NEO4J 3.5.X TO 4.X MIGRATION

TIPS AND BEST PRACTICES

Contents

1. Introduction

This document aims to serve as a best practice guide for a migration from Neo4j 3.5.x to Neo4j 4.x. It is not intended to be a replacement for the current documentation, in fact, this document will often point to (or reference) said documentation. You can think of it as a *one-stop-shop* or starting point for everything related to going from a 3.5.x to a 4.x deployment. From this document, you should be able to read and/or find everything you need to successfully migrate your Neo4j deployment to take advantage of the most up-to-date features, and the extended support which comes from running the most recent versions of Neo4j.

Both this document and associated references were written for:

- the engineer performing the Neo4j production migration.
- the operations engineer supporting and maintaining the Neo4j production database.
- the enterprise architect researching database migration.
- the infrastructure architect planning the Neo4j production migration.
- the enterprise data security manager responsible for the company's strategy for role-based access control.

Carefully reading them is essential to make sure your upgrade runs smoothly and frictionless.

As usual, the Neo4j support team are ready to help you with these operations so feel free to reach out to our support team if you run into any problem or have any questions about the upgrade process (https://support.neo4j.com/hc/en-us).

Furthermore, our Professional Services team has packages ready to help you out with migrating your Neo4j deployments. Please refer to section 5. Professional Services of this guide to learn more.

2. Important information

2.1 UNDERSTANDING UPGRADES AND MIGRATIONS

An **upgrade** is the process of upgrading an existing Neo4j deployment to a newer **minor** or **release** version of Neo4j, when such process does not require changes to the configuration or to the applications that use Neo4j. For example:

- Between adjacent patch releases within the same major and minor version, e.g., 3.5.**0** to 3.5.**1**, 4.2.**1** to 4.2.**2**.
- Between non-adjacent patch releases within the same major and minor version, e.g., 3.5.**0** to 3.5.**10**, 4.2.**1** to 4.2.**3**.
- Between adjacent minor versions, e.g., 4.**0**.0 to 4.**1**.3.

A **migration** is the process of migrating an existing Neo4j deployment to a newer **major** Neo4j version, when such a process requires a review of the configuration(s) and the applications that use Neo4j. For example:

- Between major versions, e.g., from **3**.5.13 to **4**.0.10.

Going from Neo4j 3.5.x to 4.x falls under the **migration** category and therefore a **major** Neo4j version, with a great deal of changes so caution is advised. Neo4j has put together a 4.0 migration guide which should be used alongside this document.

2.2 SUPPORTED VERSIONS

Keeping track of supported versions is super simple. For reference and planning purposes, Neo4j keeps an updated knowledge base article with a table representing the list of the currently supported Neo4j releases, release date, and date at which that release will no longer be supported. It also includes the compatible driver versions for each release.

2.3 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 special database called the **system database**. We have a great article giving us an overview of the system database which you should definitely take some time to read. For the intent and purpose of this guide, being aware of the existence of the system database beforehand will be useful.

3. Neo4j migration

3.1 SUPPORTED PATHS

Neo4j supports two paths for migrating/upgrading from an earlier version of Neo4j to the latest:

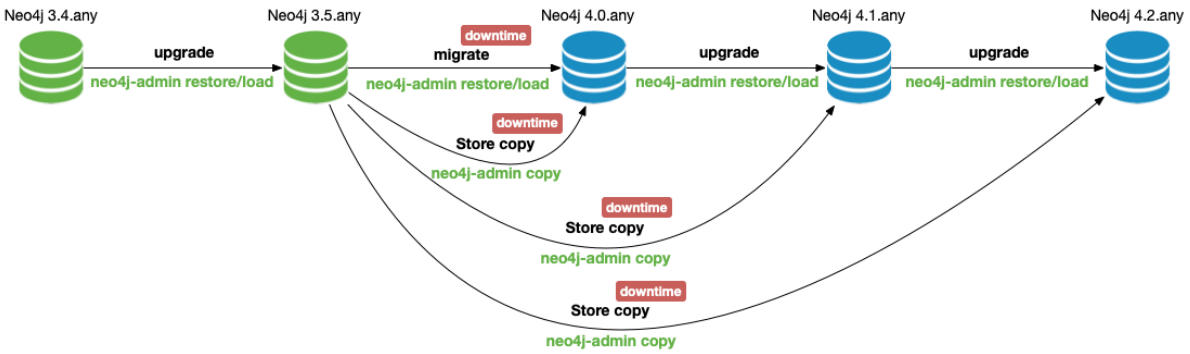
- **sequential** — 3.5.any → 4.0.any → 4.1.any → 4.2.any
- **direct** — 3.5.any → 4.x.any

When choosing which one is best there is not a *one size fits* all solution. The **sequential** path tends to be used when customers are relatively up-to-date with Neo4j's release cycle plus it allows for zero downtime between minor and patch versions. If on the other hand you have been unable to upgrade as often, you may use the **direct** path to skip some hops and go straight to the most recent version.

There are some extra items to consider regarding the **direct** path:

- The direct path method creates a **fresh new store** which will be **compacted** for optimization. This will result in a store size reduction, especially in case of write intensive scenarios. In some cases, we observed a 10x reduction of the store size.
- Using the direct path also allows you to migrate to the new aligned store format which delivers performance optimizations when you can't fit 100% of your graph in memory.
- Using the direct path will **always** require downtime.
- Indexes are **not migrated automatically** when using the direct path.

You can understand the migration flow below:



3.2 CONSIDERATIONS AND PLANNING

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

3.2.1 WHAT NEEDS TO BE MIGRATED

Details will follow below but, in an extremely high level, these are the components that need to be upgraded:

- The Neo4j product.
- The store formats (the file formats on disk, which is auto-upgraded at server startup).
- The system database schema, which restructures the contents of that database (this will be 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.

3.2.2 DOWNTIME

When **migrating** to a newer Neo4j major version, only **offline** migrations are supported and it will therefore require planning for some downtime. This holds true for the scope of this document which specifically targets migrations from 3.5x to 4.x.

Plan for this accordingly. Because each case is unique, Neo4j suggests running **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'll need.

Please refer to section 4.4 Tips on how to reduce downtime for notes on server availability and tips on how to reduce downtime during the migration.

3.2.3 DISK SPACE CONSIDERATIONS

A migration requires substantial free disk space, as it makes an entire copy of the database and creates temporary files. We also recommend taking a backup of your production store in case something goes wrong and you need to revert the process.

Because of the reasons listed above, we recommend reserving **two times** the size of the database directory (**three times** in case you take a backup beforehand) for migration purposes.

3.2.4 DURATION

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

- How big is your store?
- Do you need to upgrade the store format?
- Do indexes need to be migrated?
- How big are your indexes?

- How fast is your hardware?

Because of this, Neo4j's recommendation is to prepare an environment where you can test the migration process back-to-back. This will give you more accurate timings than you can ever estimate and can give you an idea of what to expect in terms of duration for the process.

In recent customer migrations, the data migration part of a migration process (**not** the entire process) took around 50~55 minutes for a 600GB store.

3.2.5 JAVA VERSION

In a short sentence, the Neo4j 4.x requires Java **11**. This is a change from the previous required Java **8** on 3.x. This is especially important if you have any other Java applications running on that machine so you need to make sure those applications are compatible with Java 11 as well. Alternatively, you should configure to run multiple JDKs on the same machine.

