

# Managing a Neo4j Database

# Table of Contents

About this module .....	1
Neo4j instance files .....	2
Post-installation preparation .....	3
Post-install: Changing the <i>neo4j</i> password (non-Debian) .....	3
Post-install: Debian .....	3
Managing the Neo4j instance .....	4
Checking the status of the instance .....	5
Viewing the neo4j log .....	6
<b>Exercise #1: Managing the Neo4j instance</b> .....	7
Using cypher-shell .....	8
Example: Using cypher-shell .....	8
Changing the default password (Debian only) .....	9
Accessing the database .....	10
<b>Exercise #2: Using cypher-shell to change the password</b> .....	11
Renaming a Neo4j database .....	13
Deleting a Neo4j database .....	14
Copying a Neo4j database .....	15
Creating and offline backup .....	16
Creating a database from an offline backup .....	17
Modifying config for new database .....	18
<b>Exercise #3: Copying a database</b> .....	19
Modifying the location of the database .....	22
Starting Neo4j instance with a new location .....	23
Using a different location for the database .....	24
<b>Exercise #4: Modifying the location of the database</b> .....	25
Checking the consistency of a database .....	29
Inconsistencies found .....	30
<b>Exercise #5: Checking consistency of a database</b> .....	31
Scripting with cypher-shell .....	33
Examples: Adding constraints .....	33
<b>Exercise #6: Scripting changes to the database</b> .....	34
Managing plugins .....	36
Retrieving available procedures .....	36
Adding a plugin to the Neo4j instance .....	37
Sandboxing and whitelisting .....	37
Example: Installing the Graph Algorithms plugin .....	38
Example: Download and ownership of plugin .....	39
Example: Sandboxing .....	40

Example: Restart with plugin .....	41
Example: Installing the APOC plugin .....	42
Example: Download and ownership of plugin .....	43
Example: Sandboxing .....	44
Example: Restart with plugin .....	45
<b>Exercise #7: Install a plugin .....</b>	46
Configuring connector ports for the Neo4j instance .....	50
Modifying the default connector ports .....	50
<b>Exercise #8: Modify the HTTP port .....</b>	51
Performing online backup and restore .....	53
Enabling online backup .....	53
Performing the backup .....	53
Restoring from a backup .....	54
<b>Exercise #9: Performing online backup and restore .....</b>	55
Using the import tool to create a database .....	58
Creating CSV files for the import .....	58
CSV files for nodes .....	58
CSV files for relationships .....	59
Importing the data .....	60
<b>Exercise #10: Importing data with the import command .....</b>	62
Check your understanding .....	64
Question 1 .....	64
Question 2 .....	64
Question 3 .....	64
Summary .....	65
Grade Quiz and Continue .....	66

# About this module

Now that you have installed the Neo4j Enterprise Edition, you will learn how to perform some administrative tasks with the Neo4j instance.

At the end of this module, you should be able to:

- Start a Neo4j instance.
- Stop the Neo4j instance.
- Set the password for the *neo4j* user.
- Copy a Neo4j database.
- Modify the location for a Neo4j database.
- Check the consistency of a Neo4j database.
- Create scripts for modifying a Neo4j database.
- Manage plugins for a Neo4j database.
- Configure ports used by the Neo4j instance.
- Perform an online backup of a Neo4j database.
- Create a database with the import tool.

# Neo4j instance files

Depending on your platform, a Neo4j instance's files are, by default, placed as described [here](#). Here is a brief overview of the default folders you will frequently use for managing the Neo4j instance.

Purpose of folder	Description
Tools	The <b>/usr/bin</b> folder contains the tooling scripts you will typically run to manage the Neo4j instance.
Configuration	<b>Neo4j.conf</b> is the primary configuration file for the Neo4j instance and resides in the <b>/etc/neo4j</b> folder.
Logging	The <b>/var/log/neo4j</b> folder contains log files that you can monitor.
Database(s)	The <b>/var/lib/neo4j/data</b> folder contains the database(s).

# Post-installation preparation

In this training, all screenshots and examples are shown using Neo4j Enterprise Edition installed as a Debian package. If your system is different, you will need to adjust file locations as described later in this module.

When you are setting up a production environment, you want to control who can manage the Neo4j instance. You will also want to control when the Neo4j instance starts as you will performing some configuration changes and database operations that may require that the instance to be stopped.

