade by using the following command:

```
DENY STOP ON DATABASE * TO PUBLIC
DENY DATABASE MANAGEMENT ON DBMS TO PUBLIC
```

When upgrading from Neo4j 4.0.x, you can only disable the ability to stop databases.

```
DENY STOP ON DATABASE * TO admin
```

This must be done for the `admin` role and all other roles that have the privilege to stop databases. For more information about listing privileges, see Cypher Manual 4.1 → Graph and sub-graph access control.

Upgrade the cluster

You upgrade **one cluster member at a time**, while the rest of the members are running.

If during a rolling upgrade the cluster loses quorum and cannot be recovered, then downtime may be required to do a disaster recovery.

For each cluster member

1. (Recommended) Use the process described in the status endpoint to evaluate whether it is safe to remove the old instance.
2. Shut down the instance.

3. Install the *Neo4j* version that you want to upgrade to. For more information on how to install the distribution that you are using, see Operations Manual 4.1 → Installation.
4. Replace the *neo4j.conf* file with the one that you have prepared for this instance in Prepare a new *neo4j.conf* file to be used by the new installation.
5. Start the new instance and wait for it to catch up with the rest of the cluster members.
6. Verify that the new instance has successfully joined the cluster and caught up with the rest of the members, by using the status endpoint.

> Because Read Replicas are not part of the cluster consensus group, their replacement during an upgrade does not affect the cluster availability and fault tolerance level. However, it is still recommended to incrementally add Read Replicas for a structured and maintainable upgrade process.

Upgrade the `system` database

In 4.x versions, Neo4j uses a shared `system` database, which includes complex information, such as the security configuration for users, roles, and their privileges. The structure of the graph contained in the `system` database changes with each new version of Neo4j as the capabilities of the DBMS grow. Therefore, each time a Neo4j deployment is upgraded, the contents of the `system` database, or schema, must be transformed as well. When performing an offline upgrade of a single deployment or a Causal cluster, these changes happen automatically, as a consequence of configuring `dbms.mode=SINGLE` (See Prepare for the upgrade and Upgrade your cluster). However, when performing a rolling upgrade, you never start instances with the configuration value `dbms.mode=SINGLE`, i.e., updating the `system` database automatically is not possible.

Compatibility and synchronization

With a causal cluster of many instances, while upgrading each instance in turn, there is a period during which the cluster is composed of some old and some new instances. A single `system` database is consistently replicated across the entire cluster. As a result, it is not possible to have its schema structured according to the needs of the new Neo4j version on some instances and the old version on others.

When the `system` database is not up-to-date with the Neo4j version of a given instance, that instance runs in *compatibility mode*. This means that capabilities common to both Neo4j versions will continue to work, but the features that require the new schema, will be disabled. For example, if you attempt to grant a new privilege not supported in the old schema, you will receive an error and the grant will fail. Therefore, when the rolling upgrade finishes, you must manually upgrade the `system` database schema in order to access all new features.

> If the `system` database's schema is too old to allow *compatibility mode*, the server will not start. For more information, see Troubleshooting.

Manually trigger an upgrade of the `system` database

1. On one of the cluster members, call the procedure `dbms.upgradeStatus()` to determine whether an upgrade is necessary:

   ```
   CALL dbms.upgradeStatus();
   ```

   ```
   +----------------------------------------------------------------------------------------------------------
   ---------------------+
   | status              | description                                                                  |
   resolution            |
   +----------------------------------------------------------------------------------------------------------
   ---------------------+
   | "REQUIRES_UPGRADE" | "The sub-graph is supported, but is an older version and requires upgrade" |
   "CALL dbms.upgrade()" |
   +----------------------------------------------------------------------------------------------------------
   ---------------------+
   ```

   For the full list of possible status values, see Status codes for `dbms.upgradeStatus`.

2. On one of the cluster members, perform the upgrade by calling the procedure `dbms.upgrade()` against the `system` database:

   ```
   CALL dbms.upgrade();
   ```

   ```
   +----------------------------+
   | status     | upgradeResult |
   +----------------------------+
   | "CURRENT"  | "Success"     |
   +----------------------------+
   ```

   Since Neo4j uses a shared `system` database, the upgraded `system` database will replicate across the entire cluster. If the upgrade fails for some reason, the status will not change, and the `upgradeResult` field will describe which components have failed to upgrade.

Post-upgrade steps

The following steps must be performed after a rolling upgrade.

1. Restore the privilege of the `PUBLIC` role to stop databases:

   ```
   REVOKE DENY STOP ON DATABASE * FROM PUBLIC
   ```

2. Restore the privilege of the `PUBLIC` role to create and drop databases:

   ```
   REVOKE DENY DATABASE MANAGEMENT ON DBMS FROM PUBLIC
   ```

3. (Optional) If you have started offline databases and denied some access rights during the preparation phase for a rolling upgrade, you should also restore them to the original state:

a. Stop each of the databases by running the following command:

```
STOP DATABASE [database-name]
```

b. Re-enable access to the databases by running the following command:

```
REVOKE DENY ACCESS ON DATABASE [database-name] FROM [role1],[role2]
```

4. (Recommended) Perform a *full backup*, using an empty target directory.

Rolling upgrade for cloud infrastructure

This variant is suitable for deployments that use replaceable cloud or container resources. It follows the same steps as for the fixed number of servers, but **you can add the new members before you shut down the old ones**, thus preserving the cluster fault tolerance level. Because Read Replicas are not part of the cluster consensus group, their replacement during the upgrade will not affect the cluster availability and fault tolerance level. However, it is still recommended to incrementally add Read Replicas for a structured and maintainable upgrade process.

7.8.4. Troubleshooting

Troubleshooting information that can help you diagnose and correct a problem.

You can review the Operations Manual → Neo4j log files to monitor the upgrade process and verify that no problems occur as it progresses. If a problem occurs, the following known troubleshooting information can help you diagnose and correct the problem.

Solutions to common problems

Calling `dbms.upgrade` results in a failure

The `upgradeResult` field describes which components have failed to upgrade.

Solution

To troubleshoot the root cause there are two things to try:

- Call the more verbose version of the command — `dbms.upgradeDetails`, which returns a message with the underlying exception thrown. This might not be the root cause, but can help narrow it down.
- Look for descriptive warnings and errors in the *security.log* and the *debug.log*. Stack traces should help track down the root cause of the problem.

Trying to upgrade between incompatible versions, for example migrating from Neo4j 3.5 to Neo4j 4.1

When the detected version of the `system` database is too old, the status code is one of the following:

- UNSUPPORTED_BUT_CAN_UPGRADE - the server shuts down.

  Solution

  Configure dbms.mode=SINGLE. This enables the automatic upgrade of the system database schema as the setting dbms.allow_single_automatic_upgrade is true by default when dbms.mode=SINGLE.

- UNSUPPORTED - the server shuts down.

  Solution

  No direct upgrade is possible, upgrade first to a supported intermediate version.

Calling dbms.upgrade when not all instances have been upgraded to the new version of Neo4j

- Calling dbms.upgradeStatus on any instances still running the older version returns UNSUPPORTED_FUTURE.
- These older instances are no longer able to authorize users.
- All queries against these instances result in an error.

  Solution

  Shut down, upgrade, and restart the instance in order to regain service.

Status codes for dbms.upgradeStatus

System component status

Status	Description	Resolution
UNINITIALIZED	No sub-graph detected for this component.	Requires initialization.
CURRENT	The sub-graph is already at the current version.	No action required.
REQUIRES_UPGRADE	The sub-graph is supported but is an older version and requires upgrade.	Run CALL dbms.upgrade().
UNSUPPORTED_BUT_CAN_UPGRADE	The sub-graph is unsupported; this component cannot function.	Restart Neo4j in a single-instance mode to upgrade this component at startup.
UNSUPPORTED	The sub-graph is unsupported because it is too old; this component cannot function.	Downgrade Neo4j and then upgrade this component before upgrading Neo4j again.
UNSUPPORTED_FUTURE	The sub-graph is unsupported because it is a newer version; this component cannot function.	Upgrade Neo4j.

7.9. Migrate 3.5

This topic explains the migration steps for a Neo4j 3.5 DBMS to a Neo4j 4.0 DBMS. A migration to the next MAJOR version requires careful planning and considerations.

It covers the following topics:

- Migration checklist
- Migrate 3.5 to 4.0
  - Migrate a single instance (offline)
  - Migrate a cluster (offline)
- Migrate 3.5 to 4.x
  - Migrate a single instance (offline)
  - Migrate a cluster (offline)

💡	The table Documentation for completing your upgrade/migration provides further details on what chapters to follow depending on your starting and target Neo4j versions.

7.9.1. Breaking changes between Neo4j 3.5 and Neo4j 4.x

This chapter describes the breaking changes between Neo4j 3.5 and Neo4j 4.x.

It covers changes to:

- Configuration settings
- Neo4j SSL framework
- Authentication and authorization
- Databases
- Causal Cluster
- Cypher syntax
- Procedures
- Metrics
- Logs
- Tools
- Backups
- JMX
- Neo4j API
- REST API
- HTTP API endpoints
- External dependencies

Configuration settings

This section lists breaking changes for Neo4j configuration settings.

Previous name	Change	New name (if applicable)
dbms.active_database	Renamed	dbms.default_database
dbms.connectors.default_listen_address	Renamed	dbms.default_listen_address
dbms.connectors.default_advertised_address	Renamed	dbms.default_advertised_address
dbms.backup.address	Renamed	dbms.backup.listen_address
dbms.logs.query.enabled	This is no longer a boolean setting. Valid values are: OFF, INFO or VERBOSE.	
causal_clustering.cluster_routing_ttl	Renamed	dbms.routing_ttl
causal_clustering.middleware_logging.level	Renamed Valid values are: DEBUG,INFO, WARN, ERROR or NONE	causal_clustering.middleware.logging.level
causal_clustering.disable_middleware_logging	Discontinued Set causal_clustering.middleware.logging.level=OFF to disable middleware logging.	
metrics.neo4j.logrotation.enabled	Renamed	metrics.neo4j.logs.enabled
metrics.enabled	This setting no longer changes the default values of the individual metrics. Instead it turns off the whole metrics module.	

Neo4j SSL framework

This section describes breaking changes for the Neo4j SSL framework.

v3.x	Change	v4.x
The configuration settings dbms.security.property_level.enabled and dbms.security.property_level.blacklist are used to disallow properties.	Discontinued	The Cypher DENY command replaces the blocking functionality. Note that the DENY command must be applied while Neo4j is running. For details, see Cypher Manual 4.0 → Graph and sub-graph access control.
dbms.security.auth_provider	Discontinued	This setting is replaced by two new settings: dbms.security.authentication_providers and dbms.security.authorization_providers.

v3.x	Change	v4.x
dbms.connector.https.enabled is set to true by default.	This setting is no longer true by default.	To enable Neo4j to listen for incoming connections on the HTTPS port, you have to configure this setting to true.
The different communication channels are secured independently from each other, using the following configuration settings: bolt.ssl_policy=<policy name> https.ssl_policy=<policy name> causal_clustering.ssl_policy=<policy name> dbms.backup.ssl_policy=<policy name>	Discontinued	These settings have been replaced by the setting dbms.ssl.policy.<scope>.enabled=true , where <scope> substitutes the communication channel (bolt, https, cluster, and backup).
SSL support for Bolt and HTTPS using the legacy SSL system.	Deprecated	It is recommended to use the standard SSL configuration.
The dbms.directories.certificates setting is used to explicitly configure the directory that stores the private key and certificate files.	Discontinued	It is recommended to use the standard SSL configuration.
dbms.ssl.policy.*.allow_key_generation	Discontinued	Neo4j no longer automatically generates a self-signed certificate.

For further details on the SSL framework changes, see Operations Manual → SSL framework.

Authentication and authorization

This section describes breaking changes and deprecations for the Neo4j authentication and authorization.

Removal of flat files for authentication and authorization

Neo4j 3.x manages authentication and authorization in flat files.

Neo4j 4.0 introduces a complex security model stored in the system graph, maintained in a special database called the system database. The contents of both the *auth* and *roles* files are automatically migrated to the system database. If you are bringing multiple databases together from multiple Neo4j 3.5 deployments or upgrading a cluster, you must manually merge the *auth* and *role* files before the migration.

The neo4j-admin commands set-initial-password and set-default-admin continue to work in version 4.0 and write to the same files as in 3.5. Any content in these files is considered on the first start of Neo4j after installing the new version.

The command set-initial-password is applied only if the default user neo4j with the default password is the only user present, while set-default-admin will only be used when no roles are present. For more information, see Operation Manual 4.0 → Set an initial password.

Role-based access control

Neo4j 3.1 introduced the concept of role-based access control. It was possible to create users and assign them to roles to control whether they could read, write, and administer the database. In Neo4j 4.0, this model is enhanced significantly with the addition of privileges, which are the underlying access-control rules by which the users' rights are defined.

It is still possible to have different security configurations per database after the migration, but this needs to be managed through the granting of privileges and roles specific to databases after the migration. The built-in roles from 3.5 still exist but will apply to all databases after the migration unless explicitly modified using the new security administration commands. The ability to manage database-specific roles and privileges is described in more detail in Cypher Manual 4.0 → Administration.

It is no longer possible to have different security privileges on different instances of a cluster. The entire cluster shares the privileges configured in the system database using Cypher administration commands. In practice, this means that users have the same privileges regardless of which server in a cluster they access.

Deprecated and removed security procedures

The built-in security procedures for user management dbms.security in Neo4j 3.x are deprecated in 4.x. If you still want to use them, they must be run against the system database and cannot be followed by YIELD.

There are two options for rewriting your code and routines for managing authentication and authorization.

- (Recommended) Rewrite the procedures to the corresponding Cypher administration commands, using the conversion table. All administrative commands must be executed against the system database. When connected to the DBMS using the Neo4j Drivers (Bolt Protocol), administrative commands are automatically routed to the system database (Neo4j 4.1+).
- Run the procedures against the system database and replace any YIELD parts by post-processing on the application side.

The procedure dbms.security.changePassword(password, requirePasswordChange) has been entirely removed since the corresponding Cypher administration command also requires the old password, and thus is more secure.

Procedure to Cypher administrative command conversion

Procedure	Administration command
dbms.security.createUser	CREATE USER
dbms.security.deleteUser	DROP USER

Procedure	Administration command
`dbms.security.changePassword`	`ALTER CURRENT USER SET PASSWORD`
`dbms.security.listUsers`	`SHOW USERS`
`dbms.security.changeUserPassword`	`ALTER USER`
`dbms.security.suspendUser`	`ALTER USER`
`dbms.security.activateUser`	`ALTER USER`
`dbms.security.addRoleToUser`	`GRANT ROLE TO USER`
`dbms.security.removeRoleFromUser`	`REVOKE ROLE FROM USER`
`dbms.security.listRoles`	`SHOW ROLES`
`dbms.security.listRolesForUser`	`SHOW USERS`
`dbms.security.listUsersForRole`	`SHOW ROLES WITH USERS`
`dbms.security.createRole`	`CREATE ROLE`
`dbms.security.deleteRole`	`DROP ROLE`

For more info about the administration commands, see Cypher Manual 4.0 → User and role management.

Databases

This section describes breaking changes for the Neo4j DBMS.

Multiple databases

From version 4.0 onwards, Neo4j supports the management of multiple databases within the same DBMS. The metadata for these databases, including the associated security model, is maintained in a special database called the `system` database.

A default installation of Neo4j contains two databases:

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

For more details, see Operations Manual 4.0→ Manage databases and Cypher Manual 4.0 → Administration.

Database naming rules

With the introduction of multiple databases, the rules for naming a database have changed. For example, it is no longer possible to use an underscore in a database name. For a full list of naming rules, please see Operations Manual 4.0 → Administrative commands.

Embedded layout

To support multiple databases in an embedded layout, the store files, transaction files, and log files no longer reside in the base directory. Instead, files are separated per database into separate directories. For more information, see Operations Manual → File locations.

Causal Cluster

This section describes breaking changes for the Neo4j cluster discovery and REST API endpoints.

Cluster discovery

Cluster discovery is implemented on top of Akka (https://akka.io/), instead of Hazelcast (https://hazelcast.com/). A few minor changes have been made as part of this transition:

- The `discovery_advertised_address` hostname and port must exactly match those configured for the discovery of other members.

  When `discovery_type=LIST` is used, the list of addresses in `initial_discovery_members` must match the respective advertised addresses of each server.

  When using any other discovery types (DNS, SRV, K8S), the configuration in the external service must match.

  > By default, your `discovery_advertised_address` is a combination of the default port assigned to that configuration and the hostname assigned to `default_advertised_address`.
  >
  > For more information on cluster discovery, see Operations Manual 4.0 → Discovery

- Connections are now opened from Cores to Read Replicas, in addition to vice versa. Therefore, the advertised discovery port must be **open** on Read Replicas.

Cluster REST endpoints

The REST endpoints are per database instead of per instance:

Old endpoint	New endpoint
`/db/manage/server/causalclustering/writable`	`/db/<databasename>/cluster/writable`
`/db/manage/server/causalclustering/read-only`	`/db/<databasename>/cluster/read-only`
`/db/manage/server/causalclustering/available`	`/db/<databasename>/cluster/available`
`/db/manage/server/causalclustering/status`	`/db/<databasename>/cluster/status`

Cypher syntax

This section lists breaking changes for the Cypher syntax.

All changes in the Cypher language syntax are detailed in Cypher Manual → Removals, deprecations, additions and extensions. Thoroughly review this section in the version you are moving to and make the necessary changes in your code.

From Neo4j 4.0 onwards, the parameter syntax `{parameter}` is replaced by the syntax `$parameter`.

Procedures

This section lists the refactored and new procedures.

v3.x	v4.x	Comment
`db.awaitIndex (indexId :: INTEGER?, timeOutSeconds = 300 :: INTEGER?) :: VOID`	`db.awaitIndex (indexName :: STRING?, timeOutSeconds = 300 :: INTEGER?) :: VOID`	Indexes are identified by name instead of ID.
`dbms.cluster.overview() :: (id :: STRING?, addresses :: LIST? OF STRING?, role :: STRING?, groups :: LIST? OF STRING?, database :: STRING?)`	`dbms.cluster.overview() :: (id :: STRING?, addresses :: LIST? OF STRING?, databases :: MAP?, groups :: LIST? OF STRING?)`	Show roles for all databases.
`dbms.cluster.role() :: (role :: STRING?)`	`dbms.cluster.role (database :: STRING?) :: (role :: STRING?)`	Take `database` name as a parameter.
`dbms.cluster.routing.getRoutingTable (context :: MAP?) :: (ttl :: INTEGER?, servers :: LIST? OF MAP?)`	`dbms.cluster.routing.getRoutingTable (context :: MAP?, database = null :: STRING?) :: (ttl :: INTEGER?, servers :: LIST? OF MAP?)`	Take `database` name as a parameter.
`db.createIndex (index :: STRING?, providerName :: STRING?) :: (index :: STRING?, providerName :: STRING?, status :: STRING?)`	`db.createIndex (indexName :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, config = {} :: MAP?) :: (name :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, status :: STRING?)`	• Take `labels` and `properties` as separate lists instead of the index pattern `":Label(prop)"` as an argument. • Those are also yielded as a result. • Needs `indexName`. • (Optional) Take index settings as a map.

v3.x	v4.x	Comment
`db.createUniquePropertyConstraint (index :: STRING?, providerName :: STRING?) :: (index :: STRING?, providerName :: STRING?, status :: STRING?)`	`db.createUniquePropertyConstraint (constraintName :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, config = {} :: MAP?) :: (name :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, status :: STRING?)`	• Take `labels` and `properties` as separate lists instead of the index pattern `":Label(prop)"` as an argument. • Those are also yielded as a result. • Needs `constraintName`. • (Optional) Take index settings as a map.
`db.createNodeKey (index :: STRING?, providerName :: STRING?) :: (index :: STRING?, providerName :: STRING?, status :: STRING?)`	`db.createNodeKey (constraintName :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, config = {} :: MAP?) :: (name :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, status :: STRING?)`	• Take `labels` and `properties` as separate lists instead of the index pattern `":Label(prop)"` as an argument. • Those are also yielded as a result. • Needs `constraintName`. • (Optional) Take index settings as a map.
`db.indexes() :: (description :: STRING?, indexName :: STRING?, tokenNames :: LIST? OF STRING?, properties :: LIST? OF STRING?, state :: STRING?, type :: STRING?, progress :: FLOAT?, provider :: MAP?, id :: INTEGER?, failureMessage :: STRING?)`	`db.indexes() :: (id :: INTEGER?, name :: STRING?, state :: STRING?, populationPercent :: FLOAT?, uniqueness :: STRING?, type :: STRING?, entityType :: STRING?, labelsOrTypes :: LIST? OF STRING?, properties :: LIST? OF STRING?, provider :: STRING?)`	• Rename `indexName` to `name`. • Rename `tokenNames` to `labelsOrTypes`. • Rename `progress` to `populationPercent`. • The field `type`, which described entity type (node or relationship), uniqueness, and index type, is split up into `type`, `uniqueness`, and `entityType`. • The field `provider` is a string instead of a map. • `description` is removed in favor of `db.schemaStatements`. • `failureMessage` is moved to procedure `db.indexDetails`.

v3.x	v4.x	Comment
`db.resampleIndex (index :: STRING?) :: VOID`	`db.resampleIndex (indexName :: STRING?) :: VOID`	Indexes are uniquely identified by name instead of index pattern `":Label(prop)"`.
	`db.indexDetails (indexName :: STRING?) :: (id :: INTEGER?, name :: STRING?, state :: STRING?, populationPercent :: FLOAT?, uniqueness :: STRING?, type :: STRING?, entityType :: STRING?, labelsOrTypes :: LIST? OF STRING?, properties :: LIST? OF STRING?, provider :: STRING?, indexConfig :: MAP?, failureMessage :: STRING?)`	Get all the information for the specified index from `db.indexes`, `indexConfig`, and `failureMessage`.
	`db.schemaStatements () :: (name :: STRING?, type :: STRING?, createStatement :: STRING?, dropStatement :: STRING?)`	Get all create and drop statements needed to replicate the schema rules (indexes and constraints) for this database.
	`db.ping()`	The client-side tooling uses this procedure to test the connection to a database. The procedure is available in all databases and always returns `true`.

Metrics

This section describes breaking changes for Neo4j metrics.

In 4.x, there are two types of metrics: global metrics and database-local metrics. The metric naming is different in 4.x compared to 3.x. For details about available metrics and the new naming patterns, please refer to Operations Manual → Metrics.

Please note that in 4.2 the metrics that are enabled by default have been changed.

Any specific metrics that you want to be enabled **must** be specified in the `metrics.filter`.

Additionally in the 4.2 release, metrics are no longer exposed via JMX by default. These can be enabled by adding `metrics.jmx.enabled=true` to neo4j.conf.

For more information, see Operations Manual → Enable metrics logging.

Logs

This section describes breaking changes for Neo4j logs.

General changes

From the 4.0 version onwards, Neo4j logs have the name of the database to which the log line pertains, printed before the regular text. For example, `[neo4j]` or `[system]`.

Example 1. Some log lines for the `system` database

```
2019-12-02 22:27:41.820+0000 INFO [o.n.k.d.Database] [system] No check point found in transaction log
2019-12-02 22:27:41.820+0000 INFO [o.n.k.d.Database] [system] Recovery required from position
LogPosition{logVersion=0, byteOffset=64}
2019-12-02 22:27:41.820+0000 INFO [o.n.k.r.Recovery] [system]   10% completed
2019-12-02 22:27:41.820+0000 INFO [o.n.k.r.Recovery] [system]   20% completed
2019-12-02 22:27:41.820+0000 INFO [o.n.k.r.Recovery] [system]   30% completed
...
```

Other log lines might relate to the DBMS as a whole or be logged by a component that lives on a higher level but still operates on a particular database. For example:

Example 2. Some log lines from `CoreDatabaseManager` starting the Neo4j database

```
2019-12-02 22:27:41.964+0000 INFO [c.n.c.c.CoreDatabaseManager] Creating 'neo4j' database.
2019-12-02 22:27:41.967+0000 INFO [c.n.c.c.CoreDatabaseManager] Starting 'neo4j' database.
...
```

Changes to log configurations per Neo4j version

Configuration setting	Neo4j 4.0	Neo4j 4.1	Neo4j 4.2	Neo4j 4.3
`causal_clustering.middleware.logging.level`	New			
`dbms.directories.transaction.logs.root`	New			
`dbms.tx_log.preallocate`	New			
`fabric.driver.logging.level`	New			
`metrics.neo4j.logs.enabled`	New			
`causal_clustering.middleware_logging.level`	Removed			
`dbms.logs.timezone`	Removed			
`metrics.neo4j.logrotation.enabled`	Removed			
`causal_clustering.log_shipping_retry_timeout`		New		
`dbms.logs.query.early_raw_logging_enabled`		New		
`dbms.logs.query.parameter_full_entities`		New		
`dbms.routing.driver.logging.level`		New		
`dbms.logs.security.rotation.delay`			Deprecated	

Configuration setting	Neo4j 4.0	Neo4j 4.1	Neo4j 4.2	Neo4j 4.3
`dbms.logs.user.rotation.delay`			Deprecated	
`dbms.logs.debug.rotation.delay`			Deprecated	
`dbms.logs.debug.format`				New
`dbms.logs.default_format`				New
`dbms.logs.query.format`				New
`dbms.logs.query.obfuscate_literals`				New
`dbms.logs.query.plan_description_enabled`				New
`dbms.logs.query.transaction.enabled`				New
`dbms.logs.query.transaction_id.enabled`				New
`dbms.logs.query.transaction.threshold`				New
`dbms.logs.security.format`				New
`dbms.logs.user.format`				New

Tools

This section describes breaking changes for the `neo4j-admin` command.

From Neo4j 4.0 onwards, there are some changes for the `neo4j-admin` command:

- `--database` option New

  Specify a database for a particular operation. If not specified, the default `neo4j` database is used.

- `--nodes` option Changed

  The syntax has changed to:
  `neo4j-admin import --nodes=[<label>[:<label>]…=]<files>…`

- `--relationships` option Changed

  The syntax has changed to:
  `neo4j-admin import --relationships=[<type>=]<files>…]…`

- When importing data using `neo4j-admin import`, you must create the database (using `CREATE DATABASE` against the `system` database) after the import operation finishes. Otherwise, you cannot access the imported data.

Backups

This section describes breaking changes and deprecations for Neo4j backups.

From Neo4j 4.0, every database is backed up individually. Therefore, it is very important to plan your backup strategy for each of them. For more detailed information on designing an appropriate backup strategy for your setup, see Operations Manual 4.0 → Backup and restore.

From the 4.0 version onwards, a default Neo4j installation has two databases:

- `system`, containing metadata on the DBMS and security configuration.
- `neo4j`, the default database.