3.2.6 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 → Administrative commands.

3.2.7 CONFIGURATION CHANGES

There have been changes in some configuration names. The old names have been deprecated and will be removed in later versions of Neo4j. It is advised to adjust them straight away to make sure you are bulletproof for the future. You can check the changes to the configuration settings in the Migration Guide 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. We will need to add these configurations to the new **neo4j.conf** file from Neo4j 4.x.

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

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

3.2.8 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's worth checking the impact of the log format changes.

You can check the Logging section of our Operations Manual to learn about all the available logs and how to configure them, including log rotation and retention.

3.2.9 METRICS

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

1. The metrics that are enabled by default have been changed in the 4.2 release. Any specific metrics that you want to be enabled must be specified in the configuration.
2. With the introduction of multiple databases, the rules for naming metrics have changed to include the database name prefix.
3. Namespaces for metrics were also introduced (but are disabled by default).

If 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:

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

3.2.10 EMBEDDED DEPLOYMENTS

For users with embedded deployments (when you include Neo4j in your project), Neo4j 4.2 now supports running a Causal Cluster in embedded mode. However, coming from Neo4j HA embedded, there are a small number of changes required. You can learn how to embed Neo4j in your Java application (and using a Causal Cluster!) in our Java Developer Reference.

3.2.11 APPLICATION CODE

Depending on how your application is interacting with Neo4j, you should be prepared to review your application code. As you can imagine each case is different, but a Neo4j major version will **likely require** changes to the application code at some point in the process. 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 surface changes that may affect your application and Neo4j can be found in the Migration Guide.

3.2.12 DRIVERS

Neo4j's official drivers had some significant and breaking changes you need to be aware of so – as with your application code – the driver implementation needs to be reviewed.

First of all, you should have a look at the Migrate Neo4j Drivers section of the Migration Guide. This will have all information and list all the breaking changes that were introduced. Of all breaking changes, I will list some key ones which caused some confusion amongst our user base:

- 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 will go 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 (there are no releases between 1.7 and 4.0, ie: versions 1.8, 2.x or 3.x **do not exist**).
- The driver's default configuration for encrypted is now false (meaning that driver will only attempt plain text connections **by default**).
- When encryption is explicitly enabled connections with holding **self-signed certificates will now 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/or 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 Neo4j Drivers section of the Migration Guide mentioned above. 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 Driver 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).

3.2.13 PLUGINS (INCLUDING CUSTOM PLUGINS)

Take note of the plugins you are using and make sure they are compatible with Neo4j 4.x. Below are the most commonly used plugins/plugin types:

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

TIP**: to quickly check your APOC version you can run **RETURN apoc.version();

- If you are using **Neo4j Bloom** or our **Graph Data Science Library** (**GDSL**), you can find the most recent versions for these products in our 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 our product suite is compatible with the 4.x series so the recommendation is simply to 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, as described in the previous section. The several changes in Neo4j 4.x can impact the behaviour of these custom plugins so it is highly advised to plan some time for this as well.

3.2.14 SDN+OGM/SDN RX

Our SDN team has been busy, and this resulted in quite a few changes on the SDN side as well. Our latest release, SDN 6 (which comes included with OGM) is an ongoing effort to create the next generation of Spring Data Neo4j, with full reactive support and lightweight mapping.

If you are using SDN/OGM, our SDN team prepared documentation about migrating from SDN+OGM to SDN. In here you will find information about changes, known issues and actual migration steps.

As an extra, below you can find the *15 Commandments of SDN and OGM* which is a list compiled by our SDN team and quickly answers a lot of questions usually asked about SDN migration and Neo4j compatibility:

1. Neo4j the database requires Java 11 to run.
2. Neo4j the Java driver or connector, if you wish, can have a JDK 8 baseline.
3. So does Spring Framework.
4. So does Spring Boot.
5. So does Spring Data.
6. Spring Data Neo4j 5 is based on Neo4j-OGM.
7. Spring Data Neo4j 6 does not need Neo4j-OGM. It has all of it built in.
8. SDN 6 is 100% compatible with Neo4j 3.5 up to 4.2.
9. We do not recommend using Neo4j-OGM to bring up an embedded instance
10. SDN 6 cannot bring up an embedded instance
11. SDN 6 can use an embedded instance created as bean in an application context via local bolt connection.
12. SDN 5 + OGM is also compatible with Neo4j 4.0+
13. SDN 5 + OGM cannot bring up an embedded instance itself, but can use an existing one via local bolt connection.
14. Unless you want an embedded instance there is no need to upgrade to Java 11.
15. Of course, we do not recommend starting new projects on Java 8.

3.2.15 OTHER 3RD PARTY SOFTWARE AND/OR TOOLS

Be mindful of any other 3rd party software and/or 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 will need to **revise** these as there have been changes in metrics (as mentioned in section 3.2.9 Metrics) and our operational tools now account for multiple databases. We therefore recommend **reviewing all scripts/tools/3rd party software** and make sure they are prepared and compatible with Neo4j 4.x.

3.3 PRE-MIGRATION STEPS

3.3.1 REVIEW RELEASE NOTES

Each version of Neo4j introduces numerous improvements and features. If you are waiting for a specific feature or fix or just go get an understanding of what we've developed, we recommend always reviewing the Release Notes prior to a migration.

3.3.2 PREREQUISITES

Based on what was identified in the considerations and planning section, make sure all necessary activities are done/in place/planned (ie: Java version, application code, plugins, etc).

3.4 MIGRATION STEPS – SEQUENTIAL PATH

3.4.1 SINGLE INSTANCE

3.4.1.1 SHUTDOWN

Because a migration requires the database to be offline, the first step is to cleanly shutdown the database if it's still running:

```
$NEO4J_HOME/bin/neo4j stop

or

systemctl stop neo4j
```

It is important to verify that the database shutdown process finished successfully, and the database was cleanly shutdown. You can check the **neo4j.log** for these log messages for confirmation:

```
2021-04-30 13:47:27.220+0000 INFO  Neo4j Server shutdown initiated by request
```

```
2021-04-30 13:47:27.229+0000 INFO  Stopping...
2021-04-30 13:47:32.420+0000 INFO  Stopped.
```

3.4.1.2 BACKUP

We will need to backup the following components:

- **neo4j.conf**
- Contents of **$NEO4J_HOME/data/dbms** (if there are any native users)
- All the files used for encryption i.e. private key, public certificate, and the contents of the **trusted** and **revoked** directories (located in **$NEO4J_HOME/certificates/***)
- Neo4j store

Because the database is now offline, we are going to use the **neo4j-admin dump** command to create an offline backup of your store (if you are running a Debian/RPM distribution, you can skip this step):

```
$NEO4J_HOME/bin/neo4j-admin dump --to=$BACKUP_DESTINATION
```

This will create a file called **<db_name>.dump** (in whatever path you defined in **$BACKUP_DESTINATION**) which will be the backup we are going to use for the migration.

More information about the **3.5.x** dump tool can be found here.