When Neo4j is installed as a Debian package, the *neo4j* service is enabled and the Neo4j instance is automatically started. Other platforms do not start the Neo4j instance automatically.

## Post-install: Changing the *neo4j* password (non-Debian)

After you install Neo4j and before you start the Neo4j instance, a best practice is to change the default password for the user *neo4j*. You do this on all platforms, except when you have installed a Debian package. You will learn about changing the *neo4j* password on Debian later in this module.

You change the password for the *neo4j* user by executing the following command:

```
[sudo] bin/neo4j-admin set-initial-password newPassword
```

where *newPassword* is a password you will remember.

## Post-install: Debian

Initially and on Debian, you should disable *neo4j* as a service that is started automatically when the system starts. You do this with this command:

```
[sudo] systemctl disable neo4j
```

In addition, you should create the folder **/var/run/neo4j** that is owned by *neo4j:neo4j*. This is where the PID for the currently running Neo4j instance is placed.

# Managing the Neo4j instance

When the instance is started, it creates a database named **graph.db** in the default location which is a folder under **/var/lib/neo4j/data/databases**. You can start and stop the instance regardless of whether the *neo4j* service is enabled.

You start, stop, restart, and check the status of the Neo4j instance on Debian as follows:

- [sudo] `systemctl start neo4j`
- [sudo] `systemctl stop neo4j`
- [sudo] `systemctl restart neo4j`
- [sudo] `systemctl status neo4j`

You start, stop, restart and check the status of the Neo4j instance on non-Debian systems as follows:

- [sudo] `bin/neo4j start`
- [sudo] `bin/neo4j stop`
- [sudo] `bin/neo4j restart`
- [sudo] `bin/neo4j status`

When the Neo4j instance starts, it opens the database, and writes to the folders for the database and to the log file.

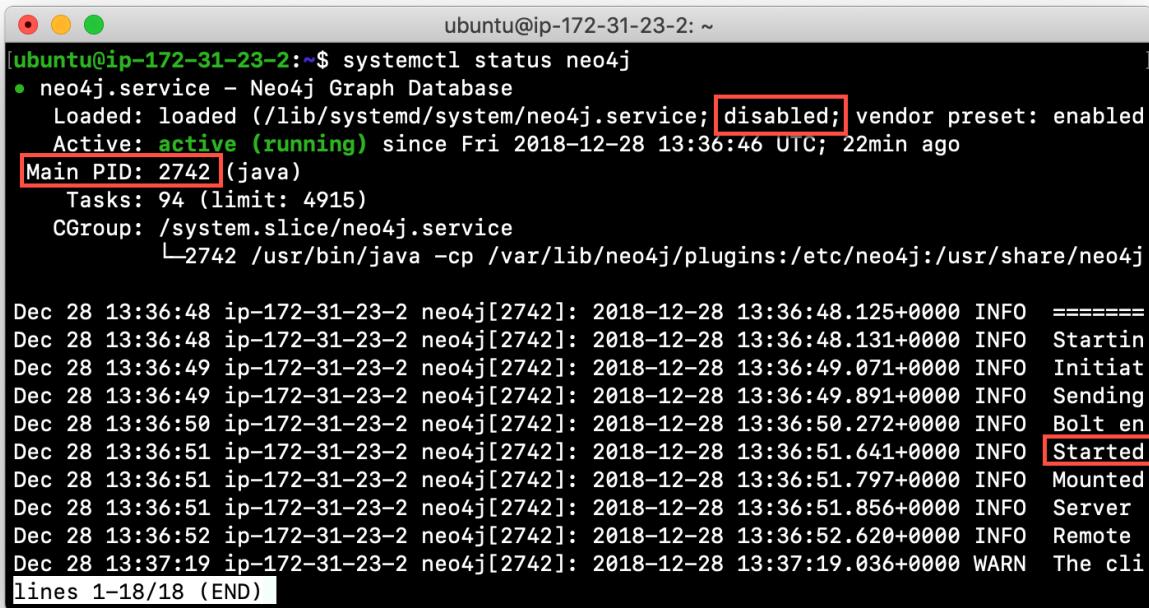
# Checking the status of the instance

At any time, you can check the status of the Neo4j instance.