Use the `--database` option of the `neo4j-admin backup` command to specify the database to back up. For more information, see Operations Manual → Back up an online database.

The `--name` parameter has been removed. It was previously used to specify the last part of the path when using `--backup-dir`. The last part of the path is now inferred from the `--database` parameter, which specifies the database name on the server.

If you previously used `--name` for customizing the backup path, for example, by including a timestamp, you can now use `--backup-dir` instead.

JMX

This section describes breaking changes for Neo4j JMX endpoints.

In 3.x, Neo4j exposed several JMX MBeans in order to provide some monitoring information in addition to the metrics exposed by Neo4j. In some instances, the provided data was incomplete or incorrect, and in some cases different beans even provided conflicting information. All of the previous JMX endpoints (`org.neo4j:*`) have been removed and are replaced by a new set of beans (`neo4j.metrics:*`) that expose exactly the same information as the corresponding Neo4j metrics.

JMX MBeans are available only in Enterprise Edition.

Neo4j API

This section describes breaking changes for migrating Neo4j 3.5 API to Neo4j 4.0 API.

JDK 11

Neo4j 4.0 is the first major release that requires JDK 11. Custom extensions and procedures can be compiled now for JDK 11, for example, `-target 11`. It is generally recommended to use the latest available JDK 11 to access all available fixes and performance improvements.

General changes

org.neo4j.graphdb.schema

- Neo4j 3.5 API — org.neo4j.graphdb.schema
- Neo4j 4.0 API — org.neo4j.graphdb.schema

Neo4j 4.0 comes with significant changes in schema and indexes. Most of the related classes have additional possibilities.

Changes include

- All indexes are named. The name of an index can be retrieved using `getName()` call on `IndexDefinition` and `ConstraintDefinition`.
- The definition of an index can be looked up by name using `Schema`.
- Single label and relationship type accessors `getLabel()` and `getRelationshipType()` have been removed from `IndexDefinition`.

Affected classes

- `org.neo4j.graphdb.schema.ConstraintCreator`
- `org.neo4j.graphdb.schema.ConstraintDefinition`
- `org.neo4j.graphdb.schema.IndexCreator`
- `org.neo4j.graphdb.schema.IndexDefinition`
- `org.neo4j.graphdb.schema.Schema`

org.neo4j.graphdb.event

- Neo4j 3.5 API — org.neo4j.graphdb.event
- Neo4j 4.0 API — org.neo4j.graphdb.event

Transaction event listeners have an updated behavior.

Changes include

- As part of the callback, you will always receive the owning `GraphDatabaseService` as one of the parameters.
- The `beforeCommit` listener method has access to an ongoing transaction over the `transaction` call parameter.
- `DatabaseEventListener` is a new type of listener that has been introduced. Since Neo4j now supports multiple databases, you might want to be able to listen to database events from several databases. It can be registered and de-registered in `DatabaseManagementService`.

Affected classes

- `org.neo4j.graphdb.event.TransactionEventListener`
- `org.neo4j.graphdb.event.DatabaseEventContext`
- `org.neo4j.graphdb.event.DatabaseEventListener`

org.neo4j.helpers

💡	• Neo4j 3.5 API — org.neo4j.helpers • Neo4j 4.0 API — org.neo4j.configuration.helpers

Most of the helpers are no longer part of the Neo4j API. The `SocketAddress` helper has minor API changes.

Affected classes

- `org.neo4j.configuration.helpers.SocketAddress`

com.neo4j.backup

💡	• Neo4j 3.5 API — org.neo4j.backup • Neo4j 4.0 API — com.neo4j.backup

The backup facade has been simplified and adapted to a multi-database environment.

Affected classes

- `com.neo4j.backup.OnlineBackup`

org.neo4j.configuration

💡	• Neo4j 3.5 API — org.neo4j.graphdb.config • Neo4j 4.0 API — org.neo4j.configuration

Configuration API has been updated to be typed. It is no longer safe to assume that the configuration is a set of random key-value pairs. All pairs unknown to Neo4j will be rejected. Additionally, some settings have been renamed. For more information, see Configuration settings.

Affected classes

- `org.neo4j.configuration.GraphDatabaseSettings`
- `org.neo4j.graphdb.config.Setting`
- `org.neo4j.configuration.connectors.BoltConnector`

org.neo4j.graphdb.Transaction

💡	• Neo4j 3.5 API — org.neo4j.graphdb.Transaction • Neo4j 4.0 API — org.neo4j.graphdb.Transaction

Transaction API changes are one of the most significant API updates that are part of 4.0.
All of the methods that must be executed in a transaction have been moved from `GraphDatabaseService` to `Transaction`. This means that if you want to create entities or access them, you can find all the methods in `Transaction`. Additionally, starting with 4.0, transactions are no longer thread-bound, which means that

any call to `GraphDatabaseService::beginTx()` creates a new independent transaction, even if it is called from one thread.

Affected classes

- `org.neo4j.graphdb.Transaction`

org.neo4j.graphdb.Entity

- Neo4j 3.5 API — org.neo4j.graphdb.Entity
- Neo4j 4.0 API — org.neo4j.graphdb.Entity

The `PropertyContainer` interface is removed, and all property-related methods are moved to `Entity`. Access to entities is transactional. An entity can be accessed only from the transaction it is created or retrieved.

Affected classes

- `org.neo4j.graphdb.Entity`
- `org.neo4j.graphdb.Node`
- `org.neo4j.graphdb.Relationship`

org.neo4j.graphdb.GraphDatabaseService

- Neo4j 3.5 API — org.neo4j.graphdb.GraphDatabaseService
- Neo4j 4.0 API — org.neo4j.graphdb.GraphDatabaseService

All methods that require transactions are moved to `Transaction`. In addition, a set of `executeTransactionally` methods have been added to provide a convenient way of query executions in a separate transaction.

Affected classes

- `org.neo4j.graphdb.GraphDatabaseService`

org.neo4j.harness and com.neo4j.harness

Support for official testing support classes. From 4.0 onwards, Neo4j provides a set of Junit 4 rules and Junit 5 extensions for community and enterprise users.

Affected classes

- `com.neo4j.harness.junit.extension.EnterpriseNeo4jExtension`
- `com.neo4j.harness.junit.rule.EnterpriseNeo4jRule`
- `org.neo4j.harness.junit.extension.Neo4j`
- `org.neo4j.harness.junit.extension.Neo4jExtension`

* `org.neo4j.harness.junit.extension.Neo4jExtensionBuilder`
* `org.neo4j.harness.junit.rule.Neo4jRule`

org.neo4j.dbms.api

* Neo4j 3.5 API — org.neo4j.dbms.api
* Neo4j 4.0 API — org.neo4j.dbms.api

The top-level Neo4j API has been updated. The main access point for individual databases and for performing any database management operations is called `DatabaseManagementService`. It can be constructed by the Community or Enterprise version of `DatabaseManagementServiceBuilder`.

The following example shows how to construct a new `managementService` and a lookup `GraphDatabaseService` for a database named `neo4j`:

Example 3. Using `DatabaseManagementService`

```
var managementService = new DatabaseManagementServiceBuilder( homeDirectory ).build();
var databaseService = managementService.database( "neo4j" );
```

Affected classes

* `org.neo4j.dbms.api.DatabaseManagementService`
* `org.neo4j.dbms.api.DatabaseManagementServiceBuilder`

Renamed classes

The following classes have been renamed:

Old class name	New class name
`org.neo4j.graphdb.factory.GraphDatabaseSettings.BoltConnector`	`org.neo4j.configuration.connectors.BoltConnector`
`org.neo4j.graphdb.factory.GraphDatabaseSettings.BoltConnector.EncryptionLevel`	`org.neo4j.configuration.connectors.BoltConnector.EncryptionLevel`
`org.neo4j.kernel.configuration.HttpConnector`	`org.neo4j.configuration.connectors.HttpConnector`
`org.neo4j.graphdb.factory.GraphDatabaseSettings`	`org.neo4j.configuration.GraphDatabaseSettings`
`org.neo4j.graphdb.factory.GraphDatabaseSettings.SchemaIndex`	`org.neo4j.configuration.GraphDatabaseSettings.SchemaIndex`
`org.neo4j.backup.OnlineBackup`	`com.neo4j.backup.OnlineBackup`
`org.neo4j.helpers.SocketAddress`	`org.neo4j.configuration.helpers.SocketAddress`
`org.neo4j.graphdb.event.TransactionEventHandler`	`org.neo4j.graphdb.event.TransactionEventListener`
`org.neo4j.graphdb.factory.GraphDatabaseSettings.TransactionStateMemoryAllocation`	`org.neo4j.configuration.GraphDatabaseSettings.TransactionStateMemoryAllocation`
`org.neo4j.graphdb.index.fulltext.AnalyzerProvider`	`org.neo4j.graphdb.schema.AnalyzerProvider`

Old class name	New class name
`org.neo4j.server.security.enterprise.auth.plugin.api.AuthProviderOperations`	`com.neo4j.server.security.enterprise.auth.plugin.api.AuthProviderOperations`
`org.neo4j.server.security.enterprise.auth.plugin.api.AuthToken`	`com.neo4j.server.security.enterprise.auth.plugin.api.AuthToken`
`org.neo4j.server.security.enterprise.auth.plugin.api.AuthenticationException`	`com.neo4j.server.security.enterprise.auth.plugin.api.AuthenticationException`
`org.neo4j.server.security.enterprise.auth.plugin.api.PredefinedRoles`	`com.neo4j.server.security.enterprise.auth.plugin.api.PredefinedRoles`
`org.neo4j.server.security.enterprise.auth.plugin.spi.AuthInfo`	`com.neo4j.server.security.enterprise.auth.plugin.spi.AuthInfo`
`org.neo4j.server.security.enterprise.auth.plugin.spi.AuthPlugin`	`com.neo4j.server.security.enterprise.auth.plugin.spi.AuthPlugin`

Removed classes

The following classes have been removed:

- org.neo4j.backup.BackupExtensionService
- org.neo4j.backup.BackupTool
- org.neo4j.backup.IncrementalBackupNotPossibleException
- org.neo4j.backup.OnlineBackupExtensionFactory.Dependencies
- org.neo4j.backup.OnlineBackupExtensionFactory
- org.neo4j.backup.OnlineBackupKernelExtension.BackupProvider
- org.neo4j.backup.OnlineBackupKernelExtension
- org.neo4j.backup.OnlineBackupSettings
- org.neo4j.backup.TheBackupInterface
- org.neo4j.cypher.export.CypherResultSubGraph
- org.neo4j.cypher.export.DatabaseSubGraph
- org.neo4j.cypher.export.SubGraphExporter
- org.neo4j.cypher.export.SubGraph
- org.neo4j.graphalgo.CommonEvaluators
- org.neo4j.graphalgo.MaxCostEvaluator
- org.neo4j.graphdb.DatabaseShutdownException
- org.neo4j.graphdb.DependencyResolver.Adapter
- org.neo4j.graphdb.DependencyResolver.SelectionStrategy
- org.neo4j.graphdb.DependencyResolver
- org.neo4j.graphdb.DynamicLabel
- org.neo4j.graphdb.DynamicRelationshipType

- org.neo4j.graphdb.InvalidTransactionTypeException
- org.neo4j.graphdb.PathExpanderBuilder
- org.neo4j.graphdb.PathExpanders
- org.neo4j.graphdb.PropertyContainer
- org.neo4j.graphdb.ResourceUtils
- org.neo4j.graphdb.TransactionGuardException
- org.neo4j.graphdb.TransientDatabaseFailureException
- org.neo4j.graphdb.TransientFailureException
- org.neo4j.graphdb.TransientTransactionFailureException
- org.neo4j.graphdb.config.BaseSetting
- org.neo4j.graphdb.config.InvalidSettingException
- org.neo4j.graphdb.config.ScopeAwareSetting
- org.neo4j.graphdb.config.SettingGroup
- org.neo4j.graphdb.config.SettingValidator
- org.neo4j.graphdb.event.ErrorState
- org.neo4j.graphdb.event.KernelEventHandler.ExecutionOrder
- org.neo4j.graphdb.event.KernelEventHandler
- org.neo4j.graphdb.event.TransactionEventHandler.Adapter
- org.neo4j.graphdb.facade.GraphDatabaseDependencies
- org.neo4j.graphdb.facade.GraphDatabaseFacadeFactory.Dependencies
- org.neo4j.graphdb.facade.GraphDatabaseFacadeFactory
- org.neo4j.graphdb.facade.embedded.EmbeddedGraphDatabase
- org.neo4j.graphdb.facade.spi.ClassicCoreSPI
- org.neo4j.graphdb.facade.spi.ProcedureGDBFacadeSPI
- org.neo4j.graphdb.factory.Description
- org.neo4j.graphdb.factory.EditionLocksFactories
- org.neo4j.graphdb.factory.EnterpriseGraphDatabaseFactory
- org.neo4j.graphdb.factory.GraphDatabaseBuilder.DatabaseCreator
- org.neo4j.graphdb.factory.GraphDatabaseBuilder.Delegator
- org.neo4j.graphdb.factory.GraphDatabaseBuilder
- org.neo4j.graphdb.factory.GraphDatabaseFactoryState
- org.neo4j.graphdb.factory.GraphDatabaseFactory
- org.neo4j.graphdb.factory.GraphDatabaseSettings.Connector.ConnectorType
- org.neo4j.graphdb.factory.GraphDatabaseSettings.Connector
- org.neo4j.graphdb.factory.GraphDatabaseSettings.LabelIndex

- org.neo4j.graphdb.factory.HighlyAvailableGraphDatabaseFactory
- org.neo4j.graphdb.factory.module.DataSourceModule
- org.neo4j.graphdb.factory.module.ModularDatabaseCreationContext
- org.neo4j.graphdb.factory.module.PlatformModule
- org.neo4j.graphdb.factory.module.ProcedureGDSFactory
- org.neo4j.graphdb.factory.module.edition.AbstractEditionModule
- org.neo4j.graphdb.factory.module.edition.CommunityEditionModule
- org.neo4j.graphdb.factory.module.edition.DefaultEditionModule
- org.neo4j.graphdb.factory.module.edition.context.DatabaseEditionContext
- org.neo4j.graphdb.factory.module.edition.context.DefaultEditionModuleDatabaseContext
- org.neo4j.graphdb.factory.module.id.DatabaseIdContext
- org.neo4j.graphdb.factory.module.id.IdContextFactoryBuilder
- org.neo4j.graphdb.factory.module.id.IdContextFactory
- org.neo4j.graphdb.index.AutoIndexer
- org.neo4j.graphdb.index.IndexHits
- org.neo4j.graphdb.index.IndexManager
- org.neo4j.graphdb.index.IndexPopulationProgress
- org.neo4j.graphdb.index.Index
- org.neo4j.graphdb.index.ReadableIndex
- org.neo4j.graphdb.index.ReadableRelationshipIndex
- org.neo4j.graphdb.index.RelationshipAutoIndexer
- org.neo4j.graphdb.index.RelationshipIndex
- org.neo4j.graphdb.index.UniqueFactory.UniqueEntity
- org.neo4j.graphdb.index.UniqueFactory.UniqueNodeFactory
- org.neo4j.graphdb.index.UniqueFactory.UniqueRelationshipFactory
- org.neo4j.graphdb.index.UniqueFactory
- org.neo4j.graphdb.security.AuthProviderFailedException
- org.neo4j.graphdb.security.AuthProviderTimeoutException
- org.neo4j.graphdb.security.AuthorizationExpiredException
- org.neo4j.graphdb.security.AuthorizationViolationException
- org.neo4j.graphdb.security.URLAccessRule
- org.neo4j.graphdb.security.URLAccessValidationError
- org.neo4j.graphdb.security.WriteOperationsNotAllowedException
- org.neo4j.graphdb.traversal.AlternatingSelectorOrderer
- org.neo4j.graphdb.traversal.BidirectionalTraversalDescription

- org.neo4j.graphdb.traversal.BidirectionalUniquenessFilter
- org.neo4j.graphdb.traversal.BranchCollisionDetector
- org.neo4j.graphdb.traversal.BranchCollisionPolicies
- org.neo4j.graphdb.traversal.BranchCollisionPolicy
- org.neo4j.graphdb.traversal.BranchOrderingPolicies
- org.neo4j.graphdb.traversal.BranchOrderingPolicy
- org.neo4j.graphdb.traversal.BranchSelector
- org.neo4j.graphdb.traversal.BranchState
- org.neo4j.graphdb.traversal.Evaluation
- org.neo4j.graphdb.traversal.Evaluator.AsPathEvaluator
- org.neo4j.graphdb.traversal.Evaluator
- org.neo4j.graphdb.traversal.Evaluators
- org.neo4j.graphdb.traversal.InitialBranchState.Adapter
- org.neo4j.graphdb.traversal.InitialBranchState.State
- org.neo4j.graphdb.traversal.InitialBranchState
- org.neo4j.graphdb.traversal.LevelSelectorOrderer
- org.neo4j.graphdb.traversal.PathEvaluator.Adapter
- org.neo4j.graphdb.traversal.PathEvaluator
- org.neo4j.graphdb.traversal.Paths.DefaultPathDescriptor
- org.neo4j.graphdb.traversal.Paths.PathDescriptor
- org.neo4j.graphdb.traversal.Paths
- org.neo4j.graphdb.traversal.SideSelectorPolicies
- org.neo4j.graphdb.traversal.SideSelectorPolicy
- org.neo4j.graphdb.traversal.SideSelector
- org.neo4j.graphdb.traversal.Sorting
- org.neo4j.graphdb.traversal.TraversalBranch
- org.neo4j.graphdb.traversal.TraversalContext
- org.neo4j.graphdb.traversal.TraversalDescription
- org.neo4j.graphdb.traversal.TraversalMetadata
- org.neo4j.graphdb.traversal.Traverser
- org.neo4j.graphdb.traversal.UniquenessFactory
- org.neo4j.graphdb.traversal.UniquenessFilter
- org.neo4j.graphdb.traversal.Uniqueness
- org.neo4j.helpers.AdvertisedSocketAddress
- org.neo4j.helpers.Args.ArgsParser

- org.neo4j.helpers.Args.Option
- org.neo4j.helpers.Args
- org.neo4j.helpers.ArrayUtil.ArrayEquality
- org.neo4j.helpers.ArrayUtil
- org.neo4j.helpers.Assertion
- org.neo4j.helpers.Cancelable
- org.neo4j.helpers.CancellationRequest
- org.neo4j.helpers.Clock
- org.neo4j.helpers.CloneableInPublic
- org.neo4j.helpers.Exceptions
- org.neo4j.helpers.Format
- org.neo4j.helpers.FutureAdapter.Present
- org.neo4j.helpers.FutureAdapter
- org.neo4j.helpers.HostnamePort
- org.neo4j.helpers.ListenSocketAddress
- org.neo4j.helpers.Listeners.Notification
- org.neo4j.helpers.Listeners
- org.neo4j.helpers.MathUtil
- org.neo4j.helpers.NamedThreadFactory.Monitor
- org.neo4j.helpers.NamedThreadFactory
- org.neo4j.helpers.Numbers
- org.neo4j.helpers.PortBindException
- org.neo4j.helpers.ProcessFailureException.Entry
- org.neo4j.helpers.ProcessFailureException
- org.neo4j.helpers.Reference
- org.neo4j.helpers.RunCarefully
- org.neo4j.helpers.Service.Implementation
- org.neo4j.helpers.Service
- org.neo4j.helpers.SocketAddressParser
- org.neo4j.helpers.Strings
- org.neo4j.helpers.TaskControl
- org.neo4j.helpers.TaskCoordinator
- org.neo4j.helpers.TextUtil
- org.neo4j.helpers.ThisShouldNotHappenError
- org.neo4j.helpers.TimeUtil

- org.neo4j.helpers.TransactionTemplate.Monitor.Adapter
- org.neo4j.helpers.TransactionTemplate.Monitor
- org.neo4j.helpers.TransactionTemplate
- org.neo4j.helpers.Uris
- org.neo4j.helpers.collection.ArrayIterator
- org.neo4j.helpers.collection.BoundedIterable
- org.neo4j.helpers.collection.CachingIterator
- org.neo4j.helpers.collection.CastingIterator
- org.neo4j.helpers.collection.CatchingIteratorWrapper
- org.neo4j.helpers.collection.CollectorsUtil
- org.neo4j.helpers.collection.CombiningIterable
- org.neo4j.helpers.collection.CombiningIterator
- org.neo4j.helpers.collection.CombiningResourceIterator
- org.neo4j.helpers.collection.ExceptionHandlingIterable
- org.neo4j.helpers.collection.FilteringIterable
- org.neo4j.helpers.collection.FilteringIterator
- org.neo4j.helpers.collection.FirstItemIterable
- org.neo4j.helpers.collection.IterableWrapper
- org.neo4j.helpers.collection.Iterables
- org.neo4j.helpers.collection.IteratorWrapper
- org.neo4j.helpers.collection.Iterators
- org.neo4j.helpers.collection.LimitingResourceIterable
- org.neo4j.helpers.collection.LimitingResourceIterator
- org.neo4j.helpers.collection.LruCache
- org.neo4j.helpers.collection.MapUtil.MapBuilder
- org.neo4j.helpers.collection.MapUtil
- org.neo4j.helpers.collection.MappingResourceIterator
- org.neo4j.helpers.collection.MultiSet
- org.neo4j.helpers.collection.NestingIterable
- org.neo4j.helpers.collection.NestingIterator
- org.neo4j.helpers.collection.NestingResourceIterator
- org.neo4j.helpers.collection.PagingIterator
- org.neo4j.helpers.collection.Pair
- org.neo4j.helpers.collection.PrefetchingIterator
- org.neo4j.helpers.collection.PrefetchingResourceIterator

- org.neo4j.helpers.collection.RangeIterator
- org.neo4j.helpers.collection.ResourceClosingIterator
- org.neo4j.helpers.collection.ResourceIterableWrapper
- org.neo4j.helpers.collection.ReverseArrayIterator
- org.neo4j.helpers.collection.Visitable
- org.neo4j.helpers.collection.Visitor.SafeGenerics
- org.neo4j.helpers.collection.Visitor
- org.neo4j.index.lucene.LuceneKernelExtensionFactory.Dependencies
- org.neo4j.index.lucene.LuceneKernelExtensionFactory
- org.neo4j.index.lucene.LuceneKernelExtension
- org.neo4j.index.lucene.LuceneTimeline
- org.neo4j.index.lucene.QueryContext
- org.neo4j.index.lucene.TimelineIndex
- org.neo4j.index.lucene.ValueContext
- org.neo4j.index.lucene.unsafe.batchinsert.LuceneBatchInserterIndexProvider
- org.neo4j.jmx.Description
- org.neo4j.jmx.JmxUtils
- org.neo4j.jmx.Kernel
- org.neo4j.jmx.ManagementInterface
- org.neo4j.jmx.Primitives
- org.neo4j.jmx.StoreFile
- org.neo4j.jmx.StoreSize
- org.neo4j.logging.AbstractLogProvider
- org.neo4j.logging.AbstractLog
- org.neo4j.logging.AbstractPrintWriterLogger
- org.neo4j.logging.BufferingLog
- org.neo4j.logging.DuplicatingLogProvider
- org.neo4j.logging.DuplicatingLog
- org.neo4j.logging.FormattedLog.Builder
- org.neo4j.logging.FormattedLogProvider.Builder
- org.neo4j.logging.FormattedLogProvider
- org.neo4j.logging.FormattedLog
- org.neo4j.logging.NullLogProvider
- org.neo4j.logging.NullLog
- org.neo4j.logging.NullLogger

- org.neo4j.logging.PrintStreamLogger
- org.neo4j.logging.RotatingFileOutputStreamSupplier.RotationListener
- org.neo4j.logging.RotatingFileOutputStreamSupplier
- org.neo4j.logging.slf4j.Slf4jLogProvider
- org.neo4j.logging.slf4j.Slf4jLog
- org.neo4j.management.BranchedStoreInfo
- org.neo4j.management.BranchedStore
- org.neo4j.management.CausalClustering
- org.neo4j.management.ClusterDatabaseInfo
- org.neo4j.management.ClusterMemberInfo
- org.neo4j.management.Diagnostics
- org.neo4j.management.HighAvailability
- org.neo4j.management.IndexSamplingManager
- org.neo4j.management.LockManager
- org.neo4j.management.MemoryMapping
- org.neo4j.management.Neo4jManager
- org.neo4j.management.PageCache
- org.neo4j.management.RemoteConnection
- org.neo4j.management.TransactionManager
- org.neo4j.management.WindowPoolInfo
- org.neo4j.procedure.Admin
- org.neo4j.procedure.PerformsWrites
- org.neo4j.procedure.ProcedureTransaction
- org.neo4j.procedure.TerminationGuard
- org.neo4j.server.helpers.PropertyTypeDispatcher.PropertyArray
- org.neo4j.server.helpers.PropertyTypeDispatcher
- org.neo4j.server.plugins.BadPluginInvocationException
- org.neo4j.server.plugins.ConfigAdapter
- org.neo4j.server.plugins.DefaultPluginManager
- org.neo4j.server.plugins.Description
- org.neo4j.server.plugins.DisabledPluginManager
- org.neo4j.server.plugins.Injectable
- org.neo4j.server.plugins.MapTypeCaster
- org.neo4j.server.plugins.Name
- org.neo4j.server.plugins.ParameterDescriptionConsumer

- org.neo4j.server.plugins.ParameterList
- org.neo4j.server.plugins.Parameter
- org.neo4j.server.plugins.PluginInvocationFailureException
- org.neo4j.server.plugins.PluginInvocatorProvider
- org.neo4j.server.plugins.PluginInvocator
- org.neo4j.server.plugins.PluginLifecycle
- org.neo4j.server.plugins.PluginLookupException
- org.neo4j.server.plugins.PluginManager
- org.neo4j.server.plugins.PluginPoint
- org.neo4j.server.plugins.PluginTarget
- org.neo4j.server.plugins.SPIPluginLifecycle
- org.neo4j.server.plugins.ServerExtender
- org.neo4j.server.plugins.ServerPlugin
- org.neo4j.server.plugins.Source
- org.neo4j.server.rest.repr.AuthorizationRepresentation
- org.neo4j.server.rest.repr.BadInputException
- org.neo4j.server.rest.repr.ConstraintDefinitionRepresentation
- org.neo4j.server.rest.repr.CypherPlanRepresentation
- org.neo4j.server.rest.repr.CypherRepresentationDispatcher
- org.neo4j.server.rest.repr.CypherResultRepresentation
- org.neo4j.server.rest.repr.CypherStatisticsRepresentation
- org.neo4j.server.rest.repr.DatabaseRepresentation
- org.neo4j.server.rest.repr.DefaultFormat
- org.neo4j.server.rest.repr.DiscoveryRepresentation
- org.neo4j.server.rest.repr.EntityRepresentation
- org.neo4j.server.rest.repr.ExceptionRepresentation
- org.neo4j.server.rest.repr.ExtensionInjector
- org.neo4j.server.rest.repr.ExtensionPointRepresentation
- org.neo4j.server.rest.repr.FullPath
- org.neo4j.server.rest.repr.IndexDefinitionRepresentation
- org.neo4j.server.rest.repr.IndexRepresentation
- org.neo4j.server.rest.repr.IndexedEntityRepresentation
- org.neo4j.server.rest.repr.InputFormatProvider
- org.neo4j.server.rest.repr.InputFormat
- org.neo4j.server.rest.repr.InvalidArgumentsException