3.4.1.3 INSTALL NEW VERSION

This step will allow you to install Neo4j 4.x binaries. Also – and as mentioned before – you will need to adjust any configuration settings that have changed from 3.5.x to 4.x.

At this point you should now have a backed up **neo4j.conf** from your old 3.5.x deployment, and also a list of **non-default/custom configurations** captured in the configuration changes section. With all of that, you need to choose one of the following methods, specific to your technology:

- If using a tarball or zip file for installation:
  - Untar or unzip Neo4j 4.x.
  - Apply configuration changes:
    - Review your backed up **neo4j.conf** and the changes to other configuration settings on our Migration Guide and update all applicable configuration settings.
    - Add any **non-default/custom configurations** previously present in your old **neo4j.conf** file.
    - Set **dbms.allow_upgrade=true** in your new 4.x **neo4j.conf** installation. Neo4j will **fail to start** without this configuration.

    - Move the files used for encryption from the old installation to the new one.
- If using a Debian or RPM distribution:
    - Set **dbms.allow_upgrade=true** in **neo4j.conf**.
    - Install Neo4j 4.x.
    - When prompted, review the differences between the **neo4j.conf** files of the previous version and Neo4j 4.x.
    - Add any **non-default/custom configurations** previously present in your old **neo4j.conf** file.
      Make sure to preserve **dbms.allow_upgrade=true**, as set in the instruction above. Neo4j will **fail to start** without this configuration.

***TIP**: When migrating, it is particularly important to note any custom values of the settings **dbms.directories.data** and **dbms.default_database**. In cluster installations, pay attention to **cluster-specific configuration settings**, which may be different on different cluster members.*

3.4.1.4 MIGRATE DATA AND START THE SERVER

Before migrating the data, we need to move the backup file to the data directory of Neo4j 4.x. (please note that this step is not applicable if you have **dbms.directories.data** pointing to a directory outside of **NEO4J_HOME**). To do this, we **need** to use the **neo4j-admin load** command. It is a requirement to use this tool 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=<DBNAME> --force
```

With the backup in place, we can initiate the data migration simply by starting Neo4j 4.x:

```
$NEO4J_HOME/bin/neo4j start

or

systemctl start neo4j
```

The database migration will take place during startup. Your indexes will also be automatically migrated to the most recent index provider during startup. Please note that indexes will **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.

***TIP**: The **neo4j.log** file contains valuable information on how many steps the migration will involve 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.*

If you used a **different database name** than what is set as the default database in neo4j.conf, you will need to **activate that database** before being able to use it. You can do this with the CREATE DATABASE command in the browser/cypher-shell:

```
CREATE DATABASE <DBNAME>;
```

3.4.2 CLUSTER

Migrating a cluster deployment requires some different and extra steps. The strategy for a cluster deployment is to complete a single migration on a single instance, as a standalone instance, then use the migrated store to **seed** the remaining members of the 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 will 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.

***IMPORTANT**: 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.*

3.4.2.1 SHUTDOWN

Because the migration will require the cluster to be offline, the first step is to cleanly shutdown **all instances** of your cluster:

```
$NEO4J_HOME/bin/neo4j stop

or

systemctl stop neo4j
```

It is important to verify that the database shutdown process finished successfully, and the database was cleanly shutdown. You can check the **neo4j.log** for these log messages for confirmation:

```
2021-04-30 13:47:27.220+0000 INFO  Neo4j Server shutdown initiated by request
2021-04-30 13:47:27.229+0000 INFO  Stopping...
2021-04-30 13:47:32.420+0000 INFO  Stopped.
```

3.4.2.2 BACKUP

We will need to backup the following components from **all instances**:

- **neo4j.conf**
- All the files used for encryption i.e. private key, public certificate, and the contents of the **trusted** and **revoked** directories (located in **$NEO4J_HOME/certificates/***)

Additionally, we will need to backup the following components from the **elected instance**:

- Contents of **$NEO4J_HOME/data/dbms** (if there are any native users)
- Neo4j store

Because the database is now offline, we are going to use the **neo4j-admin dump** command to create an offline backup of your store (if you are running a Debian/RPM distribution, you can skip this step):

```
$NEO4J_HOME/bin/neo4j-admin dump --to=$BACKUP_DESTINATION
```

This will create a file called **<db_name>.dump** (in whatever path you defined in **$BACKUP_DESTINATION**) which will be the backup we are going to use for the migration.

More information about the **3.5.x** dump tool can be found here.

3.4.2.3 UNBIND

Unbinding is necessary because this will eliminate all cluster state data and allow you to form a new cluster which is required as part of the migration. Run the following command on **all instances** of your cluster:

```
$NEO4J_HOME/bin/neo4j-admin unbind
```

3.4.2.4 INSTALL NEW VERSIONS

This step will allow you to install Neo4j 4.x binaries and should be completed on **all instances** of your cluster. Also – and as mentioned before – you will need to adjust any configuration settings that have changed from 3.5.x to 4.x.

At this point you should now have a backed up **neo4j.conf** from your old 3.5.x deployment, and also a list of **non-default/custom configurations** captured in the configuration changes section. With all of that, you need to choose one of the following methods, specific to your technology:

- If using a tarball or zip file for installation:
  - Untar or unzip Neo4j 4.x.
  - Apply configuration changes:
    - Review your backed up **neo4j.conf** and the changes to other configuration settings on our Migration Guide and update all applicable configuration settings.
    - Add any **non-default/custom configurations** previously present in your old **neo4j.conf** file.
  - Move the files used for encryption from the old installation to the new one.
- If using a Debian or RPM distribution:
  - Install Neo4j 4.x.
  - When prompted, review the differences between the **neo4j.conf** files of the previous version and Neo4j 4.x.
  - Add any **non-default/custom configurations** previously present in your old **neo4j.conf** file.

***TIP**: When migrating, it is particularly important to note any custom values of the settings **dbms.directories.data** and **dbms.default_database**. In cluster installations, pay attention to **cluster-specific configuration settings**, which may be different on different cluster members.*

3.4.2.5 EDIT NEO4J.CONF

This step should be performed on the **elected instance**.

You need to set two things on the neo4j.conf file:

- Set **dbms.allow_upgrade=true**. Neo4j will **fail to start** without this configuration so it's important to remember this.
- Set **dbms.mode=SINGLE**. We need to do this because a migration is a **single event** that needs to happen on a standalone server.

3.4.2.6 MIGRATE DATA

This step should be performed on the **elected instance**.

Before migrating the data, we need to move the backup file to the data directory of Neo4j 4.x. (please note that this step is not applicable if you have **dbms.directories.data** pointing to a directory outside of **NEO4J_HOME**). To do this, we **need** to use the **neo4j-admin load** command. It is a requirement to use these tools 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=<DBNAME> --force
```

With the backup in place, we can initiate the data migration simply by starting Neo4j 4.x:

```
$NEO4J_HOME/bin/neo4j start

or

systemctl start neo4j
```

The database migration will take place during startup. Your indexes will also be automatically migrated to the most recent index provider during startup. Please note that indexes will **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.

***TIP**: The **neo4j.log** file contains valuable information on how many steps the migration will involve 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.*

When the migration finishes, stop the server again:

```
$NEO4J_HOME/bin/neo4j stop

or

systemctl stop neo4j
```

3.4.2.7 REVERT NEO4J.CONF CHANGES

This step should be performed on the **elected instance**.