You check the status of the instance as follows:

```
systemctl status neo4j
```

Here is an example where we check the status of the Neo4j instance:



```
ubuntu@ip-172-31-23-2: ~
[ubuntu@ip-172-31-23-2:~$ systemctl status neo4j
● neo4j.service - Neo4j Graph Database
  Loaded: loaded (/lib/systemd/system/neo4j.service; disabled; vendor preset: enabled
  Active: active (running) since Fri 2018-12-28 13:36:46 UTC; 22min ago
    Main PID: 2742 (java)
      Tasks: 94 (limit: 4915)
     CGroup: /system.slice/neo4j.service
             └─2742 /usr/bin/java -cp /var/lib/neo4j/plugins:/etc/neo4j:/usr/share/neo4j

Dec 28 13:36:48 ip-172-31-23-2 neo4j[2742]: 2018-12-28 13:36:48.125+0000 INFO =====
Dec 28 13:36:48 ip-172-31-23-2 neo4j[2742]: 2018-12-28 13:36:48.131+0000 INFO Startin
Dec 28 13:36:49 ip-172-31-23-2 neo4j[2742]: 2018-12-28 13:36:49.071+0000 INFO Initiat
Dec 28 13:36:49 ip-172-31-23-2 neo4j[2742]: 2018-12-28 13:36:49.891+0000 INFO Sending
Dec 28 13:36:50 ip-172-31-23-2 neo4j[2742]: 2018-12-28 13:36:50.272+0000 INFO Bolt en
Dec 28 13:36:51 ip-172-31-23-2 neo4j[2742]: 2018-12-28 13:36:51.641+0000 INFO Started
Dec 28 13:36:51 ip-172-31-23-2 neo4j[2742]: 2018-12-28 13:36:51.797+0000 INFO Mounted
Dec 28 13:36:51 ip-172-31-23-2 neo4j[2742]: 2018-12-28 13:36:51.856+0000 INFO Server
Dec 28 13:36:52 ip-172-31-23-2 neo4j[2742]: 2018-12-28 13:36:52.620+0000 INFO Remote
Dec 28 13:37:19 ip-172-31-23-2 neo4j[2742]: 2018-12-28 13:37:19.036+0000 WARN The cli
lines 1-18/18 (END)
```

Here we see that the instance is started. Notice that the service is disabled as well. After the instance is started you can identify the process ID (Main PID) from the status command on Debian. It is sometimes helpful to know the process ID of the Neo4j instance (JVM) in the event that it is unresponsive and you must kill it.

However, knowing whether the instance is started (active) is generally not sufficient, especially if you have made some configuration changes. You can view details of the Neo4j instance by examining the log file.

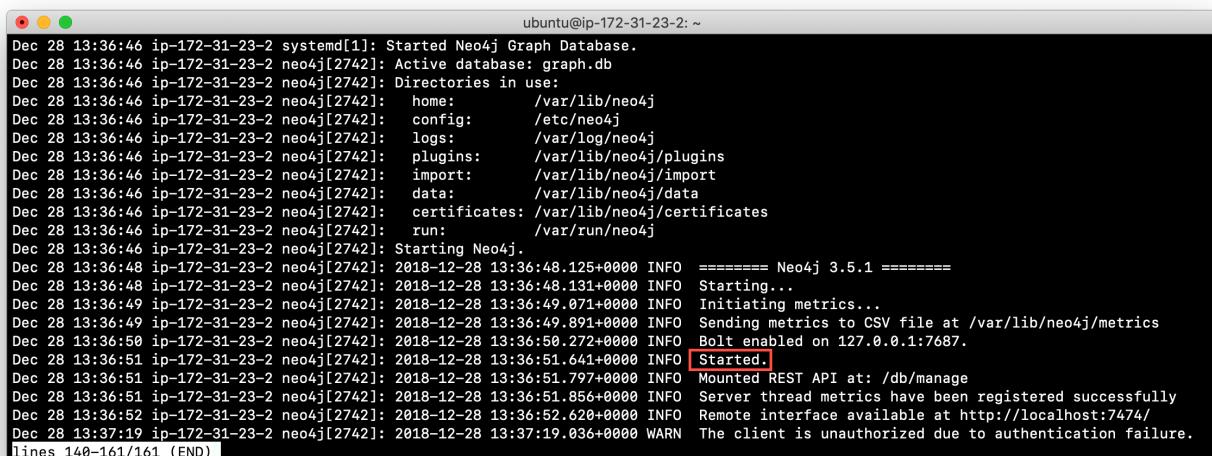
# Viewing the neo4j log

The status command gives you a short glimpse of the status of the Neo4j instance. In some cases, although the instance is *active*, it may not have started successfully. You may want to examine more information about the instance, such as the folders it is using at runtime and information about activity against the instance, and especially if any errors occurred during startup. As an administrator, you should become familiar with the types of records that are written to the log files for the Neo4j instance.