- org.neo4j.server.rest.repr.ListRepresentation
- org.neo4j.server.rest.repr.ListSerializer
- org.neo4j.server.rest.repr.ListWriter
- org.neo4j.server.rest.repr.MapRepresentation
- org.neo4j.server.rest.repr.MappingRepresentation
- org.neo4j.server.rest.repr.MappingSerializer
- org.neo4j.server.rest.repr.MappingWriter
- org.neo4j.server.rest.repr.MediaTypeNotSupportedException
- org.neo4j.server.rest.repr.NodeIndexRepresentation
- org.neo4j.server.rest.repr.NodeIndexRootRepresentation
- org.neo4j.server.rest.repr.NodeRepresentation
- org.neo4j.server.rest.repr.ObjectRepresentation
- org.neo4j.server.rest.repr.ObjectToRepresentationConverter
- org.neo4j.server.rest.repr.OutputFormatProvider
- org.neo4j.server.rest.repr.OutputFormat
- org.neo4j.server.rest.repr.PathRepresentation
- org.neo4j.server.rest.repr.PropertiesRepresentation
- org.neo4j.server.rest.repr.RelationshipIndexRepresentation
- org.neo4j.server.rest.repr.RelationshipIndexRootRepresentation
- org.neo4j.server.rest.repr.RelationshipRepresentation
- org.neo4j.server.rest.repr.RepresentationDispatcher
- org.neo4j.server.rest.repr.RepresentationExceptionHandlingIterable
- org.neo4j.server.rest.repr.RepresentationFormatRepository
- org.neo4j.server.rest.repr.RepresentationFormat
- org.neo4j.server.rest.repr.RepresentationType
- org.neo4j.server.rest.repr.RepresentationWriteHandler
- org.neo4j.server.rest.repr.Representation
- org.neo4j.server.rest.repr.ScoredEntityRepresentation
- org.neo4j.server.rest.repr.ScoredNodeRepresentation
- org.neo4j.server.rest.repr.ScoredRelationshipRepresentation
- org.neo4j.server.rest.repr.ServerExtensionRepresentation
- org.neo4j.server.rest.repr.ServerListRepresentation
- org.neo4j.server.rest.repr.StreamingFormat
- org.neo4j.server.rest.repr.ValueRepresentation
- org.neo4j.server.rest.repr.WeightedPathRepresentation

- org.neo4j.server.rest.web.BatchOperationService
- org.neo4j.server.rest.web.CollectUserAgentFilter
- org.neo4j.server.rest.web.CorsFilter
- org.neo4j.server.rest.web.CustomStatusType
- org.neo4j.server.rest.web.CypherService
- org.neo4j.server.rest.web.DatabaseActions.Provider
- org.neo4j.server.rest.web.DatabaseActions.RelationshipDirection
- org.neo4j.server.rest.web.DatabaseActions
- org.neo4j.server.rest.web.DatabaseMetadataService
- org.neo4j.server.rest.web.ExtensionService
- org.neo4j.server.rest.web.HttpConnectionInfoFactory
- org.neo4j.server.rest.web.InternalJettyServletRequest.RequestData
- org.neo4j.server.rest.web.InternalJettyServletRequest
- org.neo4j.server.rest.web.InternalJettyServletResponse
- org.neo4j.server.rest.web.NoSuchPropertyException
- org.neo4j.server.rest.web.NodeNotFoundException
- org.neo4j.server.rest.web.PropertyValueException
- org.neo4j.server.rest.web.RelationshipNotFoundException
- org.neo4j.server.rest.web.RestfulGraphDatabase.AmpersandSeparatedCollection
- org.neo4j.server.rest.web.RestfulGraphDatabase
- org.neo4j.server.rest.web.StreamingBatchOperations
- org.neo4j.server.rest.web.Surface
- org.neo4j.server.rest.web.TransactionUriScheme
- org.neo4j.server.rest.web.TransactionalService.TransactionUriBuilder
- org.neo4j.server.rest.web.TransactionalService
- org.neo4j.unsafe.batchinsert.BatchInserterIndexProvider
- org.neo4j.unsafe.batchinsert.BatchInserterIndex
- org.neo4j.unsafe.batchinsert.BatchInserter
- org.neo4j.unsafe.batchinsert.BatchInserters
- org.neo4j.unsafe.batchinsert.BatchRelationship

REST API

This section describes the removal of Neo4j REST API.

The REST API has been removed in Neo4j 4.0. Cypher commands and procedures must be used instead, either via the HTTP API, or via Bolt using the official drivers.

The following HTTP endpoints were deprecated in Neo4j 3.4 and have now been removed:

- `/db/data/branch`
- `/db/data/cypher`
- `/db/data/index/node`
- `/db/data/index/relationship`
- `/db/data/labels`
- `/db/data/node`
- `/db/data/relationship`
- `/db/data/relationship/types`
- `/db/data/schema/constraint`
- `/db/data/schema/index`
- `/db/data/schema/relationship/constraint`

HTTP API endpoints

This section describes breaking changes for the Neo4j HTTP API endpoints.

The HTTP API endpoints accommodate multi-database features. For example, the URI to begin a transaction has changed from http://localhost:7474/db/data/transaction to http://localhost:33471/db/neo4j/tx.
More generally, the HTTP API endpoints follow the pattern: http://localhost:33471/db/<database_name>/tx.

External dependencies

This section lists the external dependencies in Neo4j 4.x.

Group ID	Artifact ID	Version
`com.fasterxml.jackson.core`	`jackson-annotations`	`2.10.0`
`com.fasterxml.jackson.core`	`jackson-core`	`2.10.0`
`com.fasterxml.jackson.core`	`jackson-databind`	`2.10.0`
`com.fasterxml.jackson.jaxrs`	`jackson-jaxrs-base`	`2.10.0`
`com.fasterxml.jackson.jaxrs`	`jackson-jaxrs-json-provider`	`2.10.0`
`com.fasterxml.jackson.module`	`jackson-module-jaxb-annotations`	`2.10.0`
`com.github.ben-manes.caffeine`	`caffeine`	`2.8.0`
`com.github.luben`	`zstd-jni`	`1.4.3-1`

Group ID	Artifact ID	Version
commons-beanutils	commons-beanutils	1.9.4
commons-collections	commons-collections	3.2.2
commons-configuration	commons-configuration	1.10
commons-io	commons-io	2.6
commons-lang	commons-lang	2.6
commons-logging	commons-logging	1.2
com.profesorfalken	jPowerShell	3.0
com.profesorfalken	WMI4Java	1.6.3
com.sun.activation	jakarta.activation	1.2.1
com.sun.istack	istack-commons-runtime	3.0.8
com.sun.xml.fastinfoset	FastInfoset	1.2.16
com.typesafe.akka	akka-actor_2.12	2.5.22
com.typesafe.akka	akka-cluster_2.12	2.5.22
com.typesafe.akka	akka-cluster-tools_2.12	2.5.22
com.typesafe.akka	akka-coordination_2.12	2.5.22
com.typesafe.akka	akka-distributed-data_2.12	2.5.22
com.typesafe.akka	akka-protobuf_2.12	2.5.22
com.typesafe.akka	akka-remote_2.12	2.5.22
com.typesafe.akka	akka-stream_2.12	2.5.22
com.typesafe	config	1.3.3
com.typesafe	ssl-config-core_2.12	0.3.7
info.picocli	picocli	4.0.4
io.aeron	aeron-client	1.15.1
io.aeron	aeron-driver	1.15.1
io.dropwizard.metrics	metrics-core	4.1.0
io.dropwizard.metrics	metrics-graphite	4.1.0
io.dropwizard.metrics	metrics-jmx	4.1.0
io.netty	netty-all	4.1.35.Final
io.netty	netty	3.10.6.Final
io.projectreactor	reactor-core	3.2.10.RELEASE
io.prometheus	simpleclient_common	0.7.0
io.prometheus	simpleclient_dropwizard	0.7.0
io.prometheus	simpleclient_httpserver	0.7.0
io.prometheus	simpleclient	0.7.0
jakarta.activation	jakarta.activation-api	1.2.1
jakarta.annotation	jakarta.annotation-api	1.3.4
jakarta.ws.rs	jakarta.ws.rs-api	2.1.5

Group ID	Artifact ID	Version
jakarta.xml.bind	jakarta.xml.bind-api	2.3.2
javax.activation	activation	1.1.1
javax.servlet	javax.servlet-api	3.1.0
javax.validation	validation-api	2.0.1.Final
javax.ws.rs	javax.ws.rs-api	2.1.1
javax.xml.bind	jaxb-api	2.3.0
jline	jline	2.14.3
net.java.dev.jna	jna	5.4.0
net.jpountz.lz4	lz4	1.3.0
org.agrona	agrona	0.9.31
org.apache.commons	commons-compress	1.19
org.apache.commons	commons-lang3	3.9
org.apache.commons	commons-text	1.7
org.apache.lucene	lucene-analyzers-common	8.2.0
org.apache.lucene	lucene-codecs	8.2.0
org.apache.lucene	lucene-core	8.2.0
org.apache.lucene	lucene-queryparser	8.2.0
org.apache.shiro	shiro-cache	1.4.1
org.apache.shiro	shiro-config-core	1.4.1
org.apache.shiro	shiro-config-ogdl	1.4.1
org.apache.shiro	shiro-core	1.4.1
org.apache.shiro	shiro-crypto-cipher	1.4.1
org.apache.shiro	shiro-crypto-core	1.4.1
org.apache.shiro	shiro-crypto-hash	1.4.1
org.apache.shiro	shiro-event	1.4.1
org.apache.shiro	shiro-lang	1.4.1
org.bitbucket.inkytonik.kiama	kiama_2.12	2.1.0
org.bouncycastle	bcpkix-jdk15on	1.63
org.bouncycastle	bcprov-jdk15on	1.63
org.eclipse.collections	eclipse-collections-api	10.0.0
org.eclipse.collections	eclipse-collections	10.0.0
org.eclipse.jetty	jetty-client	9.4.17.v20190418
org.eclipse.jetty	jetty-http	9.4.17.v20190418
org.eclipse.jetty	jetty-io	9.4.17.v20190418
org.eclipse.jetty	jetty-security	9.4.17.v20190418
org.eclipse.jetty	jetty-server	9.4.17.v20190418
org.eclipse.jetty	jetty-servlet	9.4.17.v20190418

Group ID	Artifact ID	Version
org.eclipse.jetty	jetty-util	9.4.17.v20190418
org.eclipse.jetty	jetty-webapp	9.4.17.v20190418
org.eclipse.jetty	jetty-xml	9.4.17.v20190418
org.glassfish.hk2.external	jakarta.inject	2.5.0
org.glassfish.hk2	hk2-api	2.5.0
org.glassfish.hk2	hk2-locator	2.5.0
org.glassfish.hk2	hk2-utils	2.5.0
org.glassfish.jaxb	jaxb-runtime	2.3.2
org.glassfish.jaxb	txw2	2.3.2
org.glassfish.jersey.containers	jersey-container-servlet-core	2.29
org.glassfish.jersey.containers	jersey-container-servlet	2.29
org.glassfish.jersey.core	jersey-client	2.29
org.glassfish.jersey.core	jersey-common	2.29
org.glassfish.jersey.core	jersey-server	2.29
org.glassfish.jersey.inject	jersey-hk2	2.29
org.glassfish.jersey.media	jersey-media-jaxb	2.29
org.javassist	javassist	3.22.0-CR2
org.jprocesses	jProcesses	1.6.5
org.jvnet.staxex	stax-ex	1.8.1
org.neo4j.licensing-proxy	zstd-proxy	4.0.0-SNAPSHOT
org.ow2.asm	asm-analysis	7.2
org.ow2.asm	asm	7.2
org.ow2.asm	asm-tree	7.2
org.ow2.asm	asm-util	7.2
org.parboiled	parboiled-core	1.2.0
org.parboiled	parboiled-scala_2.12	1.2.0
org.reactivestreams	reactive-streams	1.0.2
org.rogach	scallop_2.12	2.1.1
org.scala-lang.modules	scala-java8-compat_2.12	0.8.0
org.scala-lang.modules	scala-parser-combinators_2.12	1.1.1
org.scala-lang	scala-library	2.12.7
org.scala-lang	scala-reflect	2.12.7
org.slf4j	slf4j-api	1.7.25
org.slf4j	slf4j-nop	1.7.25

7.9.2. Breaking changes between Neo4j 1.7 drivers and Neo4j 4.x drivers

This chapter describes the breaking changes between Neo4j 1.7 drivers and Neo4j 4.x drivers.

It covers the following topics:

Drivers changes in 4.x

This section the difference between the Neo4j 1.7 drivers and the Neo4j 4.x drivers.

High-level changes

From 4.0, the drivers are built to provide a user-friendly and unified API across all languages.

In previous versions of Neo4j, client-server communication uses encrypted local connections and generates a self-signed certificate out of the box. From 4.0 however, the default is set to unencrypted. For more information, see Driver Manual → Connection URIs.

Neo4j 4.0 introduces a reactive API compatible with the Reactive Streams standard. This enables fine-grained control of the data flow for Cypher query results, including the ability to pause or cancel part-way through. For more information, see Driver Manual → Queries and results.

When using the 4.x driver to connect to a 4.x database, it is possible to work with multiple databases. From a driver API perspective, this means that one database must be selected for use as an execution context for transactions within a session. This can be configured on session construction. If no database is selected, the driver connects to the server's default database.

Drivers 1.7 work in fallback mode with Neo4j 4.x. They do not support features introduced in Neo4j 4.0 and later, such as multiple databases, Neo4j Fabric, and fine-grained access control. To run multiple databases online concurrently and do distributed queries over them, you must migrate from 1.7 to 4.x.

The examples in this chapter are mainly written in Java, using the Java driver. However, similar code can be translated to other languages.

New features

- Starting with Neo4j 4.0, the versioning scheme for the database, driver, and protocol are all aligned. For supported drivers, this means that the version number goes from 1.7 to any 4.x.
- A 1.7+ driver communicating with a 4.x server may need encryption explicitly switched off. This is due to a change in the defaults between Neo4j 3.x and 4.x.
- Bolt 4.0 is implemented in both 4.x drivers and 4.0 servers.
- Reactive API is now available with 4.0 servers. To make use of the reactive API, the starting point is `RxSession` on the driver object.
- With 4.0 servers, session instances should now be acquired against a specific database. Causal chaining is still respected on each database (transactions cannot span across multiple databases). The driver itself connects to Neo4j DBMS.
- A new feature detection method `driver.supportsMultiDb()` is added for querying whether the remote database supports multiple databases.
- A new `driver.verifyConnectivity()` method is introduced for connectivity verification purposes. By default, the driver instances do not verify DBMS availability after construction.
- New connection URI schemes with variants that contain extra encryption and trust information - `neo4j+s`, `bolt+s`, `neo4j+ssc`, and `bolt+ssc`. The `+s` variants enable encryption with a full certificate check, and the `+ssc` variants enable encryption with no certificate check. The latter variant is designed specifically for use with self-signed certificates. For more information, see Additional URI Schemes.

Example 4. Detecting multiple database support

```
import org.neo4j.driver.Driver;
import org.neo4j.driver.Result;
import org.neo4j.driver.Session;
import org.neo4j.driver.SessionConfig;
import org.neo4j.driver.Values;

...
private final Driver driver;
...

public void printGreeting( final String message )
{
   SessionConfig sessionConfig = driver.supportsMultiDb() ? SessionConfig.forDatabase( "neo4j" )
                                                          : SessionConfig.defaultConfig();

   try ( Session session = driver.session( sessionConfig ) )
   {
       String greeting = session.writeTransaction( tx -> {
           Result result = tx.run( "CREATE (a:Greeting) SET a.message = $message RETURN a.message +
', from node ' + id(a)",
                                   Values.parameters( "message", message ) );
           return result.single().get( 0 ).asString();
       } );
       System.out.println( greeting );
   }
}
```

Compatibility

This section shows the version supportability of the Bolt protocol and drivers in Neo4j 3.5 and 4.x.

The compatibility between the different versions of the Bolt protocol, the drivers, and Neo4j 3.5 and 4.x is illustrated in the tables below:

Bolt protocol in Neo4j 3.5 and 4.x

	Neo4j 4.x	Neo4j 3.5
Bolt v4.x	All features fully supported.	Not supported.
Bolt v3	All features fully supported, but the support may be removed in next version.	All features fully supported.
Bolt v2	Not supported.	All features fully supported, but the support may be removed in next version.
Bolt v1	Not supported.	All features fully supported, but the support may be removed in next version.

Drivers in Neo4j 3.5 and 4.x

	Neo4j 4.x		Neo4j 3.5	
	Bolt version	Support	Bolt version	Support

	Neo4j 4.x		Neo4j 3.5	
Java Driver 4.x	Bolt v4.x	All features fully supported.	Bolt v3	All features fully supported, but the support may be removed in next version.
.NET Driver 4.x	Bolt v4.x	All features fully supported.	Bolt v3	All features fully supported, but the support may be removed in next version.
JavaScript Driver 4.x	Bolt v4.x	All features fully supported.	Bolt v3	All features fully supported, but the support may be removed in next version.
Python Driver 4.x	Bolt v4.x	All features fully supported.	Bolt v3	All features fully supported, but the support may be removed in next version.
Go Driver 4.x	Bolt v4.x	All features fully supported.	Bolt v3	All features fully supported, but the support may be removed in next version.

Breaking changes

This section describes the breaking changes between Neo4j 1.7 drivers and 4.x drivers.

Configuration

- The drivers' default configuration for encrypted is now false, meaning that driver will attempt only plain text connections by default. Connections to encrypted services (such as Neo4j Aura) must be set to encrypted.
- When encryption is explicitly enabled, the default is to trust the CAs trusted by the operating system. This means that, by default, encrypted connections to servers holding self-signed certificates fail on certificate verification.
- Hostname verification is turned on by default when encryption is turned on.

For more information, see The Neo4j Drivers Manual 4.0 → Configuration.

Removal of the version suffix

`v1` is removed from drivers' package name. For example, in the Java driver, all public APIs are in the package `org.neo4j.driver` instead of the old `org.neo4j.driver.v1`.

Connection URI scheme

Neo4j 4.0 exposes a routing interface for all deployment topologies, allowing `neo4j://` URIs to be used for all deployments.

Neo4j 3.x standalone instances do not expose a routing interface.

- The `neo4j://` scheme replaces `bolt+routing://` and can be used for both clustered and single-instance configurations. This is a rename only, and `neo4j://` URIs can still be used to communicate with Neo4j 3.x clusters.
- The `bolt://` scheme is used for direct connection to a particular Neo4j server. However, this scheme is no longer required for standalone machines.
- The `bolt://` scheme is mainly useful when not targeting an entire service, but rather a specific machine, such as a certain server in a Causal Cluster or the one server in a single-instance environment.
- The recommended Driver Connection URI scheme is as follows:

  Recommended Driver Connection URI scheme.

		4.0 drivers	1.7 drivers
4.0 Neo4j	Single instance	`neo4j`	`bolt`
	Cluster core members	`neo4j`	`neo4j` (`bolt`+routing)
	Cluster read replicas	`neo4j`	`bolt`
3.5 Neo4j	Single instance	`bolt`	`bolt`
	Cluster core members	`neo4j`	`neo4j` (`bolt`+routing)
	Cluster read replicas	`bolt`	`bolt`

Changes in drivers' methods

- For drivers where synchronous and asynchronous methods are both implemented, asynchronous methods are extracted out and put in `AsyncSession`, whereas synchronous methods remain in `Session`. This change ensures that blocking and non-blocking APIs can never be mixed together.
- `Driver#session` method uses a session configuration object or option builder, rather than method arguments.
- Bookmark has changed from a string, and/or a list of strings, to a Bookmark object.
- For synchronous Transaction API, `Transaction#success` and `Transaction#failure` are removed.
- The `success`/`close` pattern for Transaction objects is now obsolete and is fully superseded by `commit` and `rollback` methods. However, unlike `Transaction#success`, which only marks the transaction as successful and then waits for `Transaction#close` to perform the actual commit, `Transaction#commit` commits the transaction immediately.

  A transaction in 4.0 can only be committed or rolled back once. If a transaction is not committed explicitly using `Transaction#commit`, `Transaction#close` will roll back the transaction.

* `Statement` is renamed to `Query`. `StatementResult` is renamed to `Result`. Similarly, `StatementResultCursor` is renamed to `ResultCursor`.
* A result can only be consumed once.
* A result is consumed if either the query result has been discarded by invoking `Result#consume`, and/or the outer scope where the result is created, such as a transaction or a session, has been closed. Attempts to access consumed results are responded with a `ResultConsumedException`.
* The experimental `StatementRunner.typeSystem()` is moved to `Driver.defaultTypeSystem()`.
* `LoadBalancingStrategy` is removed from Config class and the drivers always default to `LeastConnectedStrategy`.

For more information on breaking changes for the specific drivers, see their respective sections:

* Go driver
* Java driver
* Javascript driver
* .NET driver
* Python driver

Back-pressure

This section describes back-pressure in the 4.x drivers.

Neo4j 4.0 introduces client-side back-pressure. The concept of client-side back-pressure is that the client communicates with the remote server regarding how much data it is able to process and only requests additional data when it is ready to consume more.

The back-pressure concept is naturally compatible with Reactive programming. As a result, Reactive API support is added into all language drivers in 4.0 driver releases.

The Java driver’s Reactive API exposes a raw Publisher-Subscriber API, which is defined by reactive streams. The Java driver’s Reactive API is used with a reactive library, such as Project Reactor and/or RxJava.

The .NET uses the built-in System.Reactive. The JavaScript driver uses the RxJs library.

These libraries belong to the same reactive framework ReactiveX.

To use the drivers’ Reactive API, preliminary knowledge of reactive programming is necessary. Details of how to use Neo4j Reactive Driver API can be found in the Neo4j Drivers Manual 4.0.

However, back-pressure is not only limited to the drivers' Reactive APIs. All other APIs, such as simple and async, by default have back-pressure enabled when handling query execution results.

How back-pressure is implemented in the different language driver session APIs

	Simple API	Async API	Reactive API
Java driver	Record buffer	Record buffer	Raw Publisher-Subscriber API
.NET driver	Record buffer	Record buffer	Record buffer
Javascript driver	Not applicable	Record buffer	Record buffer

Back-pressure with Bolt 4.0 and record buffer

The Neo4j 4.0 server and drivers implement Bolt 4.0. One of the main features introduced in this Bolt version is pulling query results (records) in batches. In previous Bolt versions, the complete result set is always pulled in one batch from a server to a driver. Bolt 4.0 enables you to pull these results in multiple batches where the size of each can be defined by `fetchSize`. By default, the drivers use a `fetchSize` of 1000 records.

With the introduction of batching of records, drivers can implement client-side back-pressure. For each result, the driver keeps a record buffer of unconsumed records. The buffer size is the same as `fetchSize` for each batch. The pulling of records from the server is paused when the buffer is more than 70% full, and the record pulling is re-enabled once the buffer is less than 30% full. With the default `fetchSize` of 1000 records, record pulling is paused when more than 700 records are in the buffer and is resumed when the buffer drops below 300.

Example 5. Set default `fetchSize` on a driver and alter the default value on a session.

```java
import org.neo4j.driver.AuthTokens;
import org.neo4j.driver.Config;
import org.neo4j.driver.Driver;
import org.neo4j.driver.GraphDatabase;
import org.neo4j.driver.Session;
import org.neo4j.driver.SessionConfig;
...

Config config = Config.builder().withFetchSize( 2000 ).build();
Driver driver = GraphDatabase.driver( uri, AuthTokens.basic( user, password ), config );

SessionConfig sessionConfig = SessionConfig.builder()
                                          .withDatabase( "neo4j" )
                                          .withFetchSize( 100 )
                                          .build();
try ( Session session = driver.session( sessionConfig ) ) {...}
```

Java driver Reactive API

The Java driver's Reactive API exposes a very low-level Publisher-Subscriber API. As a result, it does not perform any kind of back-pressure by default. Instead, the driver users should make use of a reactive framework to utilize back-pressure. Depending on the reactive framework, the framework may apply back-pressure by pausing the data-pulling from a Neo4j server, or dropping data when there is too much to process.

Multiple databases

This section explains how to use drivers with multiple databases.

With the addition of multiple databases in 4.0, you can now specify which database to work with. When constructing a session, you can specify in the session configuration which database the session is linked to. Queries will then be executed against that database for the duration of the session. Not specifying a database results in the session being linked to the default database as specified in the server configuration, see Operations manual → The default database. When using 4.0 drivers with 4.0 Neo4j Servers, it is recommended to specify the database of each session explicitly.

Example 6. Selecting a database for a session.

```
import org.neo4j.driver.Driver;
import org.neo4j.driver.Session;
import org.neo4j.driver.SessionConfig;
...

try ( Session session = driver.session( SessionConfig.forDatabase( "neo4j" ) ) ) {...}
```

Neo4j Community Edition does not support multiple databases beyond the `system`and a default database.

For more information on managing multiple databases in Neo4j, see Operations manual → The `system` database and Cypher manual → Administration.

Bookmarks

Bookmarks are generally handled internally by the driver. Applications work with bookmarks directly when chaining sessions.

In a multiple database context, bookmarks can only be passed among sessions for the same database. This is because the bookmarks (and/or transactions) cannot cross multiple databases in Neo4j 4.x except those generated by the `system` database.

Example 7. Using system bookmark with another database to ensure the updated system status.

```
import org.neo4j.driver.Bookmark;
import org.neo4j.driver.Driver;
import org.neo4j.driver.Result;
import org.neo4j.driver.Session;
import org.neo4j.driver.SessionConfig;
...

Bookmark sysBookmark;
try ( Session session = driver.session( SessionConfig.forDatabase( "system" ) ) )
{
   session.writeTransaction( tx -> {
       Result result = tx.run( "CREATE database foo" );
       return result.consume();
   } );
   sysBookmark = session.lastBookmark();
}

try ( Session session = driver.session( SessionConfig.builder().withDatabase( "foo" ).withBookmarks(
sysBookmark ).build() ) )
{
   session.writeTransaction( tx -> {
       Result result = tx.run( "CREATE (n)" );
       return result.consume();
   } );
}
```

Configure SSL Policy for Bolt server and HTTPS server

This section describes how to configure the SSL policy for Bolt and HTTPS servers.