You need to revert the two configuration settings on the neo4j.conf file:

- Set **dbms.allow_upgrade=false** (or comment it).
- Set **dbms.mode=CORE**.

3.4.2.8 TAKE A BACKUP OF THE MIGRATED STORE

This step should be performed on the **elected instance**.

We are going to use **neo4j-admin dump** to create an offline backup of your newly migrated database and transactions, alongside the system database:

```
$NEO4J_HOME/bin/neo4j-admin dump --database=<DBNAME> --
to=$BACKUP_DESTINATION/<DBNAME>.dump

$NEO4J_HOME/bin/neo4j-admin dump --database=system --
to=$BACKUP_DESTINATION/system.dump
```

Please be aware that after you migrate, *Neo4j Admin* commands can differ slightly because we now need to account for multiple databases.

This backup will be used to seed the remaining instances of the cluster in the next step.

Do not yet start the server.

3.4.2.9 SEED THE CLUSTER

This step should be performed on the **remaining instances**.

You will now need to copy the dumps created on the step above to the **remaining instances**. Once this is complete, we can use **neo4j-admin load** to replace each of your databases, including the system database, with the ones migrated on the elected instance:

```
$NEO4J_HOME/bin/neo4j-admin load --from=$BACKUP_DESTINATION/<DBNAME>.dump --
database=<DBNAME> --force

$NEO4J_HOME/bin/neo4j-admin load --from=$BACKUP_DESTINATION/system.dump --
database=system --force
```

3.4.2.10 START THE CLUSTER

This step should be performed on **all instances**.

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**

If everything on the list above was successful, you can go ahead and start all instances of the cluster:

```
$NEO4J_HOME/bin/neo4j start

or

systemctl start neo4j
```

If you used a **different database name** than what is set as the default database in neo4j.conf, you will need to **activate that database** before being able to use it. You can do this with the CREATE DATABASE command in the browser/cypher-shell:

```
CREATE DATABASE <DBNAME>;
```

3.5 MIGRATION STEPS – DIRECT PATH

3.5.1 SINGLE INSTANCE

3.5.1.1 SHUTDOWN

Because a migration requires the database to be offline, the first step is to cleanly shutdown the database if it's still running:

```
$NEO4J_HOME/bin/neo4j stop

or

systemctl stop neo4j
```

It is important to verify that the database shutdown process finished successfully, and the database was cleanly shutdown. You can check the **neo4j.log** for these log messages for confirmation:

```
2021-04-30 13:47:27.220+0000 INFO  Neo4j Server shutdown initiated by request
2021-04-30 13:47:27.229+0000 INFO  Stopping...
2021-04-30 13:47:32.420+0000 INFO  Stopped.
```

3.5.1.2 BACKUP

We will need to backup the following components:

- **neo4j.conf**
- Contents of **$NEO4J_HOME/data/dbms** (if there are any native users)
- All the files used for encryption i.e. private key, public certificate, and the contents of the **trusted** and **revoked** directories (located in **$NEO4J_HOME/certificates/***)
- Neo4j store (optional)

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.

If for peace of mind you still want to take a backup, because the database is now offline we are going to use the **neo4j-admin dump** command to create an offline backup of your store:

```
$NEO4J_HOME/bin/neo4j-admin dump --to=$BACKUP_DESTINATION
```

This will create a file called **<db_name>.dump** (in whatever path you defined in **$BACKUP_DESTINATION**).

More information about the **3.5.x** dump tool can be found here.

3.5.1.3 INSTALL NEW VERSION

This step will allow you to install Neo4j 4.x binaries. Also – and as mentioned before – you will need to adjust any configuration settings that have changed from 3.5.x to 4.x.

At this point you should now have a backed up **neo4j.conf** from your old 3.5.x deployment, and also a list of **non-default/custom configurations** captured in the configuration changes section. With all of that, you need to choose one of the following methods, specific to your technology:

- If using a tarball or zip file for installation:
  - Untar or unzip Neo4j 4.x.
  - Apply configuration changes:
    - Review your backed up **neo4j.conf** and the changes to other configuration settings on our Migration Guide and update all applicable configuration settings.
    - Add any **non-default/custom configurations** previously present in your old **neo4j.conf** file.
  - Move the files used for encryption from the old installation to the new one.
- If using a Debian or RPM distribution:
  - Install Neo4j 4.x.
  - When prompted, review the differences between the **neo4j.conf** files of the previous version and Neo4j 4.x.
  - Add any **non-default/custom configurations** previously present in your old **neo4j.conf** file.

***TIP**: When migrating, it is particularly important to note any custom values of the settings **dbms.directories.data** and **dbms.default_database**. In cluster installations, pay attention to **cluster-specific configuration settings**, which may be different on different cluster members.*

3.5.1.4 MIGRATE DATA

When following the direct path, the store migration happens when we run the **neo4j-admin copy** command. You need to specify the **old store location** and the **name** for the target updated database:

```
$NEO4J_HOME/bin/neo4j-admin copy --from-path=/path/to/3.5.x/graph.db --to-database=<DBNAME>
```

You will get the progress of the migration on your screen. When it finishes, you will be presented with the summary of the process:

```
IMPORT DONE in 3s 83ms.
Imported:
  172 nodes
  253 relationships
  565 properties
Peak memory usage: 15.60MiB
2021-05-17 17:53:48.312+0000 INFO  [o.n.i.b.ImportLogic] Import completed
successfully, took 3s 83ms. Imported:
  172 nodes
  253 relationships
  565 properties
2021-05-17 17:53:48.385+0000 INFO  [StoreCopy] Import summary: Copying of 1026
records took 3 seconds (342 rec/s). Unused Records 601 (58%) Removed Records 0
(0%)
```

It is likely that you have created some indexes and constraints on your database. It is important to remember that when using the direct path, indexes are **not automatically migrated** so you will have to recreate them. After running the store migration, the *neo4j-admin copy* tool **extracts the schema** and generates a list of commands you can use to recreate your schema on the new 4.x store:

```
2021-05-17 17:53:48.386+0000 INFO  [StoreCopy] ### Extracting schema ###
2021-05-17 17:53:48.387+0000 INFO  [StoreCopy] Trying to extract schema...
2021-05-17 17:53:48.872+0000 INFO  [StoreCopy] ... found 1 schema definitions.
The following can be used to recreate the schema:
2021-05-17 17:53:48.877+0000 INFO  [StoreCopy]

CALL db.createIndex('index_a146d26', ['User'], ['username'], '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]})
2021-05-17 17:53:48.878+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 /neo/neo4j-enterprise-4.2.5/logs/neo4j-
admin-copy-2021-05-17.17.53.42.log as well for reference.
```

The recreate schema commands will also be **saved in the migration log file**, located in the **logs directory**.