Neo4j 3.5 allows encrypted connections with the default configuration. Self-signed certificates are automatically generated if no certificate is installed before a server starts. However, from 4.0 onwards, the default encryption setting is off and Neo4j no longer generates certificates when none are provided. As a result, Bolt server only allows plaintext connections, and HTTPS server is not enabled by default.

Encryption and certificates differences between 3.5 and 4.x servers

	3.5 Neo4j Bolt Server	4.0 Neo4j Bolt Server	3.5 Neo4j HTTPS Server	4.0 Neo4j HTTPS Server
Server Enabled	Yes	Yes	Yes	No
Encryption on client connections	Optional	Not allowed	Always	Always
Certificates	Auto-generated self-signed certificates if not provided.	None	Auto-generated self-signed certificates if not provided.	None
Default Certificates Path	`$neo4jHome/certificates`	None	`$neo4jHome/certificates`	None

	3.5 Neo4j Bolt Server	4.0 Neo4j Bolt Server	3.5 Neo4j HTTPS Server	4.0 Neo4j HTTPS Server
Default Certificate Names	`neo4j.key` `neo4j.cert`	`private.key` `public.crt`	`neo4j.key` `neo4j.cert`	`private.key` `public.crt`

To re-enable encryption in 4.x, you have to configure the SSL policy in the `neo4j.conf` file. Given certificates named `public.crt` and `private.key` in folder `$neo4jHome/certificates/bolt` for Bolt server, and certificates with the same file names in folder `$neo4jHome/certificates/https` for HTTPS server. The example shows how to turn encryption back on for the Bolt server and re-enable the HTTPS server.

Example 8. Turn encryption on for Bolt 4.0 server.

```
dbms.connector.bolt.enabled=true
dbms.connector.bolt.tls_level=OPTIONAL          # allows both encrypted and unencrypted driver
connections

dbms.ssl.policy.bolt.enabled=true
dbms.ssl.policy.bolt.base_directory=certificates/bolt
#dbms.ssl.policy.bolt.private_key=private.key   # Optional if the file name is the same as the
default.
#dbms.ssl.policy.bolt.public_certificate=public.crt     # Optional if the file name is the same as
the default.
```

Example 9. Enable the HTTPS 4.0 server.

```
dbms.connector.https.enabled=true

dbms.ssl.policy.https.enabled=true
dbms.ssl.policy.https.base_directory=certificates/https
#dbms.ssl.policy.https.private_key=private.key  # Optional if the file name is the same as the
default.
#dbms.ssl.policy.https.public_certificate=public.crt    # Optional if the file name is the same as
the default.
```

Additional URI schemes

This section presents additional URI schemes in 4.x.

Since `4.0.1` of the Java and .NET drivers, and `4.0.2` of the JavaScript driver, you are able to configure the encryption and trust settings of the driver directly through the connection URI.

The `neo4j+s` and `bolt+s` schemes enable encryption and full certificate checks against the system's local CA store. The `neo4j+ssc` and `bolt+ssc` schemes also enable encryption with no certificate checks, typically for use with self-signed certificates.

Available URIs

URI	Routing	Description
`neo4j`	Yes	Unsecured
`neo4j+s`	Yes	Secured with full certificate
`neo4j+ssc`	Yes	Secured with self-signed certificate
`bolt`	No	Unsecured
`bolt+s`	No	Secured with full certificate
`bolt+ssc`	No	Secured with self-signed certificate

Using these new URI schemes is not compatible with configuring encryption and trust with the Configuration API. Otherwise, this does not effect the behavior of the existing `neo4j` and `bolt` schemes.

For more information, see Driver Manual → Connection URIs.

Go driver

This section presents the breaking changes between the Neo4j 1.8 Go Driver and 4.x Go Driver.

The latest version of the Go driver for Neo4j can be found on the Go driver's official page.

- All import statements need to be changed to `"github.com/neo4j/neo4j-go-driver/v4/neo4j"`.
- `NewSession` does not return an error anymore.
- `Result` now exposes a `Single` method, which outputs the only record returned by the query.
- `Record` now directly exposes `Keys` and `Values`.

See full changelog: https://github.com/neo4j/neo4j-go-driver/wiki/4.x-changelog.

Example 10. Example of changes between the 1.8 Go driver and the 4.x Go driver

Example code for the 4.x Go driver

```go
package main
import (
    "fmt"
    "github.com/neo4j/neo4j-go-driver/v4/neo4j"
)
func main() {
    // [...]
    session := driver.NewSession(neo4j
.SessionConfig{
        AccessMode:   neo4j.AccessModeWrite,
        Bookmarks:    []string{bookmark},
        DatabaseName: "neo4j",
    })
    defer session.Close()
    transaction, err := session
.BeginTransaction()
    handleError(err)
    defer transaction.Close()
    result, err := transaction.Run(
        `CREATE (a:Greeting) SET a.message =
$message RETURN a.message + ", from node " +
id(a)`,
        map[string]interface{}{
            "message": "helloWorld",
        },
    )
    record, err := result.Single()
    handleError(err)
    greeting := record.Values[0]
    fmt.Println(greeting)
    handleError(transaction.Commit())
}
```

Example code for the 1.8 Go driver

```go
package main
import (
    "fmt"
    "github.com/neo4j/neo4j-go-driver/neo4j"
)
func main() {
    // [...]
    session, err := driver.NewSession(neo4j
.SessionConfig{
        AccessMode:   neo4j.AccessModeWrite,
        Bookmarks:    []string{bookmark},
        DatabaseName: "neo4j",
    })
    handleError(err)
    defer session.Close()
    transaction, err := session
.BeginTransaction()
    handleError(err)
    defer transaction.Close()
    result, err := transaction.Run(
        `CREATE (a:Greeting) SET a.message =
$message RETURN a.message + ", from node " +
id(a)`,
        map[string]interface{}{
            "message": "helloWorld",
        },
    )
    record, err := single(result)
    handleError(err)
    greeting := record.Values()[0]
    fmt.Println(greeting)
    handleError(transaction.Commit())
}
func single(result neo4j.Result) (neo4j.Record,
error) {
    if !result.Next() {
        return nil, fmt.Errorf("expected at
least 1 result, got none")
    }
    record := result.Record()
    if result.Next() {
        return nil, fmt.Errorf("expected
exactly 1 result, got at least 1 more")
    }
    return record, nil
}
```

Java driver

This section presents the breaking changes between the Neo4j 1.7 Java driver and 4.x Java driver.

The latest version of the Java driver for Neo4j can be found on the Java driver's official page.

See full changelog: https://github.com/neo4j/neo4j-java-driver/wiki/4.0-changelog.

Example 11. Example of changes between the 1.7 Java driver and the 4.0 Java driver

Example code for the 4.0 Java driver

```
import org.neo4j.driver.Bookmark;
import org.neo4j.driver.Driver;
import org.neo4j.driver.Query;
import org.neo4j.driver.Result;
import org.neo4j.driver.Session;
import org.neo4j.driver.SessionConfig;
import org.neo4j.driver.Transaction;
import org.neo4j.driver.Values;
...

private final Driver driver;
...
public void printGreeting( String message,
Bookmark bookmark )
{
   SessionConfig sessionConfig = SessionConfig
.builder()
     .withDatabase( "neo4j" )
     .withDefaultAccessMode( AccessMode.WRITE )
     .withBookmarks( bookmark ).build();

   try ( Session session = driver.session(
sessionConfig );
         Transaction transaction = session
.beginTransaction() )
   {
      Query query = new Query( "CREATE
(a:Greeting) SET a.message = $message RETURN
a.message + ', from node ' + id(a)", Values
.parameters( "message", message ) );

      Result result = transaction.run( query );
      String greeting = result.single().get( 0
).asString();
      System.out.println( greeting );
      transaction.commit(); // commit
immediately here
   }
}
```

Example code for the 1.7 Java driver

```
import org.neo4j.driver.v1.AccessMode;
import org.neo4j.driver.v1.Driver;
import org.neo4j.driver.v1.Session;
import org.neo4j.driver.v1.Statement;
import org.neo4j.driver.v1.StatementResult;
import org.neo4j.driver.v1.Transaction;
import org.neo4j.driver.v1.Values;

...

private final Driver driver;
...
public void printGreeting( String message,
String bookmark )
{



   try ( Session session = driver.session(
AccessMode.WRITE, bookmark );
         Transaction transaction = session
.beginTransaction() )
   {
      Statement query = new Statement( "CREATE
(a:Greeting) SET a.message = $message RETURN
a.message + ', from node ' + id(a)", Values
.parameters( "message", message ) );


      StatementResult result = transaction.
run( query );
      transaction.success(); // mark success,
actually commit will happen in
transaction.close()
      String greeting = result.single().get( 0
).asString();
      System.out.println( greeting );
   }
}
```

JavaScript driver

This section presents the breaking changes between the Neo4j 1.7 JavaScript driver and 4.x JavaScript driver.

The latest version of the JavaScript driver for Neo4j can be found on the JavaScript driver's official page.

- `session#close()` and `driver#close()` now return `Promises`, and no longer accept callback function arguments.
- `driver.onError` and `driver.onCompleted` callbacks have been removed. Errors should be monitored on related code paths (e.g. through `Promise#catch`, etc.).

See full changelog: https://github.com/neo4j/neo4j-javascript-driver/wiki/4.0-changelog.

Example 12. Example of changes between the 1.7 JavaScript driver and the 4.0 JavaScript driver

Example code for the 4.0 JavaScript driver

```
var neo4j = require('neo4j-driver')
...
const driver = neo4j.driver(uri, neo4j.auth
.basic(user, password))
...

const session = driver.session()
try {
  const tx = session.beginTransaction()
  const result = await tx.run('CREATE
(a:Greeting) SET a.message = $message RETURN
a.message + ", from node " + id(a)', { message:
'hello, world' })
  const greeting = result.records[0].get(0)
  console.log(greeting)
  await tx.commit()
} finally {
  await session.close()
}
```

Example code for the 1.7 JavaScript driver

```
var neo4j = require('neo4j-driver').v1
...
const driver = neo4j.driver(uri, neo4j.auth
.basic(user, password))
...

const session = driver.session()
try {
  const tx = session.beginTransaction()
  const result = await tx.run('CREATE
(a:Greeting) SET a.message = $message RETURN
a.message + ", from node " + id(a)', { message:
'hello, world' })
  const greeting = result.records[0].get(0)
  console.log(greeting)
  await tx.commit()
} finally {
  session.close(callback) // another session
can be chained in callback
}
```

.NET driver

This section presents the breaking changes between the Neo4j 1.7 .NET driver and 4.x .NET driver.

The latest version of the .NET driver for Neo4j can be found on the .NET driver's official page.

- The `Neo4j.Driver` package contains only the asynchronous API.
  - Synchronous session API (Simple API) has been moved to the Neo4j.Driver.Simple package.
  - Reactive API is presented in the Neo4j.Driver.Reactive package.
- The `IDriverLogger` has been renamed to `ILogger`.
- `TrustStrategy` is replaced with `TrustManager`.

See full changelog: https://github.com/neo4j/neo4j-dotnet-driver/wiki/4.0-changelog.

Example 13. Example of changes between the 1.7 and 4.0 .NET drivers

Example code for the 4.0 .NET driver

```
using Neo4j.Driver.Simple;
...
private readonly IDriver _driver;
private readonly string
_previousNeo4jSessionBookmark;
...
public void PrintGreeting(string message)
{
   using (ISession session = _driver.Session(o
=>
      o.WithDatabase("neo4j")
       .WithDefaultAccessMode(AccessMode.Write)
       .WithBookmarks
(_previousNeo4jSessionBookmark)))
   {
       using (ITransaction transaction =
session.BeginTransaction())
       {
           Query query = new Query("CREATE
(a:Greeting) SET a.message = $message RETURN
a.message + ', from node ' + id(a)", new
Dictionary<string, object>{{"message",
message}});
           IResult result = transaction.Run
(query);

                string greeting = result.
Single()[0].As<string>();
                Console.WriteLine(greeting);
                transaction.Commit(); // commit
immediately here
            }
         _previousNeo4jSessionBookmark =
session.LastBookmark;
        }
    }
```

Example code for the 1.7 .NET driver

```
using Neo4j.Driver;
...
private readonly IDriver _driver;
private readonly string
_previousSessionBookmark;
...
public void PrintGreeting(string message)
{
   using (ISession session =_driver.Session(
      AccessMode.Write,
_previousSessionBookmark))
   {


      using (ITransaction transaction =
session.BeginTransaction())
      {
         Statement query = new Statement
("CREATE (a:Greeting) SET a.message = $message
RETURN a.message + ', from node ' + id(a)", new
Dictionary<string, object>{{"message",
message}});
          IStatementResult result =
transaction.Run(query);
          transaction.Success(); // mark
success, actually commit will happen in
transaction.Dispose()
          var greeting = result.Single()[0].As
<string>();
          Console.WriteLine(greeting);
   }
   _previousSessionBookmark = session
.LastBookmark;
   }
}
```

Python driver

This section presents the breaking changes between the Neo4j 1.7 Python driver and 4.x Python driver.

The latest version of the Python driver for Neo4j can be found on the Python driver's official page.

1.7	Change	4.x
`max_retry_time`	Renamed configuration	`max_transaction_retry_time`
`access_mode`	Renamed configuration and it is now a keyword argument	`default_access_mode`
`neo4j.exceptions.CypherError`	Renamed exception	`neo4j.exceptions.Neo4jError`
`neo4j.exceptions.ConnectionExpired`	Removed exception	-
`transaction.success`	Removed flag	-

1.7	Change	4.x
`Result.value(item=0, default=None)`	`neo4j.Record. helper function has changed parameter name from `item` to `key`.	`Result.value(key=0, default=None)`
`Result.values(*items)`	`neo4j.Record` helper function has changed parameter name from `items` to `keys`.	`Result.values(*keys)`
`Result.data(*items)`	`neo4j.Record` helper function has changed parameter name from `items` to `keys`.	`Result.data(*keys)`
neobolt	No longer a dependency	-
neotime	No longer a dependency	-
	Is now a dependency	pytz
`Transaction.sync()`	Removed	Use `Result.consume()` if the behavior is to exhaust the result object.
`Transaction.success`	Removed	-
`Transaction.close()`	Change in behavior, will only perform rollback if no commit has been performed.	`Transaction.close()`
Session.sync()`	Removed	Use `Result.consume()` if the behavior is to exhaust the result object.
`Session.detach()`	Removed	Use `Result.consume()` if the behavior is to exhaust the result object.
`Session.next_bookmarks()`	Removed	-
`Session.has_transaction()`	Removed	-
`Session.closed()`	Removed	-

See full changelog: https://github.com/neo4j/neo4j-python-driver/wiki/4.0-changelog.

Example 14. Example of changes between the 1.7 Python driver and the 4.0 Python driver

Example code for the 4.0 Python driver

```
driver = GraphDatabase.driver(
    uri, auth=(user, password),
    max_transaction_retry_time=120
)
def create_and_return_greeting(tx,
                               message):
    result = tx.run(
        "CREATE (a:Greeting) "
        "SET a.message = $message "
        "RETURN a.message + "
        "', from node ' + id(a)",
        message=message
    )
    return result.single()[0]
with driver.session() as session:
    greeting = session.write_transaction(
        create_and_return_greeting,
        "Hello World"
    )
    print(greeting)
```

Example code for the 1.7 Python driver

```
driver = GraphDatabase.driver(
    uri, auth=(user, password),
    max_retry_time=120
)
def create_and_return_greeting(tx,
                               message):
    result = tx.run(
        "CREATE (a:Greeting) "
        "SET a.message = $message "
        "RETURN a.message + "
        "', from node ' + id(a)",
        message=message
    )
    return result.single()[0]
with driver.session() as session:
    greeting = session.write_transaction(
        create_and_return_greeting,
        "Hello World"
    )
    print(greeting)
```

7.9.3. Migration checklist

This topic is about preparation steps for a Neo4j 3.5 DBMS migration to a Neo4j 4.0 DBMS.

Before you start preparing for migration, it is very important to read the information in Understanding upgrades and migration and Supported upgrade and migration paths.

Follow the checklist to prepare for migrating your Neo4j deployment:

- ☐ Complete all prerequisites for the migration.
- ☐ Reserve enough disk space for the pre-migration backup and the migration.
- ☐ Shut down the Neo4j DBMS that you want to migrate.
- ☐ Back up your current deployment to avoid losing data in case of a failure.
- ☐ Download the new version of Neo4j. Make sure the migration path is supported.
- ☐ Prepare a new *neo4j.conf* file to be used by the new installation.
- ☐ Migrate a single instance (offline) or Migrate a Causal Cluster (offline).
- ☐ Monitor the logs.
- ☐ Perform the migration.

If you are migrating a Causal Cluster, complete the checklist for each cluster member.

Prerequisites

1. Verify that you have installed Java 11.
2. Review the new features and improvements that have been carried out in the new version, as well as the breaking changes between 3 and 4 `MAJOR` versions.
    - Neo4j changelog
    - Release Notes
    - Breaking changes between Neo4j 3.5 and Neo4j 4.x
    - (If using drivers) Breaking changes between Neo4j 1.7 drivers and Neo4j 4.x drivers
    - Neo4j indexes
    - Changes to configuration settings in Neo4j 4
    - Changes to procedures in Neo4j 4

Reserve enough disk space

A migration requires substantial free disk space, as it makes a complete copy of the database. Therefore, it is essential to make available an additional 50% * the size of your *database directory*. In a default configuration, the *database directory* is *databases/neo4j*, which is located in the data directory. The migrated database may require slightly larger data files overall.
You must also reserve the disk space needed for the pre-migration backup.

Shut down the Neo4j DBMS

Because a migration requires the database to be offline, the first step is to cleanly shut down the Neo4j DBMS if it is still running:

```
bin/neo4j stop
```

or

```
systemctl stop neo4j
```

It is important to verify that the database shutdown process has finished successfully, and the database was cleanly shutdown. You can check the *neo4j.log* for these log messages for confirmation:

A sample log

```
2022-02-25 13:44:45.826+0000 INFO  Neo4j Server shutdown initiated by request
2022-02-25 13:44:45.830+0000 INFO  Stopped.
```

Back up your current deployment

You need to back up:

- The *neo4j.conf* file.

- All the files used for encryption, such as private key, public certificate, and the contents of the trusted and revoked directories (located in *<neo4j-home>/certificates/*).
- The contents of *<neo4j-home>/data/dbms* (if there are any native users).
- The Neo4j *data store*. If you are running a Debian/RPM distribution, you can skip this step.

ℹ	For clusters deployments, you need to back up the contents of the */dbms* folder and Neo4j *data store* only from **one** instance, which will be your elected instance.

ℹ	(Direct path only) Backing up your existing 3.5.x store is *optional*. The direct path does not replace the current store but rather makes a copy of it and migrates it at the same time. Because of this, you will always have an available backup in case of disaster.

Because the database is now offline, you use the `neo4j-admin dump` command to create an offline backup (`.dump` file) of your data store:

```
$NEO4J_HOME/bin/neo4j-admin dump --to=$BACKUP_DESTINATION
```

The result is a file called *<db_name>.dump* located in `$BACKUP_DESTINATION` you have defined. This is the backup you are going to use for the migration. For more information about the `neo4j-admin dump` command, see Operations Manual → Back up an offline database.

Prepare a new *neo4j.conf* file to be used by the new installation

A Neo4j migration requires changes to the configuration. Therefore, you have to prepare a new *neo4j.conf* file to be used by the new deployment. If you are migrating a Causal Cluster, do this for each cluster member.

- Update the new *neo4j.conf* file with any non-default settings from your old installation.
- Note any custom values of the settings `dbms.directories.*` and `dbms.active_database`.
- In cluster installations, pay attention to cluster-specific configuration settings, which might be different for the different cluster members.

ℹ	(Sequential path only) The migration of users and roles from 3.5 to 4.0 is done automatically. Therefore, you do not have to move the *data/dbms/* directory and contents to the new installation. The files in 3.5 will be parsed and the content added to the `system` database on the first startup of the Neo4j 4.0 DBMS.

Perform a test migration

Based on the findings in this chapter, allocate a staging test environment for the migration and do a test migration. The test migration will give you valuable information about the time required for the production migration. Follow the steps as per your Neo4j version and deployment type, see (Migrate a single instance (offline) or Migrate a Causal Cluster (offline)).

Monitor the logs

The *neo4j.log* file contains information on how many steps the migration involves and how far it has progressed. For large migrations, it is a good idea to monitor this log continuously.

A sample log

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

For more information, see Operations Manual → Logging.

Migrate a single instance

This chapter describes the necessary steps to migrate a single instance from Neo4j 3.5 to 4.0.

To migrate from 3.5.latest to a version beyond 4.0, the single instance must first be migrated to 4.0 and thereafter upgraded to the desired version.

Prerequisites

Ensure that you have completed all tasks on the Migration checklist.

Prepare for the migration

1. Verify that you have shut down the Neo4j DBMS. You can check the *neo4j.log*.
2. Install the Neo4j version that you want to migrate to. For more information on how to install the distribution that you are using, see Operations Manual → Installation.
3. Replace the *neo4j.conf* file with the one that you have prepared in section Prepare a new neo4j.conf file to be used by the new installation.
4. Set `dbms.allow_upgrade=true` to allow automatic store upgrade.
5. Copy all the files used for encryption, such as private key, public certificate, and the contents of the

trusted and revoked directories (located in *<neo4j-home>/certificates/*).

> If your old installation has modified configurations starting with `dbms.directories.*` or the setting `dbms.active_database`, verify that the new `neo4j.conf` file is configured properly to find these directories.

> The migration of users and roles from 3.5 to 4.0 is done automatically. Therefore, you do not have to move the *data/dbms/* directory and contents to the new installation. The files in 3.5 will be parsed and the content added to the `system` database on the first startup of the Neo4j 4.0 DBMS.

6. If you are using custom plugins, make sure they are updated and compatible with the new version, and place them in the */plugins* directory.

Migrate the data and start the server

Before migrating the data, you have to move the backup file to the */data* directory of Neo4j 4.0.

> This step is not applicable if you have *dbms.directories.data* pointing to a directory outside of <neo4j-home>.

1. Move the backup file into the new installation by running the `neo4j-admin load` command from `<neo4j-home>`:

   > It is a requirement to use this command as file system copy-and-paste of databases is not supported and may result in unwanted behaviour. If you are running a Debian/RPM distribution, you can skip this step.

   ```
   bin/neo4j-admin load --from=$BACKUP_DESTINATION/<db_name>.dump -database=<db_name> --force
   ```

2. With the backup in-place, initiate the data migration by starting Neo4j 4.0:

   ```
   bin/neo4j start
   ```

   or

   ```
   systemctl start neo4j
   ```

   The database migration takes place during startup. Your indexes are also automatically migrated to the most recent index provider during startup. However, indexes require populating after migration. Your queries will not be able to use the indexes while they are populating so you may see a temporary performance hit. If your queries are 100% dependent on some indexes, it is advisable to account for index population as part of the duration of the migration.

The neo4j.log file contains valuable information on how many steps the migration involves and how far it has progressed. Index populations are tracked on the *debug.log*. For large migrations, it is a good idea to monitor these logs continuously.

3. If you used a different database name than what is set as the default database in *neo4j.conf*, you need to activate that database before being able to use it. You can do this with the `CREATE DATABASE` command in the Browser or Cypher Shell:

   ```
   CREATE DATABASE name
   ```

Post-migration tasks

1. When the upgrade finishes, open the *neo4j.conf* file and set `dbms.allow_upgrade=false`.
2. Restart the instance.
3. It is good practice to make a full backup immediately after the migration.

Migrate a Causal Cluster to 4.0

This chapter describes the necessary steps to migrate a Causal Cluster from Neo4j 3.5 to 4.0.

The migration of a Causal Cluster from Neo4j 3.5 to 4.0 requires downtime. Therefore, it is recommended to perform a test migration in a production-like environment to get information on the duration of the downtime.

To migrate from 3.5.latest to a version beyond 4.0, the cluster must first be migrated to 4.0 and thereafter upgraded to the desired version. For more information, see Supported upgrade and migration paths.

The prerequisites and the migration steps must be completed for each cluster member.

Prerequisites

Ensure that you have completed all tasks on the Migration checklist.

Prepare for the migration

The strategy for migrating a cluster deployment is to complete a single migration on a single instance, as a standalone instance, and then use the migrated store to seed the remaining members of the cluster.

Remember, migration is a single event. Do not perform independent migrations on each of your instances! There should be a single migration event and that migrated store will be your source of truth for all the other instances of the cluster. This is important because when migrating, Neo4j generates random store IDs and, if done independently, your cluster will end up with as many store IDs as instances you have. Neo4j will fail to start if that is the case. Due to this, some of the cluster migrations steps will be performed on a single instance while others will be performed on all instances. Each step will tell you where to perform the necessary actions.

At this stage, you should elect one instance to work on. This will be the instance where the migration will actually happen. The next steps will tell you whether to perform the step on the elected instance, on the remaining instances or on all instances.

On each cluster member

1. Verify that you have shut down all cluster members (Cores and Read Replicas). You can check the *neo4j.log*.
2. Perform `neo4j-admin unbind` on each cluster member to remove cluster state data.
3. Install the *Neo4j* version that you want to migrate to on each instance. For more information on how to install the distribution that you are using, see Operations Manual 4.0 → Installation.
4. Replace the *neo4j.conf* file with the one that you have prepared for each instance in section Prepare a new *neo4j.conf* file to be used by the new installation.
5. Copy all the files used for encryption, such as private key, public certificate, and the contents of the trusted and revoked directories (located in *<neo4j-home>/certificates/*).
6. If you are using custom plugins, make sure they are updated and compatible with the new version, and place them in the */plugins* directory.

On the elected instance

1. Open the *neo4j.conf* file of the new installation and configure the following settings:
   - Uncomment `dbms.allow_upgrade=true` to allow automatic store migration. Neo4j will fail to start without this configuration.
   - Set `dbms.mode=SINGLE`. You need to do this because a migration is a single event that needs to happen on a standalone server.

Migrate the data

On the elected instance

Before migrating the data, you need to move the backup file to the *data* directory of Neo4j 4.0.

This step is not applicable if you have *dbms.directories.data* pointing to a directory outside of <neo4j-home>.

1. Move the backup file into the new installation by running the `neo4j-admin load` command from `<neo4j-home>`:

It is a requirement to use this command as file system copy-and-paste of databases is not supported and may result in unwanted behaviour. If you are running a Debian/RPM distribution, you can skip this step.

```
$NEO4J_HOME/bin/neo4j-admin load --from=$BACKUP_DESTINATION/<db_name>.dump -database=<db_name> --force
```

The migration of users and roles from 3.5 to 4.0 is done automatically. Therefore, you do not have to move the *data/dbms/* directory and contents to the new installation. The files in 3.5 will be parsed and the content added to the `system` database on the first startup of the Neo4j 4.0 DBMS.

2. With the backup in-place, initiate the migration by starting the elected instance:

   ```
   bin/neo4j start
   ```

   or

   ```
   systemctl start neo4j
   ```

   The migration takes place during startup. Your indexes are also automatically migrated to the most recent index provider during startup. However, indexes require populating after migration. Your queries will not be able to use the indexes while they are populating so you may see a temporary performance hit. If your queries are 100% dependent on some indexes, it is advisable to account for index population as part of the duration of the migration.

   The neo4j.log file contains valuable information on how many steps the migration involves and how far it has progressed. Index populations are tracked on the *debug.log*. For large migrations, it is a good idea to monitor these logs continuously.

3. When the migration finishes, stop the server.

   ```
   bin/neo4j stop
   ```

   or

   ```
   systemctl stop neo4j
   ```

Prepare for seeding the cluster

On the elected instance

1. Revert *neo4j.conf* changes:
   - Set `dbms.allow_upgrade=false`.
   - Set `dbms.mode=CORE` to re-enable Causal Clustering in the configuration.

2. Take an offline backup of your newly migrated database and transactions, alongside with the `system` database using `neo4j-admin dump`. This backup will be used to seed the remaining instances of the cluster.

   ```
   bin/neo4j-admin dump --database=<db_name> --to=$BACKUP_DESTINATION/<db_name>.dump

   bin/neo4j-admin dump --database=system --to=$BACKUP_DESTINATION/system.dump
   ```

   Be aware that after you migrate, Neo4j Admin commands can differ slightly because Neo4j now supports multiple databases.

3. Do **not** yet start the server.

Seed the cluster

On each of the remaining instances

1. Copy the dumps created in Migration steps to the remaining instances.
2. Once this is complete, use `neo4j-admin load --from=<archive-path> --database=<db_name> --force` to replace each of your databases, including the `system` database, with the ones migrated on the elected instance.

   ```
   bin/neo4j-admin load --from=$BACKUP_DESTINATION/<db_name>.dump --database=<db_name> --force

   bin/neo4j-admin load --from=$BACKUP_DESTINATION/system.dump --database=system --force
   ```

Start the cluster

On each cluster member, including the elected instance

Before continuing, make sure the following activities happened and were completed successfully:

- Content of *neo4j.conf* is correct and required changes were applied on all instances.
- Single migration event occurred on elected instance.
- Backup (via `neo4j-admin dump`) of migrated store performed on the elected instance.
- Backup of the migrated store was transferred to the remaining instances.
- Store was loaded on the remaining instances (via `neo4j-admin load`).
- `dbms.mode=CORE` and `dbms.allow_upgrade=false` (or commented) are set on *neo4j.conf* of the elected instance.

1. If everything on the list was successful, you can go ahead and start all instances of the cluster.

   ```
   bin/neo4j start
   ```

or

```
systemctl start neo4j
```

2. If the migrated database is the `default` database, it should have been started automatically on instance startup and this step is not required. If the migrated database is not the `default` database, it is still in the `STOPPED` state. You now need to start the database. On one of the cluster members, run the following command in Neo4j Browser or Cypher Shell:

   ```
   CREATE DATABASE <db_name>;
   ```

   For each Read Replica

3. Start the Read Replica and wait for it to catch up with the rest of the cluster members.

   (Optional) While an empty read replica will eventually get a full copy of all data from the other members of your cluster, catching up may take some time. To speed up the process, you can load the data first by using `neo4j-admin load --from=<archive -path> --database=<db_name> --force` to replace each of your databases with the migrated one.

4. Verify that the Read Replicas have joined the cluster.

Post-migration

It is recommended to perform a *full backup*, using an empty target directory.

Migrate a single instance

This chapter describes the necessary steps to migrate a single instance from Neo4j 3.5 directly to 4.x.

The migration of a single instance from Neo4j 3.5 to 4.x requires downtime. Therefore, it is recommended to perform a test migration in a production-like environment to get information on the duration of the downtime.

Prerequisites

Ensure that you have completed all tasks on the Migration checklist.

Prepare for the migration

1. Verify that you have shut down the Neo4j DBMS. You can check the *neo4j.log*.

2. Install the Neo4j version that you want to migrate to. For more information on how to install the distribution that you are using, see Operations Manual → Installation.

3. Replace the *neo4j.conf* file with the one that you have prepared in section Prepare a new *neo4j.conf* file to be used by the new installation.

4. Copy all the files used for encryption, such as private key, public certificate, and the contents of the trusted and revoked directories (located in *<neo4j-home>/certificates/*).

   > If your old installation has modified configurations starting with `dbms.directories.*` or the setting `dbms.active_database`, verify that the new `neo4j.conf` file is configured properly to find these directories.

5. If you are using custom plugins, make sure they are updated and compatible with the new version, and place them in the */plugins* directory.

Migrate the data

Using the 4.0 Neo4j Admin tool, migrate the data store of your 3.5 Neo4j. The `neo4j-admin copy` command also removes any inconsistent nodes, properties, and relationships and does not copy them to the newly created store.

1. From the *<neo4j-home>* folder, run the following command to copy the data store. You need to specify the old store location and the name for the target updated database:

```
bin/neo4j-admin copy --from-path=/path/to/3.5.x/graph.db --to-database=<db_name>
```

```
Starting to copy store, output will be saved to:  $neo4j_home/logs/neo4j-admin-copy-2020-11-
26.16.07.19.log
2020-11-26 16:07:19.939+0000 INFO [StoreCopy] ### Copy Data ###
2020-11-26 16:07:19.940+0000 INFO [StoreCopy] Source: /path/to/3.5.x/graph.db (page cache 8m)
2020-11-26 16:07:19.940+0000 INFO [StoreCopy] Target:  $neo4j_home/data/databases/database (page cache
8m)
2020-11-26 16:07:19.940+0000 INFO [StoreCopy] Empty database created, will start importing readable
data from the source.
2020-11-26 16:07:21.661+0000 INFO [o.n.i.b.ImportLogic] Import starting

Import starting 2020-11-26 16:07:21.699+0000
  Estimated number of nodes: 50.00 k
  Estimated number of node properties: 50.00 k
  Estimated number of relationships: 0.00
  Estimated number of relationship properties: 50.00 k
  Estimated disk space usage: 2.680MiB
  Estimated required memory usage: 8.598MiB

(1/4) Node import 2020-11-26 16:07:22.220+0000
  Estimated number of nodes: 50.00 k
  Estimated disk space usage: 1.698MiB
  Estimated required memory usage: 8.598MiB
.......... .......... .......... .......... ..........   5% ∆239ms
.......... .......... .......... .......... ..........  10% ∆1ms
.......... .......... .......... .......... ..........  15% ∆1ms
.......... .......... .......... .......... ..........  20% ∆0ms
.......... .......... .......... .......... ..........  25% ∆1ms
.......... .......... .......... .......... ..........  30% ∆0ms
.......... .......... .......... .......... ..........  35% ∆0ms
.......... .......... .......... .......... ..........  40% ∆1ms
.......... .......... .......... .......... ..........  45% ∆0ms
.......... .......... .......... .......... ..........  50% ∆1ms
.......... .......... .......... .......... ..........  55% ∆0ms
```

```
.......... .......... .......... .......... .........-  60% ∆51ms
.......... .......... .......... .......... ..........  65% ∆0ms
.......... .......... .......... .......... ..........  70% ∆0ms
.......... .......... .......... .......... ..........  75% ∆1ms
.......... .......... .......... .......... ..........  80% ∆0ms
.......... .......... .......... .......... ..........  85% ∆0ms
.......... .......... .......... .......... ..........  90% ∆1ms
.......... .......... .......... .......... ..........  95% ∆0ms
.......... .......... .......... .......... .......... 100% ∆0ms

(2/4) Relationship import 2020-11-26 16:07:22.543+0000
  Estimated number of relationships: 0.00
  Estimated disk space usage: 1006KiB
  Estimated required memory usage: 15.60MiB
(3/4) Relationship linking 2020-11-26 16:07:22.879+0000
  Estimated required memory usage: 7.969MiB
(4/4) Post processing 2020-11-26 16:07:23.272+0000
  Estimated required memory usage: 7.969MiB
-......... .......... .......... .......... ..........   5% ∆356ms
.......... .......... .......... .......... ..........  10% ∆0ms
.......... .......... .......... .......... ..........  15% ∆1ms
.......... .......... .......... .......... ..........  20% ∆0ms
.......... .......... .......... .......... ..........  25% ∆0ms
.......... .......... .......... .......... ..........  30% ∆1ms
.......... .......... .......... .......... ..........  35% ∆0ms
.......... .......... .......... .......... ..........  40% ∆0ms
.......... .......... .......... .......... ..........  45% ∆1ms
.......... .......... .......... .......... ..........  50% ∆0ms
.......... .......... .......... .......... ..........  55% ∆0ms
.......... .......... .......... .......... ..........  60% ∆0ms
.......... .......... .......... .......... ..........  65% ∆1ms
.......... .......... .......... .......... ..........  70% ∆0ms
.......... .......... .......... .......... ..........  75% ∆0ms
.......... .......... .......... .......... ..........  80% ∆0ms
.......... .......... .......... .......... ..........  85% ∆0ms
.......... .......... .......... .......... ..........  90% ∆0ms
.......... .......... .......... .......... ..........  95% ∆1ms
.......... .......... .......... .......... .......... 100% ∆0ms


IMPORT DONE in 2s 473ms.
Imported:
  1 nodes
  0 relationships
  1 properties
Peak memory usage: 15.60MiB
2020-11-26 16:07:24.140+0000 INFO [o.n.i.b.ImportLogic] Import completed successfully, took 2s 473ms.
Imported:
  1 nodes
  0 relationships
  1 properties
2020-11-26 16:07:24.668+0000 INFO [StoreCopy] Import summary: Copying of 100704 records took 4 seconds
(25176 rec/s). Unused Records 100703 (99%) Removed Records 0 (0%)
2020-11-26 16:07:24.669+0000 INFO [StoreCopy] ### Extracting schema ###
2020-11-26 16:07:24.669+0000 INFO [StoreCopy] Trying to extract schema...
2020-11-26 16:07:24.920+0000 INFO [StoreCopy] ... found 1 schema definitions. The following can be
used to recreate the schema:
2020-11-26 16:07:24.922+0000 INFO [StoreCopy]

CALL db.createIndex('index_5c0607ad', ['Person'], ['name'], 'native-btree-1.0', {`spatial.cartesian-
3d.min`: [-1000000.0, -1000000.0, -1000000.0],`spatial.cartesian.min`: [-1000000.0,
-1000000.0],`spatial.wgs-84.min`: [-180.0, -90.0],`spatial.cartesian-3d.max`: [1000000.0, 1000000.0,
1000000.0],`spatial.cartesian.max`: [1000000.0, 1000000.0],`spatial.wgs-84-3d.min`: [-180.0, -90.0,
-1000000.0],`spatial.wgs-84-3d.max`: [180.0, 90.0, 1000000.0],`spatial.wgs-84.max`: [180.0, 90.0]})
2020-11-26 16:07:24.923+0000 INFO [StoreCopy] You have to manually apply the above commands to the
database when it is stared to recreate the indexes and constraints. The commands are saved to
$neo4j_home/logs/neo4j-admin-copy-2020-11-26.16.07.19.log as well for reference.
```

When using the direct path, indexes are not automatically migrated so you have to recreate them. After running the store migration, the `neo4j-admin copy` command extracts the schema and generates a list of commands you can use to recreate your schema on the new 4.x store. The recreate schema commands are also saved in the migration log file, located in the /logs directory.

2. Start the server.

```
bin/neo4j start
```

or

```
systemctl start neo4j
```

Copying a database does not automatically create it. Therefore, it will not be visible if you do `SHOW DATABASES` in Cypher Shell or Neo4j Browser.

3. Log in to the Cypher Shell command-line console, change the active database to `system` (`:USE system;`), and create the `<db_name>` database. For more information about the Cypher Shell command-line interface (CLI) and how to use it, see Operations Manual → Cypher Shell.

```
CREATE DATABASE <db_name>;
```

4. Using the new database (`:USE <db_name>;`), recreate any indexes or constraints that were identified by the `neo4j-admin copy` command.

```
CALL db.createIndex('index_5c0607ad', ['Person'], ['name'], 'native-btree-1.0', {`spatial.cartesian-
3d.min`: [-1000000.0, -1000000.0, -1000000.0],`spatial.cartesian.min`: [-1000000.0, -1000000.0
],`spatial.wgs-84.min`: [-180.0, -90.0],`spatial.cartesian-3d.max`: [1000000.0, 1000000.0, 1000000.0
],`spatial.cartesian.max`: [1000000.0, 1000000.0],`spatial.wgs-84-3d.min`: [-180.0, -90.0, -
1000000.0],`spatial.wgs-84-3d.max`: [180.0, 90.0, 1000000.0],`spatial.wgs-84.max`: [180.0, 90.0]});
```

Post-migration tasks

It is good practice to make a full backup immediately after the migration.

Migrate a Causal Cluster

This chapter describes the necessary steps to migrate a Causal Cluster from Neo4j 3.5 directly to 4.x.

The migration of a Causal Cluster from Neo4j 3.5 to 4.x requires downtime. Therefore, it is recommended to perform a test migration in a production-like environment to get information on the duration of the downtime.

The prerequisites and the migration steps must be completed for each cluster member.

Prerequisites

Ensure that you have completed all tasks on the Migration checklist.

Prepare for the migration

The strategy for migrating a cluster deployment is to complete an offline copy from one cluster instance, and then use the copied store to seed the new cluster.

Remember, a migration is a single event. Do not perform independent migrations on each of your instances! There should be a single migration event and that migrated store will be your source of truth for all the other instances of the cluster. This is important because when migrating, Neo4j generate random store IDs and, if done independently, your cluster will end up with as many store IDs as instances you have. Neo4j will fail to start if that is the case. Due to this, some of the cluster migrations steps will be performed on a single instance while others will be performed on all instances. Each step will tell you where to perform the necessary actions.

At this stage, you should elect one instance to work on. This will be the instance where the migration will actually happen. The next steps will tell you whether to perform the step on the elected instance, on the remaining instances or on all instances.

On each cluster member

1. Verify that you have shut down all the cluster members (Cores and Read Replicas). You can check the *neo4j.log*.
2. Install the *Neo4j* version that you want to migrate to on each instance. For more information on how to install the distribution that you are using, see Operations Manual → Installation.
3. Replace the *neo4j.conf* file with the one that you have prepared for each instance in section Prepare a new *neo4j.conf* file to be used by the new installation.
4. Copy all the files used for encryption, such as private key, public certificate, and the contents of the trusted and revoked directories (located in *<neo4j-home>/certificates/*).

   If your old installation has modified configurations starting with `dbms.directories.*` or the setting `dbms.active_database`, verify that the new `neo4j.conf` file is configured properly to find these directories.

Migrate the data

On the elected instance

Using the 4.x Neo4j Admin tool, migrate the data store of your 3.5 Neo4j. The `neo4j-admin copy` command also removes any inconsistent nodes, properties, and relationships and does not copy them to the newly created store.

1. From the <*neo4j-home*> folder, run the following command to copy the data store. You need to specify the old store location and the name for the target updated database:

```
bin/neo4j-admin copy --from-path=/path/to/3.5.x/graph.db --to-database=<db_name>
```

```
Starting to copy store, output will be saved to:  $neo4j_home/logs/neo4j-admin-copy-2020-11-
26.16.07.19.log
2020-11-26 16:07:19.939+0000 INFO [StoreCopy] ### Copy Data ###
2020-11-26 16:07:19.940+0000 INFO [StoreCopy] Source: /path/to/3.5.x/graph.db (page cache 8m)
2020-11-26 16:07:19.940+0000 INFO [StoreCopy] Target:  $neo4j_home/data/databases/db_name (page
cache 8m)
2020-11-26 16:07:19.940+0000 INFO [StoreCopy] Empty database created, will start importing readable
data from the source.
2020-11-26 16:07:21.661+0000 INFO [o.n.i.b.ImportLogic] Import starting

Import starting 2020-11-26 16:07:21.699+0000
  Estimated number of nodes: 50.00 k
  Estimated number of node properties: 50.00 k
  Estimated number of relationships: 0.00
  Estimated number of relationship properties: 50.00 k
  Estimated disk space usage: 2.680MiB
  Estimated required memory usage: 8.598MiB

(1/4) Node import 2020-11-26 16:07:22.220+0000
  Estimated number of nodes: 50.00 k
  Estimated disk space usage: 1.698MiB
  Estimated required memory usage: 8.598MiB
.......... .......... .......... .......... ..........   5% ∆239ms
.......... .......... .......... .......... ..........  10% ∆1ms
.......... .......... .......... .......... ..........  15% ∆1ms
.......... .......... .......... .......... ..........  20% ∆0ms
.......... .......... .......... .......... ..........  25% ∆1ms
.......... .......... .......... .......... ..........  30% ∆0ms
.......... .......... .......... .......... ..........  35% ∆0ms
.......... .......... .......... .......... ..........  40% ∆1ms
.......... .......... .......... .......... ..........  45% ∆0ms
.......... .......... .......... .......... ..........  50% ∆1ms
.......... .......... .......... .......... ..........  55% ∆0ms
.......... .......... .......... .......... .........-  60% ∆51ms
.......... .......... .......... .......... ..........  65% ∆0ms
.......... .......... .......... .......... ..........  70% ∆0ms
.......... .......... .......... .......... ..........  75% ∆1ms
.......... .......... .......... .......... ..........  80% ∆0ms
.......... .......... .......... .......... ..........  85% ∆0ms
.......... .......... .......... .......... ..........  90% ∆1ms
.......... .......... .......... .......... ..........  95% ∆0ms
.......... .......... .......... .......... .......... 100% ∆0ms

(2/4) Relationship import 2020-11-26 16:07:22.543+0000
  Estimated number of relationships: 0.00
  Estimated disk space usage: 1006KiB
  Estimated required memory usage: 15.60MiB
(3/4) Relationship linking 2020-11-26 16:07:22.879+0000
  Estimated required memory usage: 7.969MiB
(4/4) Post processing 2020-11-26 16:07:23.272+0000
  Estimated required memory usage: 7.969MiB
-......... .......... .......... .......... ..........   5% ∆356ms
.......... .......... .......... .......... ..........  10% ∆0ms
.......... .......... .......... .......... ..........  15% ∆1ms
.......... .......... .......... .......... ..........  20% ∆0ms
.......... .......... .......... .......... ..........  25% ∆0ms
.......... .......... .......... .......... ..........  30% ∆1ms
.......... .......... .......... .......... ..........  35% ∆0ms
.......... .......... .......... .......... ..........  40% ∆0ms
.......... .......... .......... .......... ..........  45% ∆1ms
.......... .......... .......... .......... ..........  50% ∆0ms
.......... .......... .......... .......... ..........  55% ∆0ms
.......... .......... .......... .......... ..........  60% ∆0ms
```

```
.......... .......... .......... .......... ..........  65% ∆1ms
.......... .......... .......... .......... ..........  70% ∆0ms
.......... .......... .......... .......... ..........  75% ∆0ms
.......... .......... .......... .......... ..........  80% ∆0ms
.......... .......... .......... .......... ..........  85% ∆0ms
.......... .......... .......... .......... ..........  90% ∆0ms
.......... .......... .......... .......... ..........  95% ∆1ms
.......... .......... .......... .......... .......... 100% ∆0ms


IMPORT DONE in 2s 473ms.
Imported:
  1 nodes
  0 relationships
  1 properties
Peak memory usage: 15.60MiB
2020-11-26 16:07:24.140+0000 INFO [o.n.i.b.ImportLogic] Import completed successfully, took 2s
473ms. Imported:
  1 nodes
  0 relationships
  1 properties
2020-11-26 16:07:24.668+0000 INFO [StoreCopy] Import summary: Copying of 100704 records took 4
seconds (25176 rec/s). Unused Records 100703 (99%) Removed Records 0 (0%)
2020-11-26 16:07:24.669+0000 INFO [StoreCopy] ### Extracting schema ###
2020-11-26 16:07:24.669+0000 INFO [StoreCopy] Trying to extract schema...
2020-11-26 16:07:24.920+0000 INFO [StoreCopy] ... found 1 schema definitions. The following can be
used to recreate the schema:
2020-11-26 16:07:24.922+0000 INFO [StoreCopy]

CALL db.createIndex('index_5c0607ad', ['Person'], ['name'], 'native-btree-1.0',
{`spatial.cartesian-3d.min`: [-1000000.0, -1000000.0, -1000000.0],`spatial.cartesian.min`: [-
1000000.0, -1000000.0],`spatial.wgs-84.min`: [-180.0, -90.0],`spatial.cartesian-3d.max`:
[1000000.0, 1000000.0, 1000000.0],`spatial.cartesian.max`: [1000000.0, 1000000.0],`spatial.wgs-84-
3d.min`: [-180.0, -90.0, -1000000.0],`spatial.wgs-84-3d.max`: [180.0, 90.0,
1000000.0],`spatial.wgs-84.max`: [180.0, 90.0]})
2020-11-26 16:07:24.923+0000 INFO [StoreCopy] You have to manually apply the above commands to the
database when it is stared to recreate the indexes and constraints. The commands are saved to
$neo4j_home/logs/neo4j-admin-copy-2020-11-26.16.07.19.log as well for reference.
```

When using the direct path, indexes are not automatically migrated so you have to recreate them. After running the store migration, the `neo4j-admin copy` command extracts the schema and generates a list of commands you can later use to recreate your schema on the new 4.x store. The recreate schema commands are also saved in the migration log file, located in the */logs* directory.

Prepare for seeding the cluster

On the elected instance

Use `neo4j-admin dump` to make a dump of your newly migrated database and transactions.

```
bin/neo4j-admin dump --database=neo4j --to=$BACKUP_DESTINATION/neo4j.dump
```

Be aware that after you migrate, Neo4j Admin commands can differ slightly because Neo4j now supports multiple databases.

Do **not** yet start the server.

Seed the cluster

If you are migrating to a version of Neo4j prior to 4.3 and your migrated database is set as the `default` database in *neo4j.conf*, you should copy the migrated database directory from the elected instance to all other instances to seed the cluster. This step is required so that all instances have the same copy of the database when the database is started. If the migrated database is not the `default` database and the Neo4j version is 4.3+, this step is not required.

1. Copy the dump to the remaining instances.
2. Use `neo4j-admin load --from=<archive-path> --database=<db_name> --force` to replace each of your databases with the one migrated on the elected instance:

```
bin/neo4j-admin load --from=$BACKUP_DESTINATION/neo4j.dump --database=neo4j --force
```

Start the cluster

On each cluster member, including the elected instance

Before continuing, make sure the following activities happened and were completed successfully:

- Content of *neo4j.conf* is correct and required changes were applied on all instances.
- Single migration event occurred on elected instance.
- Backup (via `neo4j-admin dump`) of migrated store performed on the elected instance.
- Backup of the migrated store was transferred to the remaining instances.
- Store was loaded on the remaining instances (via `neo4j-admin load`).

1. If everything on the list was successful, you can go ahead and start all instances of the cluster.

```
bin/neo4j start
```

or

```
systemctl start neo4j
```

2. If the migrated database is the `default` database, it should have been started automatically on instance startup and this step is not required. If the migrated database is not the `default` database, it is still in the `STOPPED` state. You now need to start the database. On one of the cluster members, run the following command in Neo4j Browser or Cypher Shell:

Neo4j 4.0/4.1/4.2

```
CREATE DATABASE <db_name>;
```

Neo4j 4.3+

```
CREATE DATABASE <db_name> OPTIONS { existingData : 'use', existingDataSeedInstance:
'<seedInstanceId>'};
```

Where `<seedInstanceId>` is the ID of the elected instance, which can be found by calling `CALL dbms.cluster.overview()`.

Recreate indexes

The final step is to recreate any indexes or constraints that were output by the `neo4j-admin copy` command. On one of the cluster members, change the active database to the newly migrated one and run the following procedure:

```
CALL db.createIndex('index_5c0607ad', ['Person'], ['name'], 'native-btree-1.0', {`spatial.cartesian-
3d.min`: [-1000000.0, -1000000.0, -1000000.0],`spatial.cartesian.min`: [-1000000.0, -1000000.0],
`spatial.wgs-84.min`: [-180.0, -90.0],`spatial.cartesian-3d.max`: [1000000.0, 1000000.0, 1000000.0],
`spatial.cartesian.max`: [1000000.0, 1000000.0],`spatial.wgs-84-3d.min`: [-180.0, -90.0, -1000000.0],
`spatial.wgs-84-3d.max`: [180.0, 90.0, 1000000.0],`spatial.wgs-84.max`: [180.0, 90.0]})
```

Post-migration

Recreate user data

Neo4j 3.5.x stores user and roles information in a flat file located under $NEO4J_HOME/data/dbms directory. Starting with Neo4j 4.0, this information is stored instead on the `system` database. If you were using native users, you need to recreate them. Go to the backed-up content of your old *$NEO4J_HOME/data/dbms* directory. The authentication data is found in the `auth` file, which is a column separated CSV file looking like this:

```
neo4j:SHA256,1066956C2D4E46C810CA39AE218AAD128854F2C08E9E831C379958CBFA6FF17D,899F9D67F2
96746766848D92B325B29EAFD9AC93940257713BA7CF4CF2B166FF:
```

The first column contains the username, the second column the password information. User can be recreated using the `CREATE USER` statement against the `system` database, such as:

```
CREATE USER neo4j SET ENCRYPTED PASSWORD
‘0,1066956C2D4E46C810CA39AE218AAD128854F2C08E9E831C379958CBFA6FF17D,899F9D67F29
6746766848D92B325B29EAFD9AC93940257713BA7CF4CF2B166FF’ CHANGE NOT REQUIRED
```

Where the string SHA-256 is replaced by the character 0 (zero).

The role data is found in the roles files, looking like this:

```
admin:neo4j
```

This can be recreated by running the following, again against the system database:

```
GRANT ROLE admin TO neo4j
```

You can use Neo4j to parse the auth and roles files. This will process the files and generate all `CREATE USER` and `GRANT ROLE` commands required to recreate all users and roles. To do this, you simply need to move both your backed-up auth and roles files to Neo4j's */import* directory. After that you can use the following two queries, one for users and the other for roles:

Recreate all users

```
LOAD CSV FROM 'file:///auth' as line
with split(line[0], ":")[0] as user, split(line[2], ":") as hash
with user, hash[0] as pwd, CASE hash[1] WHEN "" THEN "NOT" ELSE "" END as
pwdChange
with "CREATE OR REPLACE USER "+user+" SET ENCRYPTED PASSWORD '0,"+pwd+"' CHANGE
"+pwdChange+" REQUIRED" as cypher
return *
```

Recreate all roles

```
LOAD CSV FROM 'file:///roles' as line FIELDTERMINATOR ':'
WITH line[0] as role, split(line[1],",") as users
UNWIND users as user
with "GRANT ROLE "+role+" TO "+user as cypher
return *
```

Each of these queries returns a list of Cypher commands which, when executed against the `system` database, recreates all users and roles previously used in the Neo4j 3.5.x deployment.