3.5.1.5 START THE SERVER

After the migration is complete, we can start the server running Neo4j 4.x:

```
$NEO4J_HOME/bin/neo4j start

or

systemctl start neo4j
```

If you used a **different database name** than what is set as the default database in neo4j.conf, you will need to **activate that database** before being able to use it. You can do this with the CREATE DATABASE command in the browser/cypher-shell:

```
CREATE DATABASE <DBNAME>;
```

3.5.1.6 RECREATE INDEXES

The final step is to recreate any indexes or constraints that were identified by the *neo4j-admin copy* tool. In the example above, only one index was found and to recreate it, you simply need to **run the exact command** that was generated against the desired database (to select a database simply run **:USE <DBNAME>**):

```
CALL db.createIndex('index_a146d26', ['User'], ['username'], '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]})
```

3.5.2 CLUSTER

Migrating a cluster deployment requires some different and extra steps. The strategy for a cluster deployment is to complete an offline copy from one cluster instance, & 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 will 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.

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.

> *IMPORTANT: 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.*

3.5.2.1 SHUTDOWN

Because the migration will require the cluster to be offline, the first step is to cleanly shutdown **all instances** of your cluster:

```
$NEO4J_HOME/bin/neo4j stop

or

systemctl stop neo4j
```

It is important to verify that the database shutdown process finished successfully, and the database was cleanly shutdown. You can check the **neo4j.log** for these log messages for confirmation:

```
2021-04-30 13:47:27.220+0000 INFO  Neo4j Server shutdown initiated by request
2021-04-30 13:47:27.229+0000 INFO  Stopping...
2021-04-30 13:47:32.420+0000 INFO  Stopped.
```

3.5.2.2 BACKUP

We will need to backup the following components from **all instances**:

- **neo4j.conf**
- All the files used for encryption i.e. private key, public certificate, and the contents of the **trusted** and **revoked** directories (located in **$NEO4J_HOME/certificates/***)

Additionally, we will need to backup the following components from the **elected instance**:

- Contents of **$NEO4J_HOME/data/dbms** (if there are any native users)
- Neo4j store (optional)

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.

If for peace of mind you still want to take a backup, because the database is now offline we are going to use the **neo4j-admin dump** command to create an offline backup of your store:

```
$NEO4J_HOME/bin/neo4j-admin dump --to=$BACKUP_DESTINATION
```

This will create a file called **<db_name>.dump** (in whatever path you defined in **$BACKUP_DESTINATION**).

More information about the **3.5.x** dump tool can be found here.

3.5.2.3 INSTALL NEW VERSIONS

This step will allow you to install Neo4j 4.x binaries and should be completed on **all instances** of your cluster. Also – and as mentioned before – you will need to adjust any configuration settings that have changed from 3.5.x to 4.x.

At this point you should now have a backed up **neo4j.conf** from your old 3.5.x deployment, and also a list of **non-default/custom configurations** captured in the configuration changes section. With all of that, you need to choose one of the following methods, specific to your technology:

- If using a tarball or zip file for installation:
  - Untar or unzip Neo4j 4.x.
  - Apply configuration changes:
    - Review your backed up **neo4j.conf** and the changes to other configuration settings on our Migration Guide and update all applicable configuration settings.
    - Add any **non-default/custom configurations** previously present in your old **neo4j.conf** file.
  - Move the files used for encryption from the old installation to the new one.
- If using a Debian or RPM distribution:
  - Install Neo4j 4.x but do **NOT** start it.
  - When prompted, review the differences between the **neo4j.conf** files of the previous version and Neo4j 4.x.
  - Add any **non-default/custom configurations** previously present in your old **neo4j.conf** file.

***TIP**: When migrating, it is particularly important to note any custom values of the settings **dbms.directories.data** and **dbms.default_database**. In cluster installations, pay attention to **cluster-specific configuration settings**, which may be different on different cluster members.*

3.5.2.4 MIGRATE DATA

This step should be performed on the **elected instance** (using the 4.x neo4j-admin tool).

When following the direct path, the store migration happens when we run the **neo4j-admin copy** command. You need to specify the **old store location** and the **name** for the target updated database:

```
$NEO4J_HOME/bin/neo4j-admin copy --from-path=/path/to/3.5.x/graph.db --to-
database=<DBNAME>
```

You will get the progress of the migration on your screen. When it finishes, you will be presented with the summary of the process:

```
IMPORT DONE in 3s 83ms.
Imported:
  172 nodes
  253 relationships
  565 properties
Peak memory usage: 15.60MiB
2021-05-17 17:53:48.312+0000 INFO  [o.n.i.b.ImportLogic] Import completed
successfully, took 3s 83ms. Imported:
  172 nodes
  253 relationships
  565 properties
2021-05-17 17:53:48.385+0000 INFO  [StoreCopy] Import summary: Copying of 1026
records took 3 seconds (342 rec/s). Unused Records 601 (58%) Removed Records 0
(0%)
```

It is likely that you have created some indexes and constraints on your database. It is important to remember that when using the direct path, indexes are **not automatically migrated** so you will have to recreate them. After running the store migration, the *neo4j-admin copy* tool **extracts the schema** and generates a list of commands you can use to recreate your schema on the new 4.x store:

```
2021-05-17 17:53:48.386+0000 INFO  [StoreCopy] ### Extracting schema ###
2021-05-17 17:53:48.387+0000 INFO  [StoreCopy] Trying to extract schema...
2021-05-17 17:53:48.872+0000 INFO  [StoreCopy] ... found 1 schema definitions.
The following can be used to recreate the schema:
2021-05-17 17:53:48.877+0000 INFO  [StoreCopy]

CALL db.createIndex('index_a146d26', ['User'], ['username'], '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]})
2021-05-17 17:53:48.878+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 /neo/neo4j-enterprise-4.2.5/logs/neo4j-
admin-copy-2021-05-17.17.53.42.log as well for reference.
```

The recreate schema commands will also be **saved in the migration log file**, located in the **logs directory**.

3.5.2.5 SEED THE CLUSTER

If either of the following is true:

- you're installing a version of Neo4j prior to 4.3
- your migrated database is set as the default database in neo4j.conf

Then you should copy the migrated database directory **from** the **elected instance** to **all other instances**, in order 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**.

If seeding the cluster is required, you can use **neo4j-admin dump** to create an offline backup of your newly migrated database and transactions:

```
$NEO4J_HOME/bin/neo4j-admin dump --database=neo4j --
to=$BACKUP_DESTINATION/neo4j.dump
```

Please be aware that after you migrate, *Neo4j Admin* commands can differ slightly because we now need to account for multiple databases.

Do not yet start the server.

You will now need to copy the dump created above to the **remaining instances**. Once this is complete, we can use **neo4j-admin load** to replace each of your database with the one migrated on the elected instance:

```
$NEO4J_HOME/bin/neo4j-admin load --from=$BACKUP_DESTINATION/neo4j.dump --
database=neo4j --force
```

3.5.2.6 START THE CLUSTER

This step should be performed on **all instances**.

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*)

If everything on the list above was successful, you can go ahead and start all instances of the cluster:

```
$NEO4J_HOME/bin/neo4j start

or

systemctl start neo4j
```

3.5.2.7 START THE DATABASE

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.

Run the following command in Neo4j browser/cypher-shell :

- Neo4j 4.0/4.1/4.2 :

```
CREATE DATABASE <DBNAME>;
```

- Neo4j 4.3+ :

```
CREATE DATABASE <DBNAME> OPTIONS { existingData : 'use',
existingDataSeedInstance: '<seedInstanceId>'};
```

Where **<seedInstanceId>** is the id of the **elected instance**, which can be found by calling ***CALL dbms.cluster.overview()***.

3.5.2.8 RECREATE INDEXES

The final step is to recreate any indexes or constraints that were identified by the *neo4j-admin copy* tool. In the example above, only one index was found and to recreate it, you simply need to **run the exact command** that was generated against the desired database (to select a database simply run **:USE <DBNAME>**):

```
CALL db.createIndex('index_a146d26', ['User'], ['username'], '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]})
```

3.6 POST-MIGRATION STEPS

3.6.1 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, we 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:SHA-
256,1066956C2D4E46C810CA39AE218AAD128854F2C08E9E831C379958CBFA6FF17D,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 and looks like:

```
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:

```
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 *
```

```
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 will return a list of Cypher commands which, when executed against the system database, will **recreate all users and roles** previously used in the Neo4j 3.5.x deployment.

3.6.2 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.

3.6.3 RESTART THE SERVER

It is advisable to restart the server/cluster one last time just to clear everything and assume the last configuration changes.

3.6.4 REACTIVATE EXTERNAL APPLICATION(S) 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.

3.6.5 CLEAN UP SPACE

You can clean up the disk space taken by the extra backups required for the migration.

3.6.6 BACKUP

This is **completely optional**, but it is **good practice** to make a full backup immediately after the migration.

4. Tips

4.1 CONFIGURING SSL POLICY

Pay special attention to the configuration of the SSL policy. If you require encryption, then it's worth spending some time understanding what the changes in Neo4j 4.x are. Very briefly, on Neo4j 3.5.x we encrypted all connections by default and would issue a self-signed certificate. Modern browsers no longer support self-signed certificates, so we opted to default to unencrypted connections unless you provide a valid, not self-signed cert. This means you will need trusted certificates from a CA. You can read more about how to configure the SSL policy of our Migration Guide.

4.2 PERFORM A TEST MIGRATION

Neo4j's Customer Success and Customer Support teams cannot stress this enough: **Always perform (at least) one test migration**. You should take a backup of your production data and find a separate environment where you can perform mock migrations without affecting your production system. This will allow you to understand some important data points like:

- Downtime required
- Disk space
- Any specific infrastructure areas not covered by this guide or our documentation
- Any company specific details not covered by this guide or our documentation
- Any company specific processes

Working with assumptions and inaccurate information will mean that you will not be prepared for what could happen or go wrong, leaving you in a bad position if you are in a middle of a migration.

4.3 EXPLORE ROLE-BASED ACCESS CONTROL (RBAC)

Neo4j 4.0 completely revamped authentication and authorization. Authorization is now managed using **role-based access control** (**RBAC**). Permissions that define access control are assigned to roles, which are in turn assigned to users. This will allow you to have fine-grained access control to all graph elements.

This is an incredibly significant change but also an immensely powerful one. Take some time to review how RBAC works and how it applies to your deployment. You can check the authentication and authorization and the fine-grained access control sections of the Operations Manual which contain a lot of details about the changes as well as working examples.

4.4 TIPS ON HOW TO REDUCE DOWNTIME

Availability is important, we understand that. Whilst a database outage is always required, there are a few tips and strategies that can alleviate this.

Depending on your infrastructure and combining 3rd party software, we can even achieve read and (a form of) read write availability. However, all these strategies need to be seen as

workarounds and whilst they are theoretically possible, they are still **not officially supported**. Their mention on this document should be seen as ideas to help come up with strategies for use-cases where availability is a critical issue. Due to the vast number of moving parts, we recommend thoroughly testing to understand if they are viable for your use-case.

4.4.1 PLAN AHEAD

Plan ahead and complete tasks beforehand. Spend time putting things in place to save you time during the upgrade and reduce overall downtime. For example:

- You can prepare directories and install the binaries beforehand. Since we are going to be using new binaries, you can install and prepare everything without needing the database to be offline.
- You can prepare **neo4j.conf** beforehand. Based on test migrations, you can understand what you need to change in **neo4j.conf** and have a file prepared instead of making changes during the migration.
- Script operations. A migration is a delicate process that **requires human supervision**, but you can still script operations to save you time. For example, you can script taking a backup and sending it to the remaining instances via scp.

4.4.2 REUSE BACKUPS

If you've got an **up-to-date** backup and know for a fact that the data hasn't changed since the backup (ie: no new writes since last backup), you can reuse that backup instead of taking new ones just for the migration. We still advise to work with a copy of the backup (make a backup of the backup so to speak) so that you have a safeguard in case of failure.

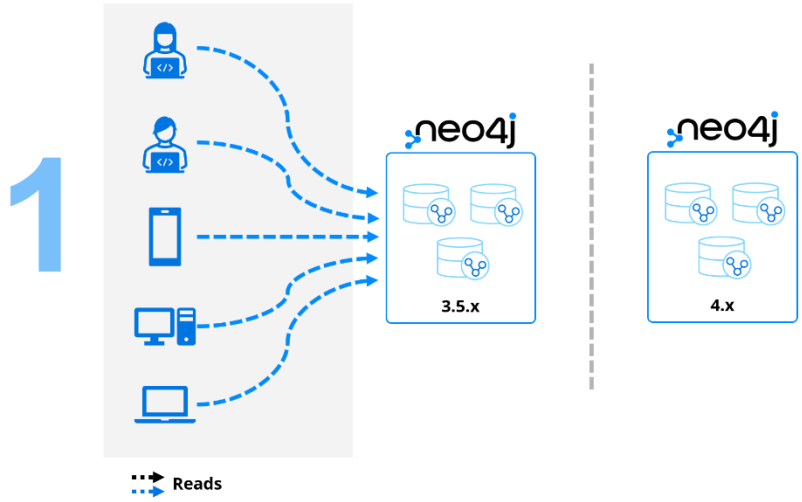
If you use this tip, you will likely have an **online backup** taken with **neo4j-admin backup** rather than an offline backup taken with neo4j-admin dump as described in this guide. Because of this, you will need to use **neo4j-admin restore** instead of neo4j-admin load to move the backup to the correct location (section 3.4.3 for single instance migrations, section 3.5.5 for cluster migrations). Use this command instead of the neo4j-admin load:

```
$NEO4J_HOME/bin/neo4j-admin restore --from=$BACKUP_DESTINATION --force
```

4.4.3 MINIMUM DOWNTIME

This approach allows for read availability, while you migrate the store, by building a **second cluster** in parallel. You will **have a period of downtime** towards the end of the process, but it should only last for the duration of a server shutdown and a startup.