Review the logs and metrics

It is advisable to review the logs and metrics to make sure everything looks good. All things going well, you should see error free logs and correctly reported metrics.

Restart the server/cluster

It is advisable to restart the server/cluster one last time just to clear everything and assume the last configuration changes.

Reactivate external applications connecting to Neo4j

After the restart and confirmation that everything was successfully migrated and healthy, you can proceed to reactivate any applications you have connecting to Neo4j. At this point, the Neo4j store migration is complete, and you need to focus on the application side, making sure that all your requests are being served and your application is on a healthy state.

Clean up space

You can clean up the disk space taken by the extra backups required for the migration.

Back up

It is recommended to perform a *full backup*, using an empty target directory.

7.10. Tutorials

This chapter contains tutorials on how to migrate/upgrade a single database into a Neo4j DBMS on a later version.

It covers the following topics:

- Back up and restore a single database in a single instance
- Back up and restore a single database in a cluster
- Back up and copy a single database in a single instance
- Back up and copy a single database in a cluster

7.10.1. Tutorial: Back up and restore a single database in a single instance

This tutorial provides a detailed example of how to back up and restore a database, in this example version 3.5, into a running (already upgraded/migrated) 4.x standalone instance.

The following example assumes that your database has users and roles associated with it and describes how to back it up, migrate it, and then restore it on a running standalone instance.

Prepare to back up the database

Before you perform the backup, it is good to take a note of the data and metadata of the database that you want to restore. You can use this information later to verify that the restore is successful and to recreate the database users and roles. In this example, the database uses the Movie Graph dataset from the Neo4j Browser → Favorites → Example Graphs.

1. In the 3.5 Neo4j instance, where the database is running, navigate to the */bin* folder and log in to the Cypher Shell command-line console with your credentials. For more information about the Cypher Shell command-line interface (CLI) and how to use it, see Operations Manual → Cypher Shell.

   ```
   ./cypher-shell -u neo4j -p <password>
   ```

   ```
   Connected to Neo4j at neo4j://localhost:7687 as user neo4j.
   Type :help for a list of available commands or :exit to exit the shell.
   Note that Cypher queries must end with a semicolon.
   ```

2. Run a query to count the number of nodes in the database.

```
MATCH (n) RETURN count(n) AS countNode;
```

```
+-----------+
| countNode |
+-----------+
| 171       |
+-----------+

1 row available after 22 ms, consumed after another 1 ms
```

3. Run a query to count the number of relationships.

```
MATCH (n)-[r]->() RETURN count(r) AS countRelationships;
```

```
+--------------------+
| countRelationships |
+--------------------+
| 253                |
+--------------------+

1 row available after 29 ms, consumed after another 0 ms
```

4. Run the following two queries to see if there is a schema defined.

```
CALL db.constraints()
```

```
0 rows available after 2 ms, consumed after another 0 ms
```

The result shows that there are no constraints defined.

```
CALL db.indexes;
```

```
+-----------------------------------------------------------------------------------------------------
------------------------------------------------------------------------+
| description              | indexName | tokenNames | properties | state    | type                  |
progress | provider                              | id | failureMessage |
+-----------------------------------------------------------------------------------------------------
------------------------------------------------------------------------+
| "INDEX ON :Movie(title)" | "index_1" | ["Movie"]  | ["title"]  | "ONLINE" | "node_label_property" |
100.0    | {version: "1.0", key: "native-btree"} | 1  | ""             |
+-----------------------------------------------------------------------------------------------------
------------------------------------------------------------------------+
```

The result shows that there is one index defined on the property `title` of the `:Movie` node.

5. Run a query to list all users associated with this database and their roles.

```
CALL dbms.security.listUsers;
```

```
+-------------------------------------------+
| username | roles                | flags |
+-------------------------------------------+
| "user1"  | ["editor", "reader"] | []    |
| "neo4j"  | ["admin"]            | []    |
+-------------------------------------------+

2 rows available after 2 ms, consumed after another 0 ms
```

The result shows two users - the default `neo4j` user, which has `admin` privileges, and a custom user `user1`, which has the combined privileges of the built-in roles `editor` and `reader`.

6. Exit the Cypher Shell command-line console.

```
:exit;

Bye!
```

Back up the database

Now you are ready to back up the database.

Navigate to the */bin* folder, and run the following command to back up the database in your targeted folder. If the folder where you want to place your backup does not exist, you have to create it. In this example, it is called */tmp/3.5.24*.

```
./neo4j-admin backup --backup-dir=/tmp/3.5.24 --name=graphdbbackup
```

For details on performing a backup and the different command options, see Operations Manual 3.5 → Perform a backup.

Restore the database backup on a 4.x standalone instance

You have a running Neo4j 4.x instance, and you want to restore your backed up database in it.

1. In the *neo4j.conf* file of the 4.x standalone instance, set `dbms.allow_upgrade=true`.

   If your Neo4j standalone instance is a version earlier than 4.1, you have to restart it for the configuration to take effect.

2. Navigate to the */bin* folder and run the following command to restore the database backup.

```
./neo4j-admin restore --from=/tmp/3.5.24/graphdbbackup --database=graphdbbackup
```

3. Run the following command to verify that the database `graphdbbackup` exists:

```
ls -al ../data/databases
```

```
total 0
drwxr-xr-x@  6 username  staff   192  4 Dec 14:15 .
drwxr-xr-x@  5 username  staff   160  7 Dec 09:35 ..
drwxr-xr-x  42 username  staff  1344  4 Dec 14:15 graphdbbackup
drwxr-xr-x  37 username  staff  1184  4 Dec 14:06 neo4j
-rw-r--r--   1 username  staff     0  4 Dec 14:06 store_lock
drwxr-xr-x  38 username  staff  1216  4 Dec 14:06 system
```

However, restoring a database does not automatically create it. Therefore, it will not be visible if you do `SHOW DATABASES` in Cypher Shell or Neo4j Browser.

4. Log in to the Cypher Shell command-line console.
5. Change the active database to `system` (`:USE system;`), and create the `graphdbbackup` database.

```
CREATE DATABASE graphdbbackup;
```

```
0 rows available after 145 ms, consumed after another 0 ms
```

6. Verify that the `graphdbbackup` database is online.

```
SHOW DATABASES;
```

```
+----------------------------------------------------------------------------------------------------------
--+
| name            | address          | role         | requestedStatus | currentStatus | error |
default |
+----------------------------------------------------------------------------------------------------------
--+
| "graphdbbackup" | "localhost:7687" | "standalone" | "online"        | "online"      | ""    | FALSE
|
| "neo4j"         | "localhost:7687" | "standalone" | "online"        | "online"      | ""    | TRUE
|
| "system"        | "localhost:7687" | "standalone" | "online"        | "online"      | ""    | FALSE
|
+----------------------------------------------------------------------------------------------------------
--+

3 rows available after 175 ms, consumed after another 9 ms
```

7. Change the active database to `graphdbbackup`, and repeat steps 2, 3, and 4 from the section Prepare to back up the database, to verify that all the data has been successfully restored.

Recreate the database users and roles

You need to manually recreate all users and roles of the restored database using your notes from step 5 of the section Prepare to back up the database and the Cypher Manual → Cypher administration commands.

Example 15. Run the following commands against the `system` database to recreate the `graphdbbackup` database's custom users and roles.

1. Create the user `user1`.

```
CREATE USER user1 IF NOT EXISTS
SET PASSWORD 'password'
SET STATUS ACTIVE;
```

2. Grant the role `reader` to the user `user1`.

```
GRANT ROLE reader TO user1;
```

3. Grant the role `editor` to the user `user1`.

```
GRANT ROLE editor TO user1;
```

4. Verify that the user `user1` has the same roles as in the database backup.

```
SHOW USERS;
```

```
+-----------------------------------------------------------------------+
| user    | roles                  | passwordChangeRequired | suspended |
+-----------------------------------------------------------------------+
| "neo4j" | ["admin"]              | FALSE                  | FALSE     |
| "user1" | ["editor", "reader"]   | TRUE                   | FALSE     |
+-----------------------------------------------------------------------+
```

7.10.2. Tutorial: Back up and restore a single database in a cluster

This tutorial provides a detailed example of how to back up and restore a database, in this example version 3.5, in a running (already upgraded/migrated) 4.x cluster.

The following example assumes that your database has users and roles associated with it and describes how to back it up, migrate it into a standalone instance, and then restore it in a running cluster. For more information on how to set up a cluster, see Operations Manual → Set up a local cluster.

In a Neo4j DBMS, every database is backed up individually. Therefore, it is very important to plan your backup strategy for each of them. For more detailed information on how to design an appropriate backup strategy for your setup, see Operations Manual 3.5 → Backup.

Prepare to back up the database

Before you perform the backup, it is good to take a note of the data and metadata of the database that you want to restore. You can use this information later to verify that the restore is successful and to recreate the database users and roles. In this example, the database uses the Movie Graph dataset from the Neo4j Browser → Favorites → Example Graphs.

1. In the 3.5 Neo4j instance, where the database is running, navigate to the */bin* folder and log in to the Cypher Shell command-line console with your credentials. For more information about the Cypher Shell command-line interface (CLI) and how to use it, see Operations Manual → Cypher Shell.

   ```
   ./cypher-shell -u neo4j -p <password>
   ```

   ```
   Connected to Neo4j at neo4j://localhost:7687 as user neo4j.
   Type :help for a list of available commands or :exit to exit the shell.
   Note that Cypher queries must end with a semicolon.
   ```

2. Run a query to count the number of nodes in the database.

   ```
   MATCH (n) RETURN count(n) AS countNode;
   ```

   ```
   +-----------+
   | countNode |
   +-----------+
   | 171       |
   +-----------+

   1 row available after 22 ms, consumed after another 1 ms
   ```

3. Run a query to count the number of relationships.

   ```
   MATCH (n)-[r]->() RETURN count(r) AS countRelationships;
   ```

   ```
   +--------------------+
   | countRelationships |
   +--------------------+
   | 253                |
   +--------------------+

   1 row available after 29 ms, consumed after another 0 ms
   ```

4. Run the following two queries to see if there is a schema defined.

   ```
   CALL db.constraints()
   ```

   ```
   0 rows available after 2 ms, consumed after another 0 ms
   ```

   The result shows that there are no constraints defined.

   ```
   CALL db.indexes;
   ```

```
+----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
| description               | indexName | tokenNames | properties | state    | type                 |
progress | provider                              | id | failureMessage |
+----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
| "INDEX ON :Movie(title)" | "index_1" | ["Movie"]  | ["title"]  | "ONLINE" | "node_label_property" |
100.0    | {version: "1.0", key: "native-btree"} | 1  | ""             |
+----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
```

The result shows that there is one index defined on the property `title` of the `:Movie` node.

5. Run a query to list all users associated with this database and their roles.

```
CALL dbms.security.listUsers;
```

```
+-------------------------------------------+
| username | roles                 | flags |
+-------------------------------------------+
| "user1"  | ["editor", "reader"] | []    |
| "neo4j"  | ["admin"]             | []    |
+-------------------------------------------+

2 rows available after 2 ms, consumed after another 0 ms
```

The result shows two users - the default `neo4j` user, which has `admin` privileges, and a custom user `user1`, which has the combined privileges of the built-in roles `editor` and `reader`.

6. Exit the Cypher Shell command-line console.

```
:exit;

Bye!
```

Back up the database

Now you are ready to back up the database.

Navigate to the */bin* folder, and run the following command to back up the database in your targeted folder. If the folder where you want to place your backup does not exist, you have to create it. In this example, it is called */tmp/3.5.24*.

```
./neo4j-admin backup --backup-dir=/tmp/3.5.24 --name=graphdbbackup
```

For details on performing a backup and the different command options, see Operations Manual 3.5 → Perform a backup.

Migrate the database backup to a 4.x standalone instance

To migrate the database backup and upgrade its store, you can spin up a standalone Neo4j instance and use the command `neo4j-admin restore`.

`neo4j-admin restore` must be invoked as the `neo4j` user to ensure the appropriate file permissions. For more information, see Operations Manual → Administrative commands.

If the `--prepare-restore` option has been disabled when backing up your database, you must run the `neo4j-admin prepare-restore` command before you can restore your database. This is to apply the latest transactions pulled at the backup time but not yet applied to the store. For more information, see Prepare a database for restoring.

1. In the *neo4j.conf* file of a 4.x standalone instance, set `dbms.allow_upgrade=true`.
2. Start the server.
3. Navigate to the */bin* folder and run the following command to restore the database backup.

   ```
   ./neo4j-admin restore --from=/tmp/3.5.24/graphdbbackup --database=graphdbbackup
   ```

4. Run the following command to verify that the database `graphdbbackup` exists:

   ```
   ls -al ../data/databases
   ```

   ```
   total 0
   drwxr-xr-x@  6 username  staff   192  4 Dec 14:15 .
   drwxr-xr-x@  5 username  staff   160  7 Dec 09:35 ..
   drwxr-xr-x  42 username  staff  1344  4 Dec 14:15 graphdbbackup
   drwxr-xr-x  37 username  staff  1184  4 Dec 14:06 neo4j
   -rw-r--r--   1 username  staff     0  4 Dec 14:06 store_lock
   drwxr-xr-x  38 username  staff  1216  4 Dec 14:06 system
   ```

   However, restoring a database does not automatically create it. Therefore, it will not be visible if you do `SHOW DATABASES` in Cypher Shell or Neo4j Browser.

5. Log in to the Cypher Shell command-line console.
6. Change the active database to `system` (`:USE system;`), and create the `graphdbbackup` database.

   ```
   CREATE DATABASE graphdbbackup;
   ```

   ```
   0 rows available after 145 ms, consumed after another 0 ms
   ```

7. Verify that the `graphdbbackup` database is online.

   ```
   SHOW DATABASES;
   ```

```
+----------------------------------------------------------------------------------------------------
--+
| name            | address          | role         | requestedStatus | currentStatus | error |
default |
+----------------------------------------------------------------------------------------------------
--+
| "graphdbbackup" | "localhost:7687" | "standalone" | "online"        | "online"      | ""    | FALSE
|
| "neo4j"         | "localhost:7687" | "standalone" | "online"        | "online"      | ""    | TRUE
|
| "system"        | "localhost:7687" | "standalone" | "online"        | "online"      | ""    | FALSE
|
+----------------------------------------------------------------------------------------------------
--+

3 rows available after 175 ms, consumed after another 9 ms
```

8. Change the active database to `graphdbbackup`, and repeat steps 2, 3, and 4 from the section Prepare to back up the database, to verify that all the data has been successfully restored.

Back up the migrated database

To restore the migrated database in your cluster, you create a backup of it.

Navigate to the */bin* folder, and run the following command to back up the database in your targeted folder. If the folder where you want to place your backup does not exist, you have to create it. In this example, it is called */tmp/4.0.11*.

```
./neo4j-admin backup --backup-dir=/tmp/4.0.11 --database=graphdbbackup
```

For details on performing a backup and the different command options, see Operations Manual → Back up an online database.

Now you are ready to restore your migrated database backup into your running cluster.

Restore the migrated database on all cluster members

On each cluster member, navigate to the */bin* folder, and run the following command to restore the backup of the migrated database. For details on performing a restore and the different command options, see Operations Manual → Restore command.

```
./neo4j-admin restore --from=/tmp/4.0.11/graphdbbackup --database=graphdbbackup
```

Then, on each cluster member, run the following command to verify that the database `graphdbbackup` exists:

```
ls -al ../data/databases
```

```
total 0
drwxr-xr-x@  6 username  staff   192  7 Dec 09:41 .
drwxr-xr-x@  5 username  staff   160  4 Dec 14:17 ..
drwxr-xr-x  38 username  staff  1216  7 Dec 09:41 graphdbbackup
drwxr-xr-x  37 username  staff  1184  7 Dec 08:57 neo4j
-rw-r--r--   1 username  staff     0  4 Dec 14:17 store_lock
drwxr-xr-x  38 username  staff  1216  7 Dec 08:56 system
```

Create the database backup on the cluster leader

You create the database backup **only on one of your cluster members** using the command `CREATE DATABASE`. The command is automatically routed to the leader, and from there, to the other cluster members.

1. In the Cypher Shell command-line console on one of the cluster members, use the `system` database and create the database `graphdbbackup`.

   ```
   CREATE DATABASE graphdbbackup;
   ```

   ```
   0 rows available after 132 ms, consumed after another 0 ms
   ```

2. Verify that the `graphdbbackup` database is online on all members.

   ```
   SHOW DATABASES;
   ```

   ```
   +----------------------------------------------------------------------------------------------------------
   +
   | name            | address          | role       | requestedStatus | currentStatus | error | default
   |
   +----------------------------------------------------------------------------------------------------------
   +
   | "graphdbbackup" | "localhost:7689" | "follower" | "online"        | "online"      | ""    | FALSE
   |
   | "graphdbbackup" | "localhost:7688" | "leader"   | "online"        | "online"      | ""    | FALSE
   |
   | "graphdbbackup" | "localhost:7687" | "follower" | "online"        | "online"      | ""    | FALSE
   |
   | "neo4j"         | "localhost:7689" | "leader"   | "online"        | "online"      | ""    | TRUE
   |
   | "neo4j"         | "localhost:7688" | "follower" | "online"        | "online"      | ""    | TRUE
   |
   | "neo4j"         | "localhost:7687" | "follower" | "online"        | "online"      | ""    | TRUE
   |
   | "system"        | "localhost:7689" | "leader"   | "online"        | "online"      | ""    | FALSE
   |
   | "system"        | "localhost:7688" | "follower" | "online"        | "online"      | ""    | FALSE
   |
   | "system"        | "localhost:7687" | "follower" | "online"        | "online"      | ""    | FALSE
   |
   +----------------------------------------------------------------------------------------------------------
   +
   ```

3. Change your active database to `graphdbbackup` and verify that all the data has been successfully restored by completing steps 2, 3, and 4 of the Prepare to back up the database section.

Recreate the database users and roles

On all cluster members, manually recreate all users and roles of the restored database using your notes from step 5 of the section Prepare to back up the database and the Cypher Manual → Cypher administration commands.

Example 16. Run the following commands against the `system` database to recreate the `graphdbbackup` database's custom users and roles.

1. Create the user `user1`.

```
CREATE USER user1 IF NOT EXISTS
SET PASSWORD 'password'
SET STATUS ACTIVE;
```

2. Grant the role `reader` to the user `user1`.

```
GRANT ROLE reader TO user1;
```

3. Grant the role `editor` to the user `user1`.

```
GRANT ROLE editor TO user1;
```

4. Verify that the user `user1` has the same roles as in the database backup.

```
SHOW USERS;
```

```
+---------------------------------------------------------------------+
| user    | roles                 | passwordChangeRequired | suspended |
+---------------------------------------------------------------------+
| "neo4j" | ["admin"]             | FALSE                  | FALSE     |
| "user1" | ["editor", "reader"] | TRUE                   | FALSE     |
+---------------------------------------------------------------------+
```

7.10.3. Tutorial: Back up and copy a single database in a running single instance

This tutorial provides a detailed example of how to back up a single database, in this example version 3.5, and use the `neo4j-admin copy` command to copy it into a running 4.x Neo4j standalone instance.

The `neo4j-admin copy` command can be used to clean up database inconsistencies, compact stores, and upgrade/migrate a database (from Community or Enterprise) to a later version of Neo4j Enterprise edition. Since the `neo4j-admin copy` command does not copy the schema store, the intermediary steps of the sequential path are not needed. If a schema is defined, you just have to recreate it by running the commands that the `neo4j-admin copy` operation outputs.

Keep in mind that:

- The `neo4j-admin copy` command copies the node IDs, but the relationships get new IDs.
- `neo4j-admin copy` is a copy tool, not an upgrade or migration tool.
  It copies only a single database and **cannot** be applied to the `system` database.

Therefore, if you want to preserve your relationships IDs, or to upgrade the whole DBMS, you should follow the sequential path.

It is important to note that `neo4j-admin copy` is an IOPS-intensive process.

Estimations for how long the `neo4j-admin copy` command will take can be made based on the following:

- Neo4j, like many other databases, does IO in 8K pages.
- The maximum value of IOPS your disc can process, provided by your disc manufacturer.

For example, if your disc manufacturer has provided a maximum of 5000 IOPS, you can reasonably expect up to 5000 such page operations a second. Therefore, the maximal theoretical throughput you can expect is 40MB/s (or 144 GB/hour) on that disc. You may then assume that the best-case scenario for running `neo4j-admin copy` on that 5000 IOPS disc is that it will take at least 1 hour to process a 144 GB database. [1]

However, it is important to remember that the process must read 144 GB from the source database, and must also write to the target store (assuming the target store is of comparable size). Additionally, there are internal processes during the copy that will read/modify/write the store multiple times. Therefore, with an additional 144 GB of both read and write, the best-case scenario for running `neo4j-admin copy` on a 5000 IOPS disc is that it will actually take **at least 3 hours to process a 144 GB database**.

Finally, it is also important to consider that in almost all Cloud environments, the published IOPS value may not be the same as the actual value, or be able to continuously maintain the maximum possible IOPS. The real processing time for this example *could* be well above that estimation of 3 hours.

This tutorial walks through the basics of checking your database store usage, in this example version 3.5, performing a backup, compacting the database backup (using `neo4j-admin copy`), and creating it in a running Neo4j 4.x standalone instance.

Check your 3.5 database store usage

Before you back up and copy your 3.5 database, let's look at the database store usage and see how it changes when you load, delete, and then reload data.

1. Log in to Neo4j Browser of your running 3.5 Neo4j standalone instance, add 100k nodes to the `graph.db` database using the following command:

```
FOREACH (x IN RANGE (1,100000) | CREATE (n:Person {name:x}))
```

2. Create an index on the `name` property of the `Person` node:

```
CREATE INDEX ON :Person(name)
```

3. Use the `dbms.checkpoint()` procedure to flush all cached updates from the page cache to the store files.

```
CALL dbms.checkpoint()
```

4. In your terminal, navigate to the `graph.db` database (*$neo4j_home/data/databases/graph.db*) and run the following command to check the store size of the loaded nodes and properties.

```
ls -alh
```

```
...
-rw-r--r--   1 username  staff   1.4M 26 Nov 15:51 neostore.nodestore.db
-rw-r--r--   1 username  staff   3.9M 26 Nov 15:51 neostore.propertystore.db
...
```

The output reports that the node store (*neostore.nodestore.db*) and the property store (*neostore.propertystore.db*) occupy `1.4M` and `3.9M`, respectively.

5. In Neo4j Browser, delete the nodes created above and run `CALL dbms.checkpoint` again to force a checkpoint.

```
MATCH (n) DETACH DELETE n
```

```
CALL dbms.checkpoint()
```

6. Now, add just one node, force a checkpoint, and repeat step 4 to see if the store size has changed.

```
CREATE (n:Person {name:"John"})
```

```
CALL dbms.checkpoint()
```

If you check the size of the node store and the property store now, they will still be `1.4M` and `3.9M`, even though the database only contains one node and one property. Neo4j does not shrink the store files on the hard drive.

In a production database, where numerous load/delete operations are performed, the result is a significant unused space occupied by store files.

Back up your 3.5 database

Navigate to the */bin* folder, and run the following command to back up your database in the targeted folder. If the folder where you want to place your backup does not exist, you have to create it. In this example, it is called */tmp/3.5.24*.

```
./neo4j-admin backup --backup-dir=/tmp/3.5.24 --name=graphdbbackup
```

For details on performing a backup and the different command options, see Operations Manual → Perform a backup.

Copy your 3.5 database backup to 4.x Neo4j

You can use the `neo4j-admin copy` command to reclaim the unused space and create a defragmented copy of your database backup in your 4.x standalone instance.

> To speed up the copy operation, you can use the `--from-pagecache` and `--to-pagecache` options to specify how much cache to be allocated when reading the source and writing the destination. As a rule of thumb, `--to-pagecache` should be around 1-2GB, since it mostly does sequential writes. The `--from-pagecache` should then be assigned whatever memory you can spare, since Neo4j does random reads from the source.

1. In your 4.x Neo4j standalone instance, navigate to the */bin* folder and run the following command to create a compacted store copy of your 3.5 database backup. Any inconsistent nodes, properties, and relationships will not be copied over to the newly created store.