The goal is to **interrupt write requests**, take a backup of your store, and use it to migrate the data. Finally, we will use that data to create an identical cluster but running version 4.x:



While your cluster is read-only mode, you can take an **online backup** using **neo4j-admin backup**:

```
$NEO4J_HOME/bin/neo4j-admin backup --backup-dir=$BACKUP_DESTINATION --
name=<backup_name>
```

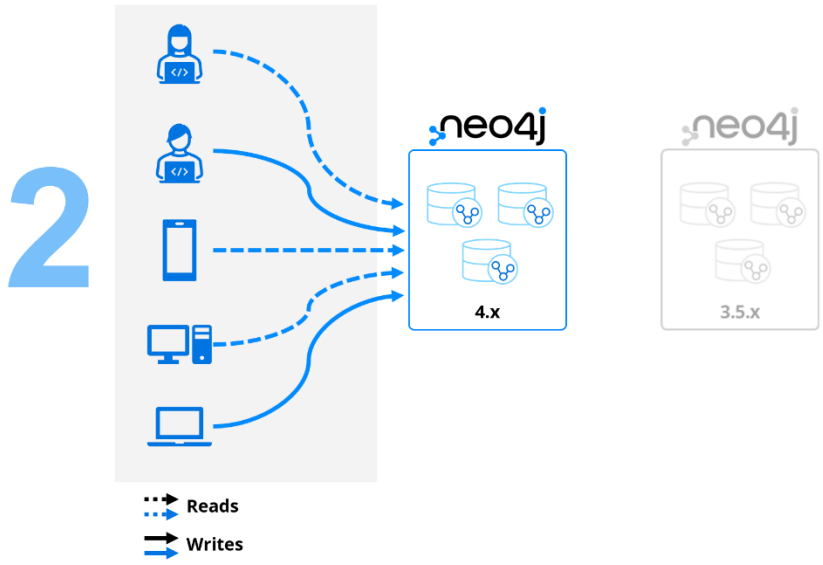
After taking the backup, you can proceed to the *Install new versions* section of this guide, either on the sequential or direct path and carry on from there.

If using the sequential path, because took an **online backup** using **neo4j-admin backup** rather than an offline backup using *neo4j-admin dump* as described in this guide, you will need to use **neo4j-admin restore** instead of *neo4j-admin load* to move the backup to the correct location:

```
$NEO4J_HOME/bin/neo4j-admin restore --from=$BACKUP_DESTINATION --force
```

When the migration is complete and your new cluster is running on version 4.x, you will have two clusters with the same data: an old one running version 3.5.x and a new one running version 4.x.

All that is left to do is shutdown the old cluster and start the new one. At this point, you will have the **whole cluster offline** for some time. Make sure that the new cluster is good to go and configured the same way (ie: same hostnames, same ports, same configuration) so that you can shut one down and start the other up as fast as possible.

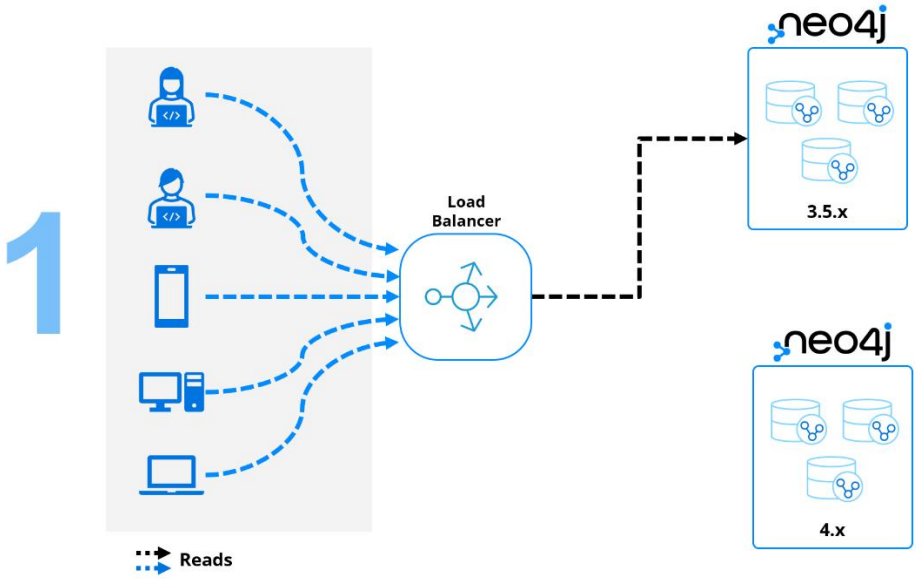


Finally, you can then decommission the old Neo4j 3.5.x cluster.

4.4.4 ACHIEVE ZERO READ DOWNTIME

You can maintain read availability to Neo4j while migrating with the help from a **load balancer** and a **second cluster**.

The idea is to create a new, freshly installed 4.x cluster alongside your existing 3.5.x cluster, both in front of a load balancer. You will then **interrupt write requests** and configure the load balancer to send all **read requests** to your existing Neo4j 3.5.x cluster:



While your cluster is read-only mode, you can take an **online backup** using **neo4j-admin backup**:

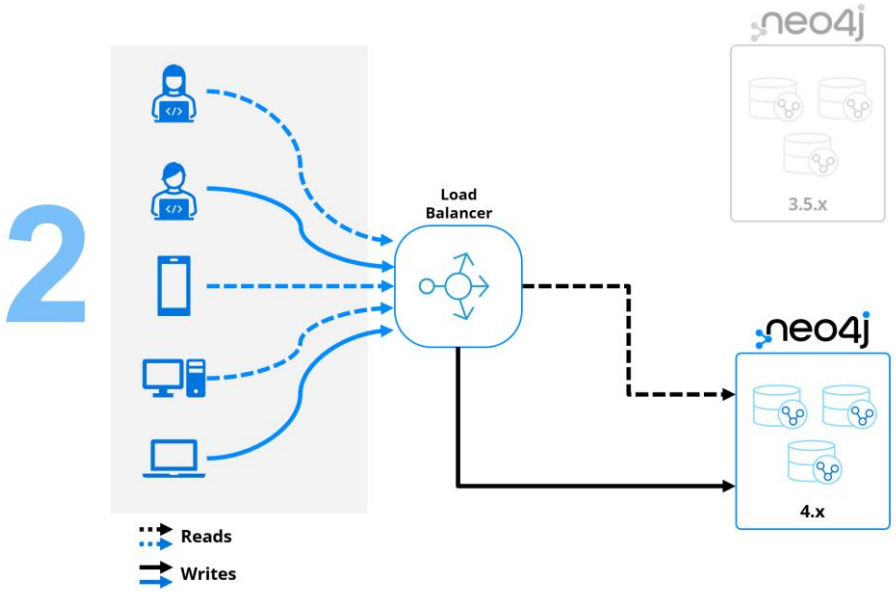
```
$NEO4J_HOME/bin/neo4j-admin backup --backup-dir=$BACKUP_DESTINATION --
name=<backup_name>
```

After taking the backup, you can proceed to the *Install new versions* section of this guide, either on the sequential or direct path and carry on from there.

If using the sequential path, because took an **online backup** using **neo4j-admin backup** rather than an offline backup using *neo4j-admin dump* as described in this guide, you will need to use **neo4j-admin restore** instead of *neo4j-admin load* to move the backup to the correct location:

```
$NEO4J_HOME/bin/neo4j-admin restore --from=$BACKUP_DESTINATION --force
```

When the migration is complete and your new cluster is running on version 4.x, you will have two clusters with the same data: an old one running version 3.5.x and a new one running version 4.x. All that is left to do is configure the load balancer to send all incoming traffic to your Neo4j 4.x cluster and re-enable writes:



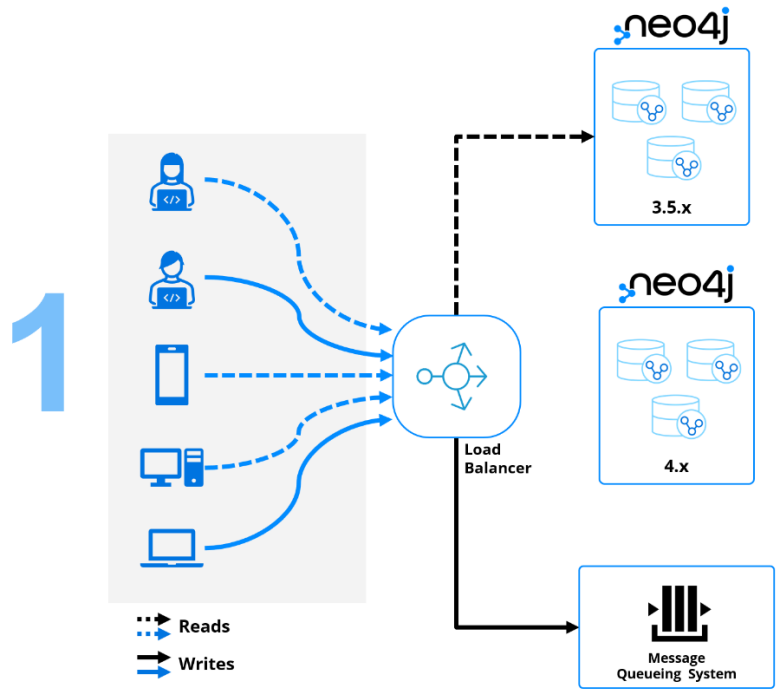
Finally, you can then decommission the old Neo4j 3.5.x cluster. If it's not serving any other purpose, you can also decommission the load balancer.

4.4.5 ACHIEVE ZERO READ DOWNTIME AND (SOME) WRITE AVAILABILITY

You can maintain read availability and a certain level of write availability while migrating with the help from a **load balancer**, a **second cluster** and a messaging queuing system like **Kafka**

The idea is to create a new, freshly installed 4.x cluster alongside your existing 3.5.x cluster, both in front of a load balancer and have a separate messaging queuing system to queue your writes

while the migration is in progress. The load balancer needs to be configured to send **all read requests** to the 3.5.x cluster and **all write requests** to the messaging queuing system:



While your writes requests are being queued, you can take an **online backup** from your existing 3.5.x instance using **neo4j-admin backup**:

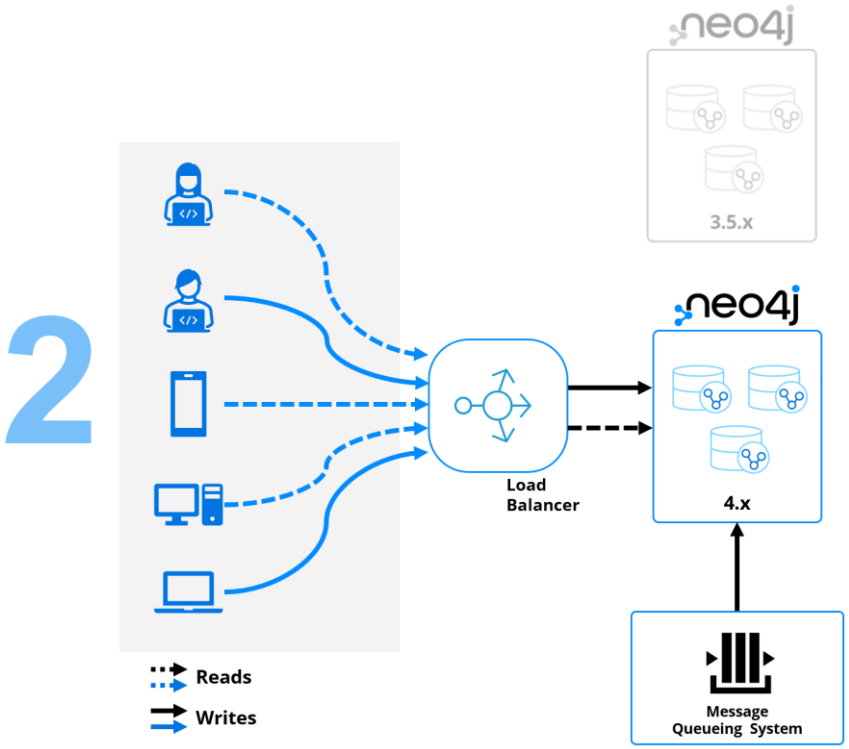
```
$NEO4J_HOME/bin/neo4j-admin backup --backup-dir=$BACKUP_DESTINATION --
name=<backup_name>
```

After taking the backup, you can proceed to the *Install new versions* section of this guide, either on the sequential or direct path and carry on from there.

If using the sequential path, because took an **online backup** using **neo4j-admin backup** rather than an offline backup using *neo4j-admin dump* as described in this guide, you will need to use **neo4j-admin restore** instead of *neo4j-admin load* to move the backup to the correct location:

```
$NEO4J_HOME/bin/neo4j-admin restore --from=$BACKUP_DESTINATION --force
```

When the migration is complete and your new cluster is running on version 4.x, you will have two clusters with the same data: an old one running version 3.5.x and a new one running version 4.x. All that is left to do is configure the load balancer to send **all incoming traffic** to your Neo4j 4.x cluster and instruct the message queueing system to **replay the messages queued** to the new Neo4j 4.x cluster:



Finally, you can then decommission the old Neo4j 3.5.x cluster. If they are not serving any other purposes, you can also decommission the load balancer and the messaging queueing system.

This strategy presents a **limitation** which is the fact that the writes queued during the migration will **not be available to read until the new cluster is up and finished processing the queue**. It is however the best we can do given the context and constraints present during a migration.

Whilst this solution won't work for all use cases, some might be able to tolerate this *delayed writes* strategy. If the use case permits, **the end-user experience should not be affected** since all read requests will be served by the old 3.5.x cluster, and all writes will be successful **albeit delayed**.

Lastly, if you are considering this strategy be mindful of any constraints that you store may have. Queueing write requests in the message queueing system means that **any constraints violations will not be evaluated until the request reaches the database** after the migration is complete. For this reason, some of the queued write requests **may fail**.

5. Professional Services

If after reading this document (and the related migration material) you were left with the feeling that all of this is a mammoth task, we can still help you! Our Professional Services team has packages ready to help you out with migrating your Neo4j deployments.

You can learn more by sending us an email with your enquiry to customer-success@neo4j.com.

5.1 NEO4J 4.0 RELEASE WORKSHOP (1-DAY)

The 1-day Neo4j 4.0 Release workshop gives you an insight into the various new features that are part of the 4.x releases allowing you to maximally benefit from this release and further leveraging your investment in our software.

The workshop will leave you with the following:

- A good technical understanding of the latest features
- A backlog of tasks needed/wanted to implement these features
- A high-level plan for upgrade of your environment to 4.0

5.2 BESPOKE DELIVERY SERVICES

If you require a more hands-on approach, we have bespoke delivery services including installation, configuration, and upgrade assistance. Subject to a short scoping call (or after the above workshop), we will gladly deliver you a quote for the upgrade of your system. Quotes typically range from 1-2 days of consulting (basic database upgrade) up to 10 days (database + wider application ecosystem).