   ```
   ./neo4j-admin copy --from-path=/private/tmp/3.5.24/graphdbbackup --to-database=compactdb
   ```

   ```
   Starting to copy store, output will be saved to:  $neo4j_home/logs/neo4j-admin-copy-2020-11-
   26.16.07.19.log
   2020-11-26 16:07:19.939+0000 INFO [StoreCopy] ### Copy Data ###
   2020-11-26 16:07:19.940+0000 INFO [StoreCopy] Source: /private/tmp/3.5.24/graphdbbackup (page cache
   8m)
   2020-11-26 16:07:19.940+0000 INFO [StoreCopy] Target:  $neo4j_home/data/databases/compactdb (page
   cache 8m)
   2020-11-26 16:07:19.940+0000 INFO [StoreCopy] Empty database created, will start importing readable
   data from the source.
   2020-11-26 16:07:21.661+0000 INFO [o.n.i.b.ImportLogic] Import starting

   Import starting 2020-11-26 16:07:21.699+0000
     Estimated number of nodes: 50.00 k
     Estimated number of node properties: 50.00 k
     Estimated number of relationships: 0.00
     Estimated number of relationship properties: 50.00 k
     Estimated disk space usage: 2.680MiB
     Estimated required memory usage: 8.598MiB

   (1/4) Node import 2020-11-26 16:07:22.220+0000
     Estimated number of nodes: 50.00 k
     Estimated disk space usage: 1.698MiB
     Estimated required memory usage: 8.598MiB
   .......... .......... .......... .......... ..........   5% ∆239ms
   .......... .......... .......... .......... ..........  10% ∆1ms
   .......... .......... .......... .......... ..........  15% ∆1ms
   .......... .......... .......... .......... ..........  20% ∆0ms
   .......... .......... .......... .......... ..........  25% ∆1ms
   .......... .......... .......... .......... ..........  30% ∆0ms
   .......... .......... .......... .......... ..........  35% ∆0ms
   ```

```
.......... .......... .......... .......... ..........  40% ∆1ms
.......... .......... .......... .......... ..........  45% ∆0ms
.......... .......... .......... .......... ..........  50% ∆1ms
.......... .......... .......... .......... ..........  55% ∆0ms
.......... .......... .......... .......... .........-  60% ∆51ms
.......... .......... .......... .......... ..........  65% ∆0ms
.......... .......... .......... .......... ..........  70% ∆0ms
.......... .......... .......... .......... ..........  75% ∆1ms
.......... .......... .......... .......... ..........  80% ∆0ms
.......... .......... .......... .......... ..........  85% ∆0ms
.......... .......... .......... .......... ..........  90% ∆1ms
.......... .......... .......... .......... ..........  95% ∆0ms
.......... .......... .......... .......... .......... 100% ∆0ms

(2/4) Relationship import 2020-11-26 16:07:22.543+0000
  Estimated number of relationships: 0.00
  Estimated disk space usage: 1006KiB
  Estimated required memory usage: 15.60MiB
(3/4) Relationship linking 2020-11-26 16:07:22.879+0000
  Estimated required memory usage: 7.969MiB
(4/4) Post processing 2020-11-26 16:07:23.272+0000
  Estimated required memory usage: 7.969MiB
-......... .......... .......... .......... ..........   5% ∆356ms
.......... .......... .......... .......... ..........  10% ∆0ms
.......... .......... .......... .......... ..........  15% ∆1ms
.......... .......... .......... .......... ..........  20% ∆0ms
.......... .......... .......... .......... ..........  25% ∆0ms
.......... .......... .......... .......... ..........  30% ∆1ms
.......... .......... .......... .......... ..........  35% ∆0ms
.......... .......... .......... .......... ..........  40% ∆0ms
.......... .......... .......... .......... ..........  45% ∆1ms
.......... .......... .......... .......... ..........  50% ∆0ms
.......... .......... .......... .......... ..........  55% ∆0ms
.......... .......... .......... .......... ..........  60% ∆0ms
.......... .......... .......... .......... ..........  65% ∆1ms
.......... .......... .......... .......... ..........  70% ∆0ms
.......... .......... .......... .......... ..........  75% ∆0ms
.......... .......... .......... .......... ..........  80% ∆0ms
.......... .......... .......... .......... ..........  85% ∆0ms
.......... .......... .......... .......... ..........  90% ∆0ms
.......... .......... .......... .......... ..........  95% ∆1ms
.......... .......... .......... .......... .......... 100% ∆0ms


IMPORT DONE in 2s 473ms.
Imported:
  1 nodes
  0 relationships
  1 properties
Peak memory usage: 15.60MiB
2020-11-26 16:07:24.140+0000 INFO [o.n.i.b.ImportLogic] Import completed successfully, took 2s 473ms.
Imported:
  1 nodes
  0 relationships
  1 properties
2020-11-26 16:07:24.668+0000 INFO [StoreCopy] Import summary: Copying of 100704 records took 4 seconds
(25176 rec/s). Unused Records 100703 (99%) Removed Records 0 (0%)
2020-11-26 16:07:24.669+0000 INFO [StoreCopy] ### Extracting schema ###
2020-11-26 16:07:24.669+0000 INFO [StoreCopy] Trying to extract schema...
2020-11-26 16:07:24.920+0000 INFO [StoreCopy] ... found 1 schema definitions. The following can be
used to recreate the schema:
2020-11-26 16:07:24.922+0000 INFO [StoreCopy]

CALL db.createIndex('index_5c0607ad', ['Person'], ['name'], 'native-btree-1.0', {`spatial.cartesian-
3d.min`: [-1000000.0, -1000000.0, -1000000.0],`spatial.cartesian.min`: [-1000000.0,
-1000000.0],`spatial.wgs-84.min`: [-180.0, -90.0],`spatial.cartesian-3d.max`: [1000000.0, 1000000.0,
1000000.0],`spatial.cartesian.max`: [1000000.0, 1000000.0],`spatial.wgs-84-3d.min`: [-180.0, -90.0,
-1000000.0],`spatial.wgs-84-3d.max`: [180.0, 90.0, 1000000.0],`spatial.wgs-84.max`: [180.0, 90.0]})
2020-11-26 16:07:24.923+0000 INFO [StoreCopy] You have to manually apply the above commands to the
database when it is stared to recreate the indexes and constraints. The commands are saved to
$neo4j_home/logs/neo4j-admin-copy-2020-11-26.16.07.19.log as well for reference.
```

2. Run the following command to verify that database has been successfully copied.

```
ls -al ../data/databases
```

```
total 0
drwxr-xr-x@  5 username  staff   160 26 Nov 18:00 .
drwxr-xr-x@  5 username  staff   160 26 Nov 18:00 ..
drwxr-xr-x  35 username  staff  1120 26 Nov 17:58 compactdb
-rw-r--r--   1 username  staff     0 26 Nov 18:00 store_lock
drwxr-xr-x  33 username  staff  1056 26 Nov 18:00 system
```

Copying a database does not automatically create it. Therefore, it will not be visible if you do `SHOW DATABASES` in Cypher Shell or Neo4j Browser.

Create your compacted backup

You can now create the copied database and compare its store size with the size of the backed up database.

1. Log in to the Cypher Shell command-line console, change the active database to `system` (`:USE system;`), and create the `compactdb` database. For more information about the Cypher Shell command-line interface (CLI) and how to use it, see Operations Manual → Cypher Shell.

```
CREATE DATABASE compactdb;
```

```
0 rows available after 145 ms, consumed after another 0 ms
```

2. Verify that the `compactdb` database is online.

```
SHOW DATABASES;
```

```
+---------------------------------------------------------------------------------------------------------
--+
| name             | address          | role         | requestedStatus | currentStatus | error |
default |
+---------------------------------------------------------------------------------------------------------
--+
| "compactdb"      | "localhost:7687" | "standalone" | "online"        | "online"      | ""    | FALSE
|
| "neo4j"          | "localhost:7687" | "standalone" | "online"        | "online"      | ""    | TRUE
|
| "system"         | "localhost:7687" | "standalone" | "online"        | "online"      | ""    | FALSE
|
+---------------------------------------------------------------------------------------------------------
--+

3 rows available after 10 ms, consumed after another 3 ms
```

3. Change your active database to `compactdb` and recreate the schema using the output from the `neo4j-admin copy` command.

```
CALL db.createIndex('index_5c0607ad', ['Person'], ['name'], 'native-btree-1.0', {`spatial.cartesian-
3d.min`: [-1000000.0, -1000000.0, -1000000.0],`spatial.cartesian.min`: [-1000000.0, -1000000.0
],`spatial.wgs-84.min`: [-180.0, -90.0],`spatial.cartesian-3d.max`: [1000000.0, 1000000.0, 1000000.0
],`spatial.cartesian.max`: [1000000.0, 1000000.0],`spatial.wgs-84-3d.min`: [-180.0, -90.0, -
1000000.0],`spatial.wgs-84-3d.max`: [180.0, 90.0, 1000000.0],`spatial.wgs-84.max`: [180.0, 90.0]});
```

```
+-----------------------------------------------------------------------------------+
| name             | labels     | properties | providerName       | status          |
+-----------------------------------------------------------------------------------+
| "index_5c0607ad" | ["Person"] | ["name"]   | "native-btree-1.0" | "index created" |
+-----------------------------------------------------------------------------------+

1 row available after 50 ms, consumed after another 5 ms
```

4. Verify that all the data has been successfully copied. In this example, there should be one node.

```
MATCH (n) RETURN n.name;
```

```
+--------+
| n.name |
+--------+
| "John" |
+--------+

1 row available after 106 ms, consumed after another 2 ms
```

5. Exit the Cypher Shell command-line console.

```
:exit;

Bye!
```

6. Navigate to the `compactdb` database (*$neo4j_home/data/databases/compactdb*) and check the store size of the copied nodes and properties.

```
ls -alh
```

```
...
-rw-r--r--   1 username  staff   8.0K 26 Nov 17:58 neostore.nodestore.db
-rw-r--r--   1 username  staff   8.0K 26 Nov 17:58 neostore.propertystore.db
...
```

The output reports that the node store and the property store now occupy only `8K` each, compared to the previous `1.4M` and `3.9M`.

7.10.4. Tutorial: Back up and copy a single database in a running cluster

This tutorial provides a detailed example of how to back up a single database, in this example version 3.5, and use the `neo4j-admin copy` *command to copy it into a running 4.x Neo4j cluster.*

The `neo4j-admin copy` command can be used to clean up database inconsistencies, compact stores, and upgrade/migrate a database (from Community or Enterprise) to a later version of Neo4j Enterprise edition. Since the `neo4j-admin copy` command does not copy the schema store, the intermediary steps of the sequential path are not needed. If a schema is defined, the commands that the `neo4j-admin copy` operation outputs can be used to create the new schema.

Keep in mind that:

- The `neo4j-admin copy` command copies the node IDs, but the relationships get new IDs.
- The `neo4j-admin copy` command is used to copy a single database from a specified Neo4j DBMS path to another Neo4j DBMS. Note that no schema data will be created on the new Neo4j DBMS. The `system` database **cannot** be copied with the `neo4j-admin copy` command.

Therefore, if you want to preserve your relationships IDs, or to upgrade the whole DBMS, you should follow the sequential path.

It is important to note that `neo4j-admin copy` is an IOPS-intensive process.

Estimations for how long the `neo4j-admin copy` command will take can be made based on the following:

- Neo4j, like many other databases, does IO in 8K pages.
- The maximum value of IOPS your disc can process, provided by your disc manufacturer.

For example, if your disc manufacturer has provided a maximum of 5000 IOPS, you can reasonably expect up to 5000 such page operations a second. Therefore, the maximal theoretical throughput you can expect is 40MB/s (or 144 GB/hour) on that disc. You may then assume that the best-case scenario for running `neo4j-admin copy` on that 5000 IOPS disc is that it will take at least 1 hour to process a 144 GB database. [2]

However, it is important to remember that the process must read 144 GB from the source database, and must also write to the target store (assuming the target store is of comparable size). Additionally, there are internal processes during the copy that will read/modify/write the store multiple times. Therefore, with an additional 144 GB of both read and write, the best-case scenario for running `neo4j-admin copy` on a 5000 IOPS disc is that it will actually take **at least 3 hours to process a 144 GB database.**

Finally, it is also important to consider that in almost all Cloud environments, the published IOPS value may not be the same as the actual value, or be able to continuously maintain the maximum possible IOPS. The real processing time for this example *could* be well above that estimation of 3 hours.

This tutorial walks through the basics of checking your database store usage, in this example version 3.5, performing a backup, compacting the database backup (using `neo4j-admin copy`), and creating it in a running Neo4j 4.x cluster.

Check your 3.5 database store usage

Before you back up and copy your 3.5 database, let's look at the database store usage and see how it changes when you load, delete, and then reload data.

1. Log in to Neo4j Browser of your running 3.5 Neo4j standalone instance and add 100k nodes to the `graph.db` database using the following command:

   ```
   FOREACH (x IN RANGE (1,100000) | CREATE (n:Person {name:x}))
   ```

2. Create an index on the `name` property of the `Person` node:

   ```
   CREATE INDEX ON :Person(name)
   ```

3. Use the `dbms.checkpoint()` procedure to flush all cached updates from the page cache to the store files.

   ```
   CALL dbms.checkpoint()
   ```

4. In your terminal, navigate to the `graph.db` database (*$neo4j_home/data/databases/graph.db*) and run the following command to check the store size of the loaded nodes and properties.

   ```
   ls -alh
   ```

   ```
   ...
   -rw-r--r--   1 username  staff   1.4M 26 Nov 15:51 neostore.nodestore.db
   -rw-r--r--   1 username  staff   3.9M 26 Nov 15:51 neostore.propertystore.db
   ...
   ```

   The output reports that the node store (*neostore.nodestore.db*) and the property store (*neostore.propertystore.db*) occupy `1.4M` and `3.9M`, respectively.

5. In Neo4j Browser, delete the nodes created above and run `CALL dbms.checkpoint` again to force a checkpoint.

   ```
   MATCH (n) DETACH DELETE n
   ```

   ```
   CALL dbms.checkpoint()
   ```

6. Now, add just one node, force a checkpoint, and repeat step 4 to see if the store size has changed.

   ```
   CREATE (n:Person {name:"John"})
   ```

   ```
   CALL dbms.checkpoint()
   ```

If you check the size of the node store and the property store now, they will still be `1.4M` and `3.9M`, even though the database only contains one node and one property. Neo4j does not shrink the store files on the hard drive.

In a production database, where numerous load/delete operations are performed, the result is a significant unused space occupied by store files.

Back up your 3.5 database

Navigate to the */bin* folder, and run the following command to back up your database in the targeted folder. If the folder where you want to place your backup does not exist, you have to create it. In this example, it is called */tmp/3.5.24*.

```
./neo4j-admin backup --backup-dir=/tmp/3.5.24 --name=graphdbbackup
```

For details on performing a backup and the different command options, see Operations Manual → Perform a backup.

Copy your 3.5 database backup to 4.x Neo4j cluster

You can use the `neo4j-admin copy` command to reclaim the unused space and create a defragmented copy of your database backup in your 4.x cluster.

To speed up the copy operation, you can use the `--from-pagecache` and `--to-pagecache` options to specify how much cache to be allocated when reading the source and writing the destination. As a rule of thumb, `--to-pagecache` should be around 1-2GB, since it mostly does sequential writes. The `--from-pagecache` should then be assigned whatever memory you can spare, since Neo4j does random reads from the source.

1. On **each cluster member**, navigate to the */bin* folder and run the following command to create a compacted store copy of your 3.5 database backup. Any inconsistent nodes, properties, and relationships will not be copied over to the newly created store.

```
./neo4j-admin copy --from-path=/private/tmp/3.5.24/graphdbbackup --to-database=compactdb
```

```
Selecting JVM - Version:11.0.6+8-LTS, Name:Java HotSpot(TM) 64-Bit Server VM, Vendor:Oracle
Corporation
Starting to copy store, output will be saved to: /Users/renetapopova/neo4j/cc-4.4.0/core1/logs/neo4j-
admin-copy-2022-02-07.11.13.05.log
2022-02-07 11:13:06.920+0000 INFO  [StoreCopy] ### Copy Data ###
2022-02-07 11:13:06.923+0000 INFO  [StoreCopy] Source: /private/tmp/3.5.24/graphdbbackup (page cache
8m)
2022-02-07 11:13:06.924+0000 INFO  [StoreCopy] Target: /Users/renetapopova/neo4j/cc-
4.4.0/core1/data/databases/compactdb
2022-02-07 11:13:06.924+0000 INFO  [StoreCopy] Empty database created, will start importing readable
data from the source.
2022-02-07 11:13:09.911+0000 INFO  [o.n.i.b.ImportLogic] Import starting

Import starting 2022-02-07 11:13:09.963+0000
  Estimated number of nodes: 50.00 k
  Estimated number of node properties: 50.00 k
  Estimated number of relationships: 0.00
  Estimated number of relationship properties: 50.00 k
  Estimated disk space usage: 2.680MiB
  Estimated required memory usage: 36.71MiB

(1/4) Node import 2022-02-07 11:13:11.069+0000
  Estimated number of nodes: 50.00 k
  Estimated disk space usage: 1.698MiB
  Estimated required memory usage: 36.71MiB
.......... .......... .......... .......... ..........   5% ∆236ms
.......... .......... .......... .......... ..........  10% ∆24ms
.......... .......... .......... .......... ..........  15% ∆3ms
.......... .......... .......... .......... ..........  20% ∆2ms
.......... .......... .......... .......... ..........  25% ∆1ms
.......... .......... .......... .......... ..........  30% ∆0ms
.......... .......... .......... .......... ..........  35% ∆0ms
.......... .......... .......... .......... ..........  40% ∆3ms
.......... .......... .......... .......... ..........  45% ∆2ms
.......... .......... .......... .......... ..........  50% ∆1ms
.......... .......... .......... .......... ..........  55% ∆0ms
.......... .......... .......... .......... .........-  60% ∆77ms
.......... .......... .......... .......... ..........  65% ∆2ms
.......... .......... .......... .......... ..........  70% ∆0ms
.......... .......... .......... .......... ..........  75% ∆1ms
.......... .......... .......... .......... ..........  80% ∆0ms
.......... .......... .......... .......... ..........  85% ∆0ms
.......... .......... .......... .......... ..........  90% ∆0ms
.......... .......... .......... .......... ..........  95% ∆0ms
.......... .......... .......... .......... .......... 100% ∆0ms

Node import COMPLETED in 458ms

(2/4) Relationship import 2022-02-07 11:13:11.528+0000
  Estimated number of relationships: 0.00
  Estimated disk space usage: 1006KiB
  Estimated required memory usage: 43.90MiB
Relationship import COMPLETED in 571ms

(3/4) Relationship linking 2022-02-07 11:13:12.100+0000
  Estimated required memory usage: 36.08MiB
Relationship linking COMPLETED in 645ms

(4/4) Post processing 2022-02-07 11:13:12.745+0000
  Estimated required memory usage: 36.08MiB
-......... .......... .......... .......... ..........   5% ∆717ms
.......... .......... .......... .......... ..........  10% ∆1ms
.......... .......... .......... .......... ..........  15% ∆0ms
.......... .......... .......... .......... ..........  20% ∆1ms
.......... .......... .......... .......... ..........  25% ∆1ms
.......... .......... .......... .......... ..........  30% ∆0ms
.......... .......... .......... .......... ..........  35% ∆0ms
.......... .......... .......... .......... ..........  40% ∆0ms
.......... .......... .......... .......... ..........  45% ∆0ms
.......... .......... .......... .......... ..........  50% ∆1ms
.......... .......... .......... .......... ..........  55% ∆0ms
.......... .......... .......... .......... ..........  60% ∆0ms
.......... .......... .......... .......... ..........  65% ∆0ms
.......... .......... .......... .......... ..........  70% ∆0ms
.......... .......... .......... .......... ..........  75% ∆0ms
.......... .......... .......... .......... ..........  80% ∆1ms
.......... .......... .......... .......... ..........  85% ∆0ms
.......... .......... .......... .......... ..........  90% ∆0ms
.......... .......... .......... .......... ..........  95% ∆0ms
.......... .......... .......... .......... .......... 100% ∆0ms

Post processing COMPLETED in 1s 781ms


IMPORT DONE in 4s 606ms.
Imported:
  1 nodes
  0 relationships
  1 properties
Peak memory usage: 43.90MiB
2022-02-07 11:13:14.527+0000 INFO  [o.n.i.b.ImportLogic] Import completed successfully, took 4s 606ms.
Imported:
  1 nodes
  0 relationships
  1 properties
```

```
2022-02-07 11:13:15.484+0000 INFO  [StoreCopy] Import summary: Copying of 100704 records took 8
seconds (12588 rec/s). Unused Records 100703 (99%) Removed Records 0 (0%)
2022-02-07 11:13:15.485+0000 INFO  [StoreCopy] ### Extracting schema ###
2022-02-07 11:13:15.485+0000 INFO  [StoreCopy] Trying to extract schema...
2022-02-07 11:13:15.606+0000 INFO  [StoreCopy] ... found 1 readable schema definitions. The following
can be used to recreate the schema:
2022-02-07 11:13:15.606+0000 INFO  [StoreCopy]

CREATE BTREE INDEX `index_5c0607ad` FOR (n:`Person`) ON (n.`name`) OPTIONS {indexProvider: 'native-
btree-1.0', indexConfig: {`spatial.cartesian-3d.min`: [-1000000.0, -1000000.0, -1000000.0],
`spatial.cartesian.min`: [-1000000.0, -1000000.0], `spatial.wgs-84.min`: [-180.0, -90.0],
`spatial.cartesian-3d.max`: [1000000.0, 1000000.0, 1000000.0], `spatial.cartesian.max`: [1000000.0,
1000000.0], `spatial.wgs-84-3d.min`: [-180.0, -90.0, -1000000.0], `spatial.wgs-84-3d.max`: [180.0,
90.0, 1000000.0], `spatial.wgs-84.max`: [180.0, 90.0]}}
2022-02-07 11:13:15.606+0000 INFO  [StoreCopy] You have to manually apply the above commands to the
database when it is started to recreate the indexes and constraints. The commands are saved to
/Users/renetapopova/neo4j/cc-4.4.0/core1/logs/neo4j-admin-copy-2022-02-07.11.13.05.log as well for
reference.
```

2. On **each cluster member**, run the following command to verify that database has been successfully copied.

```
ls -al ../data/databases
```

```
total 0
drwxr-xr-x@  6 renetapopova  staff   192 Feb  7 11:11 .
drwxr-xr-x@  8 renetapopova  staff   256 Feb  7 10:36 ..
drwxr-xr-x  34 renetapopova  staff  1088 Feb  7 11:12 compactdb
drwxr-xr-x  38 renetapopova  staff  1216 Feb  7 10:39 neo4j
-rw-r--r--   1 renetapopova  staff     0 Feb  7 10:36 store_lock
drwxr-xr-x  39 renetapopova  staff  1248 Feb  7 10:39 system
```

Copying a database does not automatically create it. Therefore, it will not be visible if you do `SHOW DATABASES` in Cypher Shell or Neo4j Browser.

Create your compacted backup on one of the cluster members

You create the database copy **only on one of the cluster members** using the command `CREATE DATABASE`. The command is automatically routed to the leader, and from there, to the other cluster members.

1. On **one of the cluster members**, navigate to the */bin* folder and run the following command to log in to the Cypher Shell command-line console:

```
./cypher-shell -u neo4j -p password
```

2. Change the active database to `system`:

```
:USE system;
```

3. Create the `compactdb` database:

```
CREATE DATABASE compactdb;
```

```
0 rows available after 145 ms, consumed after another 0 ms
```

4. Verify that the `compactdb` database is online.

```
SHOW DATABASES;
```

```
+-----------------------------------------------------------------------------------------------------
-----------------------------+
| name        | aliases | access       | address          | role       | requestedStatus |
currentStatus | error | default | home  |
+-----------------------------------------------------------------------------------------------------
-----------------------------+
| "compactdb" | []      | "read-write" | "localhost:7687" | "follower" | "online"        | "online"
| ""    | FALSE   | FALSE |
| "compactdb" | []      | "read-write" | "localhost:7688" | "leader"   | "online"        | "online"
| ""    | FALSE   | FALSE |
| "compactdb" | []      | "read-write" | "localhost:7689" | "follower" | "online"        | "online"
| ""    | FALSE   | FALSE |
| "neo4j"     | []      | "read-write" | "localhost:7687" | "leader"   | "online"        | "online"
| ""    | TRUE    | TRUE  |
| "neo4j"     | []      | "read-write" | "localhost:7688" | "follower" | "online"        | "online"
| ""    | TRUE    | TRUE  |
| "neo4j"     | []      | "read-write" | "localhost:7689" | "follower" | "online"        | "online"
| ""    | TRUE    | TRUE  |
| "system"    | []      | "read-write" | "localhost:7687" | "follower" | "online"        | "online"
| ""    | FALSE   | FALSE |
| "system"    | []      | "read-write" | "localhost:7688" | "follower" | "online"        | "online"
| ""    | FALSE   | FALSE |
| "system"    | []      | "read-write" | "localhost:7689" | "leader"   | "online"        | "online"
| ""    | FALSE   | FALSE |
+-----------------------------------------------------------------------------------------------------
-----------------------------+

9 rows
ready to start consuming query after 21 ms, results consumed after another 29 ms
```

5. On one of the cluster members, change your active database to `compactdb` and recreate the schema using the output from the `neo4j-admin copy` command.

```
CREATE BTREE INDEX `index_5c0607ad` FOR (n:`Person`) ON (n.`name`) OPTIONS {indexProvider: 'native-
btree-1.0', indexConfig: {`spatial.cartesian-3d.min`: [-1000000.0, -1000000.0, -1000000.0],
`spatial.cartesian.min`: [-1000000.0, -1000000.0], `spatial.wgs-84.min`: [-180.0, -90.0],
`spatial.cartesian-3d.max`: [1000000.0, 1000000.0, 1000000.0], `spatial.cartesian.max`: [1000000.0,
1000000.0], `spatial.wgs-84-3d.min`: [-180.0, -90.0, -1000000.0], `spatial.wgs-84-3d.max`: [180.0,
90.0, 1000000.0], `spatial.wgs-84.max`: [180.0, 90.0]}};
```

```
0 rows
ready to start consuming query after 95 ms, results consumed after another 0 ms
Added 1 indexes
```

6. On each cluster member, log in to the Cypher Shell command-line console, change the active database to `compactdb`, and verify that the index has been successfully created:

```
CALL db.indexes;
```

```
+-----------------------------------------------------------------------------------------------------
-----------------------------------------+
| id | name               | state    | populationPercent | uniqueness  | type     | entityType |
labelsOrTypes | properties | provider          |
+-----------------------------------------------------------------------------------------------------
-----------------------------------------+
| 1  | "index_343aff4e" | "ONLINE" | 100.0             | "NONUNIQUE" | "LOOKUP" | "NODE"     | []
| []         | "token-lookup-1.0" |
| 2  | "index_5c0607ad" | "ONLINE" | 100.0             | "NONUNIQUE" | "BTREE"  | "NODE"     |
["Person"]    | ["name"]   | "native-btree-1.0" |
+-----------------------------------------------------------------------------------------------------
-----------------------------------------+

2 rows
ready to start consuming query after 31 ms, results consumed after another 5 ms
```

7. Verify that all the data has been successfully copied. In this example, there should be one node.

```
MATCH (n) RETURN n.name;
```

```
+--------+
| n.name |
+--------+
| "John" |
+--------+

1 row available after 106 ms, consumed after another 2 ms
```

8. Exit the Cypher Shell command-line console.

```
:exit;

Bye!
```

You can now compare the store size with the size of the backed up database.

9. On one of the cluster members, navigate to the `compactdb` database (*$core1_home/data/databases/compactdb*) and check the store size of the copied nodes and properties.

```
ls -alh
```

```
...
-rw-r--r--   1 username  staff   736K Feb  7 16:00 neostore.nodestore.db
-rw-r--r--   1 username  staff    16K Feb  7 16:00 neostore.propertystore.db
...
```

The output reports that the node store and the property store now occupy only `736K` and `16K` respectively, compared to the previous `1.4M` and `3.9M`.

7.11. Indexes in Neo4j 4.x

This section contains necessary information about indexes in different versions of Neo4j.

Indexes are automatically updated to the most recent index provider and rebuilt. The only exception are the 3.5 indexes that were previously using Lucene for single-property strings. They are upgraded to a fallback version that uses Lucene for those properties, and they have to be rebuilt. For details on creating, using, and dropping b-tree indexes, see Cypher Manual 4.0 → Indexes for search performance.

> The `neo4j-admin copy` command does not copy the *schema* store. Therefore, if following the path Neo4j 3.5 (Community or Enterprise) → Neo4j 4.x Enterprise, you have to recreate all indexes and constraints by running the Cypher commands that the `neo4j-admin copy` operation outputs. For more information on how to do that, see Tutorial: Back up and copy a database.

7.11.1. Index providers changes

The following table provides details on the index providers mapping:

Index provider in 3.5	Index provider in 4.x
`native-btree-1.0`	`native-btree-1.0`
`lucene+native-2.0`	`native-btree-1.0`
`lucene+native-1.0`	`lucene+native-3.0`
`lucene-1.0`	`lucene+native-3.0`

7.11.2. Memory allocation

Depending on what index providers were used previously, the upgrade of indexes may change the distribution of memory utilization. In a database with many indexes, a significant amount of memory may have been reserved for Lucene. Therefore, it is recommended to run `neo4j-admin memrec --database` before and after migration, and adjust memory settings in accordance with the findings. It might be necessary to allocate some of the memory reserved to Lucene back to the page cache. For a detailed description on how memory is allocated and used, refer to Operations Manual → Memory configuration and Operations Manual 4.0 → Capacity planning in particular.

7.11.3. Property size limits

Between 3.5 and 4.x, some changes have been made in how large a key (a property) can be in a b-tree index. These changes are only relevant for indexes that use the index providers `lucene-1.0` or `lucene+native-1.0` in 3.5, and hold large strings or large arrays.

Property size limits for index providers in 3.5

Index and property type	`lucene-1.0`	`lucene+native-1.0`	`lucene+native-2.0`	`native-btree-1.0`
Single property strings	~32kB	~32kB	~4kB	~4kB

Index and property type	`lucene-1.0`	`lucene+native-1.0`	`lucene+native-2.0`	`native-btree-1.0`
Single property arrays	~32kB	~32kB	~4kB	~4kB
Multiple properties	~32kB per property	~32kB per property	~4kB for properties combined	~4kB for properties combined

Property size limits for index providers in 4.x

Index and property type	`lucene+native-3.0`	`native-btree-1.0`
Single property strings	~32kB	~8kB
Single property arrays	~8kB	~8kB
Multiple properties	~8kB for properties combined	~8kB for properties combined

There is no workaround in 4.x to index arrays or multiple properties larger than ~8kB. For more information about what happens when this property size limit is reached, see Operations Manual 4.0 → Limitations on key size.

7.11.4. Explicit indexes

Support for explicit indexes has been removed from 4.0 onwards and the functionality has been replaced by full-text indexes. For details, see Cypher Manual 4.0 → Indexes to support full-text search.

7.11.5. Token lookup indexes

With Neo4j 4.3, a new index is introduced, the *token lookup index*. This index is present by default in all databases created in 4.3 and later and applies to both node labels and relationship types. See Operations Manual 4.3 → Token lookup indexes for more information.

The node lookup index evolved from the label scan store, which has been present in various forms for a long time. There are many advantages of turning a label scan store into an index, but it must be noted that the node lookup index is droppable, unlike the label scan store.

The node lookup index is the most important index in the database and greatly improves the performance of Cypher queries, Core API operations as well as the population of other indexes. Dropping this index leads to severe performance degradation for most workloads. Therefore, carefully consider the consequences before doing so.

Accordingly, any index cleaning scripts that list and drop all indexes also include the node lookup index and should thus also be used with extreme caution.

7.11.6. Changes to configuration settings in Neo4j 4

This section describes all changes and deprecations for the Neo4j configuration settings per version.

Neo4j 4.4

Configuration setting	Change	Comment
dbms.connector.bolt.connection_keep_alive	New	
dbms.connector.bolt.connection_keep_alive_for_requests	New	
dbms.connector.bolt.connection_keep_alive_probes	New	
dbms.connector.bolt.connection_keep_alive_streaming_scheduling_interval	New	

Neo4j 4.3

Configuration setting	Change	Comment
browser.retain_editor_history	New	
causal_clustering.max_raft_channels	New	
causal_clustering.store_copy_parallelism	New	
dbms.backup.incremental.strategy	New	
dbms.clustering.enable	New	
dbms.databases.default_to_read_only	New	
dbms.databases.read_only	New	
dbms.databases.writable	New	
dbms.directories.licenses	New	
dbms.logs.debug.format	New	
dbms.logs.default_format	New	
dbms.logs.query.format	New	
dbms.logs.query.max_parameter_length	New	
dbms.logs.query.obfuscate_literals	New	
dbms.logs.query.plan_description_enabled	New	
dbms.logs.query.transaction.enabled	New	
dbms.logs.query.transaction_id.enabled	New	
dbms.logs.query.transaction.threshold	New	
dbms.logs.security.format	New	

Configuration setting	Change	Comment
dbms.logs.user.format	New	
dbms.panic.shutdown_on_panic	New	
dbms.routing.client_side.enforce_for_domains	New	
dbms.routing.default_router	New	
dbms.security.ldap.authentication.attribute	New	
dbms.security.ldap.authentication.search_for_attribute	New	
dbms.security.ldap.authorization.access_permitted_group	New	
dbms.store.files.preallocate	New	
dbms.security.procedures.roles	Deprecated	Replaced by `EXECUTE BOOSTED privileges` (for both procedures and user-defined functions) and will be removed in version 5.x.
dbms.security.procedures.default_allowed	Deprecated	Replaced by `EXECUTE BOOSTED privileges` (for both procedures and user-defined functions) and will be removed in version 5.x.

Neo4j 4.2

Configuration setting	Change	Comment
causal_clustering.cluster_allow_reads_on_leader	New	
causal_clustering.connect_randomly_to_server_group	New	
causal_clustering.kubernetes.cluster_domain	New	
causal_clustering.leadership_priority_group	New	
dbms.connector.bolt.ocsp_stapling_enabled	New	
dbms.directories.cluster_state	New	
dbms.directories.script.root	New	
dbms.dynamic.setting.allowlist	New	Replacing `dbms.dynamic.setting.whitelist`
dbms.http_enabled_modules	New	
dbms.memory.pagecache.flush.buffer.enabled	New	
dbms.memory.pagecache.flush.buffer.size_in_pages	New	
dbms.memory.pagecache.warmup.preload.allowlist	New	Replacing `dbms.memory.pagecache.warmup.preload.whitelist`
dbms.memory.transaction.database_max_size	New	
dbms.security.http_auth_allowlist	New	Replacing `dbms.security.http_auth_whitelist`

Configuration setting	Change	Comment
dbms.security.procedures.allowlist	New	
metrics.csv.rotation.compression	New	
metrics.filter	New	
metrics.namespaces.enabled	New	
metrics.jmx.enabled	Changed	The change is introduced in Neo4j 4.2.2.
dbms.dynamic.setting.whitelist	Deprecated	
dbms.memory.pagecache.warmup.preload.whitelist	Deprecated	
dbms.security.http_auth_whitelist	Deprecated	
dbms.logs.security.rotation.delay	Deprecated	Redundant after replacing the custom logging framework with Log4j.
dbms.logs.user.rotation.delay	Deprecated	Redundant after replacing the custom logging framework with Log4j.
dbms.logs.debug.rotation.delay	Deprecated	Redundant after replacing the custom logging framework with Log4j.
dbms.security.procedures.default_allowed	Deprecated	Replaced by Cypher: `EXECUTE PROCEDURE`, `EXECUTE BOOSTED PROCEDURE`, `EXECUTE FUNCTION`, and `EXECUTE BOOSTED FUNCTION` privileges.
dbms.security.procedures.roles	Deprecated	Replaced by Cypher: `EXECUTE PROCEDURE`, `EXECUTE BOOSTED PROCEDURE`, `EXECUTE FUNCTION`, and `EXECUTE BOOSTED FUNCTION` privileges.
causal_clustering.connect-randomly-to-server-group	Removed	
dbms.memory.transaction.datababase_max_size	Removed	

Neo4j 4.1

Configuration setting	Change	Comment
causal_clustering.command_applier_parallelism	New	
causal_clustering.election_failure_detection_window	New	
causal_clustering.join_catch_up_max_lag	New	
causal_clustering.leader_failure_detection_windows	New	
causal_clustering.leadership_balancing	New	
causal_clustering.log_shipping_retry_timeout	New	
causal_clustering.raft_handler_parallelism	New	
dbms.allow_single_automatic_upgrade	New	
dbms.directories.dumps.root	New	

Configuration setting	Change	Comment
dbms.logs.query.early_raw_logging_enabled	New	
dbms.logs.query.parameter_full_entities	New	
dbms.memory.off_heap.block_cache_size	New	
dbms.memory.off_heap.max_cacheable_block_size	New	
dbms.memory.off_heap.max_size	New	
dbms.memory.pagecache.directio	New	
dbms.memory.pagecache.scan.prefetchers	New	
dbms.memory.tracking.enable	New	
dbms.memory.transaction.datababase_max_size	New	
dbms.memory.transaction.global_max_size	New	
dbms.memory.transaction.max_size	New	
dbms.reconciler.max_parallelism	New	
dbms.routing.advertised_address	New	
dbms.routing.driver.api	New	
dbms.routing.driver.connection.connect_timeout	New	
dbms.routing.driver.connection.max_lifetime	New	
dbms.routing.driver.connection.pool.acquisition_timeout	New	
dbms.routing.driver.connection.pool.idle_test	New	
dbms.routing.driver.connection.pool.max_size	New	
dbms.routing.driver.logging.level	New	
dbms.routing.enabled	New	
dbms.routing.listen_address	New	
dbms.upgrade_max_processors	New	
metrics.jvm.pause_time.enabled	New	
metrics.neo4j.pools.enabled	New	
causal_clustering.leader_election_timeout	Deprecated	This setting is moved and enhanced into `causal_clustering.leader_failure_detection_window` and `causal_clustering.election_failure_detection_window`.
cypher.query_max_allocations	Removed	
dbms.security.property_level.enabled	Removed	
dbms.tx_state.max_off_heap_memory	Removed	
dbms.tx_state.off_heap.block_cache_size	Removed	

Configuration setting	Change	Comment
dbms.tx_state.off_heap.max_cacheable_block_size	Removed	

Neo4j 4.0

Configuration setting	Change	Comment
causal_clustering.middleware.logging.level	New	
causal_clustering.replicated_lease_state_size	New	
causal_clustering.replication_leader_await_timeout	New	
causal_clustering.status_throughput_window	New	
causal_clustering.store_copy_chunk_size	New	
cypher.query_max_allocations	New	
dbms.backup.listen_address	New	
dbms.connector.bolt.unsupported_thread_pool_shutdown_wait_time	New	
dbms.default_advertised_address	New	
dbms.default_database	New	
dbms.default_listen_address	New	
dbms.directories.neo4j_home	New	
dbms.directories.transaction.logs.root	New	
dbms.dynamic.setting.whitelist	New	
dbms.max_databases	New	
dbms.memory.pagecache.warmup.enable	New	
dbms.memory.pagecache.warmup.preload	New	
dbms.memory.pagecache.warmup.preload.whitelist	New	
dbms.memory.pagecache.warmup.profile.interval	New	
dbms.reconciler.max_backoff	New	
dbms.reconciler.may_retry	New	
dbms.reconciler.min_backoff	New	
dbms.recovery.fail_on_missing_files	New	
dbms.routing_ttl	New	
dbms.security.authorization_providers	New	
dbms.security.authentication_providers	New	
dbms.security.http_auth_whitelist	New	
dbms.transaction.concurrent.maximum	New	

Configuration setting	Change	Comment
dbms.transaction.sampling.percentage	New	
dbms.transaction.tracing.level	New	
dbms.tx_log.preallocate	New	
fabric.database.name	New	
fabric.driver.api	New	
fabric.driver.connection.connect_timeout	New	
fabric.driver.connection.max_lifetime	New	
fabric.driver.connection.pool.acquisition_timeout	New	
fabric.driver.connection.pool.idle_test	New	
fabric.driver.connection.pool.max_size	New	
fabric.driver.logging.level	New	
fabric.routing.servers	New	
fabric.routing.ttl	New	
fabric.stream.buffer.low_watermark	New	
fabric.stream.buffer.size	New	
fabric.stream.concurrency	New	
metrics.jmx.enabled	New	
metrics.jvm.file.descriptors.enabled	New	
metrics.jvm.heap.enabled	New	
metrics.neo4j.database_operation_count.enabled	New	
metrics.neo4j.data.counts.enabled	New	
metrics.neo4j.logs.enabled	New	
metrics.neo4j.size.enabled	New	
bolt.ssl_policy	Removed	
causal_clustering.array_block_id_allocation_size	Removed	
causal_clustering.cluster_routing_ttl	Removed	
causal_clustering.database	Removed	
causal_clustering.disable_middleware_logging	Removed	
causal_clustering.expected_core_cluster_size	Removed	
causal_clustering.id_alloc_state_size	Removed	
causal_clustering.label_token_id_allocation_size	Removed	
causal_clustering.label_token_name_id_allocation_size	Removed	

Configuration setting	Change	Comment
causal_clustering.load_balancing.config	Removed	
causal_clustering.middleware_logging.level	Removed	
causal_clustering.neostore_block_id_allocation_size	Removed	
causal_clustering.node_id_allocation_size	Removed	
causal_clustering.node_labels_id_allocation_size	Removed	
causal_clustering.property_id_allocation_size	Removed	
causal_clustering.property_key_token_id_allocation_size	Removed	
causal_clustering.property_key_token_name_id_allocation_size	Removed	
causal_clustering.read_replica_time_to_live	Removed	
causal_clustering.reconnection_backoff	Removed	
causal_clustering.relationship_group_id_allocation_size	Removed	
causal_clustering.relationship_id_allocation_size	Removed	
causal_clustering.relationship_type_token_id_allocation_size	Removed	
causal_clustering.relationship_type_token_name_id_allocation_size	Removed	
causal_clustering.replicated_lock_token_state_size	Removed	
causal_clustering.schema_id_allocation_size	Removed	
causal_clustering.ssl_policy	Removed	
causal_clustering.string_block_id_allocation_size	Removed	
dbms.active_database	Removed	
dbms.allow_format_migration	Removed	
dbms.backup.address	Removed	
dbms.backup.ssl_policy	Removed	
dbms.connectors.default_advertised_address	Removed	
dbms.connectors.default_listen_address	Removed	
dbms.directories.certificates	Removed	
dbms.directories.tx_log	Removed	
dbms.ids.reuse.types.override	Removed	
dbms.index_sampling.buffer_size	Removed	
dbms.logs.timezone	Removed	
dbms.procedures.kill_query_verbose	Removed	
dbms.security.auth_provider	Removed	

Configuration setting	Change	Comment
dbms.security.ha_status_auth_enabled	Removed	
dbms.security.http_authorization_classes	Removed	
dbms.security.property_level.blacklist	Removed	
dbms.ssl.policy.<policyname>.allow_key_generation	Removed	
dbms.ssl.policy.<policyname>.base_directory	Removed	
dbms.ssl.policy.<policyname>.ciphers	Removed	
dbms.ssl.policy.<policyname>.client_auth	Removed	
dbms.ssl.policy.<policyname>.private_key	Removed	
dbms.ssl.policy.<policyname>.public_certificate	Removed	
dbms.ssl.policy.<policyname>.revoked_dir	Removed	
dbms.ssl.policy.<policyname>.tls_versions	Removed	
dbms.ssl.policy.<policyname>.trust_all	Removed	
dbms.ssl.policy.<policyname>.trusted_dir	Removed	
dbms.ssl.policy.<policyname>.verify_hostname	Removed	
dbms.udc.enabled	Removed	
ha.allow_init_cluster	Removed	
ha.branched_data_copying_strategy	Removed	
ha.branched_data_policy	Removed	
ha.broadcast_timeout	Removed	
ha.configuration_timeout	Removed	
ha.data_chunk_size	Removed	
ha.default_timeout	Removed	
ha.election_timeout	Removed	
ha.heartbeat_interval	Removed	
ha.heartbeat_timeout	Removed	
ha.host.coordination	Removed	
ha.host.data	Removed	
ha.initial_hosts	Removed	
ha.internal_role_switch_timeout	Removed	
ha.join_timeout	Removed	
ha.learn_timeout	Removed	
ha.leave_timeout	Removed	

Configuration setting	Change	Comment
ha.max_acceptors	Removed	
ha.max_channels_per_slave	Removed	
ha.paxos_timeoutha.pull_batch_size	Removed	
ha.pull_interval	Removed	
ha.role_switch_timeout	Removed	
ha.server_id	Removed	
ha.slave_lock_timeout	Removed	
ha.slave_only	Removed	
ha.slave_read_timeout	Removed	
ha.tx_push_factor	Removed	
ha.tx_push_strategy	Removed	
https.ssl_policy	Removed	
metrics.neo4j.cluster.enabled	Removed	
metrics.neo4j.enabled	Removed	
metrics.neo4j.logrotation.enabled	Removed	
metrics.neo4j.network.enabled	Removed	
tools.consistency_checker.check_graph	Removed	
tools.consistency_checker.check_indexes	Removed	
tools.consistency_checker.check_index_structure	Removed	
tools.consistency_checker.check_label_scan_store	Removed	
tools.consistency_checker.check_property_owners	Removed	
ha.phase1_timeout	Removed	
ha.phase2_timeout	Removed	
ha.pull_batch_size	Removed	
ha.pull_interval	Removed	
ha.role_switch_timeout	Removed	
ha.server_id	Removed	
ha.slave_lock_timeout	Removed	
ha.slave_only	Removed	
ha.slave_read_timeout	Removed	
ha.tx_push_factor	Removed	
ha.tx_push_strategy	Removed	

Configuration setting	Change	Comment
https.ssl_policy	Removed	
metrics.neo4j.cluster.enabled	Removed	
metrics.neo4j.enabled	Removed	
metrics.neo4j.logrotation.enabled	Removed	
metrics.neo4j.network.enabled	Removed	
tools.consistency_checker.check_graph	Removed	
tools.consistency_checker.check_indexes	Removed	
tools.consistency_checker.check_index_structure	Removed	
tools.consistency_checker.check_label_scan_store	Removed	
tools.consistency_checker.check_property_owners	Removed	

7.11.7. Changes to procedures and functions in Neo4j 4

This section describes all changes and deprecations for the Neo4j procedures per version.

Neo4j 4.4

Name	Community Edition	Enterprise Edition	Comment
`dbms.killQueries()`	Yes	Yes	Deprecated Replaced by `TERMINATE TRANSACTIONS`.
`dbms.killQuery()`	Yes	Yes	Deprecated Replaced by `TERMINATE TRANSACTIONS`.
`dbms.killTransaction()`	Yes	Yes	Deprecated Replaced by `TERMINATE TRANSACTIONS`.
`dbms.killTransactions()`	Yes	Yes	Deprecated Replaced by `TERMINATE TRANSACTIONS`.
`dbms.listQueries()`	Yes	Yes	Deprecated Replaced by `SHOW TRANSACTIONS`.
`dbms.listTransactions()`	Yes	Yes	Deprecated Replaced by `SHOW TRANSACTIONS`.

Neo4j 4.3

Name	Community Edition	Enterprise Edition	Comment
`dbms.procedures()`	Yes	Yes	Deprecated Replaced by `SHOW PROCEDURES`.
`dbms.functions()`	Yes	Yes	Deprecated Replaced by `SHOW FUNCTIONS`.
`db.createIndex()`	Yes	Yes	Deprecated Replaced by `OPTIONS` of the `CREATE INDEX` command. To be removed in 5.x.

Name	Community Edition	Enterprise Edition	Comment
`db.createNodeKey()`	No	Yes	Deprecated Replaced by `OPTIONS` of the `CREATE CONSTRAINT ... IS NODE KEY` command. To be removed in 5.x.
`db.createUniquePropertyConstraint()`	Yes	Yes	Deprecated Replaced by `OPTIONS` of the `CREATE CONSTRAINT ... IS UNIQUE` command. To be removed in 5.x.
`db.indexes()`	Yes	Yes	Deprecated Replaced by `SHOW INDEXES`. To be removed in 5.x.
`db.indexDetails()`	Yes	Yes	Deprecated Replaced by `SHOW INDEXES YIELD *`. To be removed in 5.x.
`db.index.fulltext.createNodeIndex()`	Yes	Yes	Deprecated Replaced by `CREATE FULLTEXT INDEX`.
`db.index.fulltext.createRelationshipIndex()`	Yes	Yes	Deprecated Replaced by `CREATE FULLTEXT INDEX`.
`db.index.fulltext.drop()`	Yes	Yes	Deprecated Replaced by `DROP INDEX ...`.
`db.constraints()`	Yes	Yes	Deprecated Replaced by `SHOW CONSTRAINTS`. To be removed in 5.x.
`db.schemaStatements()`	Yes	Yes	Deprecated Replaced by `SHOW INDEXES YIELD *` and `SHOW CONSTRAINTS YIELD *`. To be removed in 5.x.

Neo4j 4.2

Name	Community Edition	Enterprise Edition	Comment
`dbms.cluster.quarantineDatabase()`	No	Yes	New
`dbms.cluster.readReplicaToggle()`	No	Yes	New
`dbms.scheduler.jobs()`	No	Yes	New
`dbms.scheduler.failedJobs()`	No	Yes	New
`dbms.functions()`	No	Yes	Changed Signature changed.
`dbms.killConnections()`	Yes	Yes	Changed Available also for Community Edition.
`dbms.killQueries()`	Yes	Yes	Changed Available also for Community Edition.
`dbms.killQuery()`	Yes	Yes	Changed Available also for Community Edition.
`dbms.killTransaction()`	Yes	Yes	Changed Available also for Community Edition.
`dbms.killTransactions()`	Yes	Yes	Changed Available also for Community Edition.
`dbms.listConnections()`	Yes	Yes	Changed Available also for Community Edition.
`dbms.listTransactions()`	Yes	Yes	Changed Available also for Community Edition.

Name	Community Edition	Enterprise Edition	Comment
`db.listLocks()`	Yes	Yes	Changed Signature changed to `db.listLocks() :: (mode :: STRING?, resourceType :: STRING?, resourceId :: INTEGER?, transactionId :: STRING?)`.
`db.constraints()`	Yes	Yes	Deprecated Replaced by `SHOW CONSTRAINTS`.
`db.createIndex()`	Yes	Yes	Deprecated Replaced by `CREATE INDEX`.
`db.createNodeKey()`	No	Yes	Deprecated Replaced by `CREATE CONSTRAINT ... IS NODE KEY`.
`db.createUniquePropertyConstraint()`	Yes	Yes	Deprecated Replaced by `CREATE CONSTRAINT ... IS UNIQUE`.
`db.indexDetails()`	Yes	Yes	Deprecated Replaced by `SHOW INDEXES VERBOSE OUTPUT`.
`db.indexes()`	Yes	Yes	Deprecated Replaced by `SHOW INDEXES`.
`db.schemaStatements()`	Yes	Yes	Deprecated Replaced by `SHOW INDEXES VERBOSE OUTPUT` and `SHOW CONSTRAINTS VERBOSE OUTPUT`.

Neo4j 4.1

Name	Community Edition	Enterprise Edition	Comment
`dbms.listPools()`	No	Yes	New
`dbms.upgrade()`	Yes	Yes	New
`dbms.upgradeStatus()`	Yes	Yes	New
`dbms.cluster.setDefaultDatabase()`	No	Yes	New
`dbms.listQueries()`	No	Yes	Changed The queryId procedure format has changed, and no longer includes the database name. For example, mydb-query-123 is now query-123.
`db.index.fulltext.queryNodes()`	Yes	Yes	Changed Skip and limit options added, e.g. `{skip: 10, limit: 100}`.
`db.index.fulltext.queryRelationships()`	Yes	Yes	Changed Skip and limit options added, e.g. `{skip: 10, limit: 100}`.
`db.constraints()`	Yes	Yes	Changed Signature changed. Added a `details` field: `db.constraints() :: (name :: STRING?, description :: STRING?, details :: STRING?)`.

Name	Community Edition	Enterprise Edition	Comment
`dbms.listTransactions()`	No	Yes	Changed Signature changed. Added an `estimatedUsedHeapMemory` field: `dbms.listTransactions() :: (transactionId :: STRING?, username :: STRING?, metaData :: MAP?, startTime :: STRING?, protocol :: STRING?, clientAddress :: STRING?, requestUri :: STRING?, currentQueryId :: STRING?, currentQuery :: STRING?, activeLockCount :: INTEGER?, status :: STRING?, resourceInformation :: MAP?, elapsedTimeMillis :: INTEGER?, cpuTimeMillis :: INTEGER?, waitTimeMillis :: INTEGER?, idleTimeMillis :: INTEGER?, allocatedBytes :: INTEGER?, allocatedDirectBytes :: INTEGER?, pageHits :: INTEGER?, pageFaults :: INTEGER?, connectionId :: STRING?, initializationStackTrace :: STRING?, database :: STRING?, estimatedUsedHeapMemory :: INTEGER?)`.

Neo4j 4.0

Name	Community Edition	Enterprise Edition	Comment
`dbms.cluster.overview()`	No	Yes	New
`dbms.cluster.protocols()`	No	Yes	New
`dbms.cluster.role()`	No	Yes	New
`dbms.security.changePassword()`	Yes	Yes	Deprecated Replaced by `ALTER CURRENT USER SET PASSWORD`.
`dbms.security.activateUser()`	No	Yes	Deprecated Replaced by `ALTER USER`.
`dbms.security.addRoleToUser()`	No	Yes	Deprecated Replaced by `GRANT ROLE TO USER`.
`dbms.security.changePassword()`	Yes	Yes	Deprecated Replaced by `ALTER CURRENT USER SET PASSWORD`.
`dbms.security.changeUserPassword()`	No	Yes	Deprecated Replaced by `ALTER USER`.
`dbms.security.createRole()`	No	Yes	Deprecated Replaced by `CREATE ROLE`.
`dbms.security.createUser()`	Yes	Yes	Deprecated Replaced by `CREATE USER`.
`dbms.security.deleteUser()`	Yes	Yes	Deprecated Replaced by `DROP USER`.
`dbms.security.listRoles()`	No	Yes	Deprecated Replaced by `SHOW ROLES`.
`dbms.security.listRolesForUser()`	No	Yes	Deprecated Replaced by `SHOW USERS`.
`dbms.security.listUsers()`	Yes	Yes	Deprecated Replaced by `SHOW USERS`.
`dbms.security.listUsersForRole()`	No	Yes	Deprecated Replaced by `SHOW ROLES WITH USERS`.
`dbms.security.removeRoleFromUser()`	No	Yes	Deprecated Replaced by `REVOKE ROLE FROM USER`.

Name	Community Edition	Enterprise Edition	Comment
`dbms.security.suspendUser()`	No	Yes	Deprecated Replaced by `ALTER USER`.

License

[1] The calculations are based on `MB/s = (IOPS * B) ÷ 10^6`, where `B` is the block size in bytes; in the case of Neo4j, this is `8000`. GB/hour can then be calculated from `(MB/s * 3600) ÷ 1000`.
[2] The calculations are based on `MB/s = (IOPS * B) ÷ 10^6`, where `B` is the block size in bytes; in the case of Neo4j, this is `8000`. GB/hour can then be calculated from `(MB/s * 3600) ÷ 1000`.