You can view the log file for the instance on Debian as follows:

- `journalctl -u neo4j` to view the entire neo4j log file.
- `journalctl -e -u neo4j` to view the end of the neo4j log file.
- `journalctl -u neo4j -b > neo4j.log` where you can view **neo4j.log** in an editor.

Here is the result from `journalctl`:



```
ubuntu@ip-172-31-23-2: ~
Dec 28 13:36:46 ip-172-31-23-2 systemd[1]: Started Neo4j Graph Database.
Dec 28 13:36:46 ip-172-31-23-2 neo4j[2742]: Active database: graph.db
Dec 28 13:36:46 ip-172-31-23-2 neo4j[2742]: Directories in use:
Dec 28 13:36:46 ip-172-31-23-2 neo4j[2742]:   home:          /var/lib/neo4j
Dec 28 13:36:46 ip-172-31-23-2 neo4j[2742]:   config:        /etc/neo4j
Dec 28 13:36:46 ip-172-31-23-2 neo4j[2742]:   logs:          /var/log/neo4j
Dec 28 13:36:46 ip-172-31-23-2 neo4j[2742]:   plugins:       /var/lib/neo4j/plugins
Dec 28 13:36:46 ip-172-31-23-2 neo4j[2742]:   import:        /var/lib/neo4j/import
Dec 28 13:36:46 ip-172-31-23-2 neo4j[2742]:   data:          /var/lib/neo4j/data
Dec 28 13:36:46 ip-172-31-23-2 neo4j[2742]:   certificates: /var/lib/neo4j/certificates
Dec 28 13:36:46 ip-172-31-23-2 neo4j[2742]:   run:           /var/run/neo4j
Dec 28 13:36:46 ip-172-31-23-2 neo4j[2742]: Starting Neo4j.
Dec 28 13:36:48 ip-172-31-23-2 neo4j[2742]: 2018-12-28 13:36:48.125+0000 INFO ===== Neo4j 3.5.1 =====
Dec 28 13:36:48 ip-172-31-23-2 neo4j[2742]: 2018-12-28 13:36:48.131+0000 INFO Starting...
Dec 28 13:36:49 ip-172-31-23-2 neo4j[2742]: 2018-12-28 13:36:49.071+0000 INFO Initiating metrics...
Dec 28 13:36:49 ip-172-31-23-2 neo4j[2742]: 2018-12-28 13:36:49.891+0000 INFO Sending metrics to CSV file at /var/lib/neo4j/metrics
Dec 28 13:36:50 ip-172-31-23-2 neo4j[2742]: 2018-12-28 13:36:50.272+0000 INFO Bolt enabled on 127.0.0.1:7687.
Dec 28 13:36:51 ip-172-31-23-2 neo4j[2742]: 2018-12-28 13:36:51.641+0000 INFO Started.
Dec 28 13:36:51 ip-172-31-23-2 neo4j[2742]: 2018-12-28 13:36:51.797+0000 INFO Mounted REST API at: /db/manage
Dec 28 13:36:51 ip-172-31-23-2 neo4j[2742]: 2018-12-28 13:36:51.856+0000 INFO Server thread metrics have been registered successfully
Dec 28 13:36:52 ip-172-31-23-2 neo4j[2742]: 2018-12-28 13:36:52.620+0000 INFO Remote interface available at http://localhost:7474/
Dec 28 13:37:19 ip-172-31-23-2 neo4j[2742]: 2018-12-28 13:37:19.036+0000 WARN The client is unauthorized due to authentication failure.
lines 140-161/161 (END)
```

When the Neo4j instance starts, you can also confirm that it is started by seeing the *Started* record in the log file.

**NOTE** You can also view the log file in the **logs** folder on all platforms.

# Exercise #1: Managing the Neo4j instance

In this Exercise, you will stop and start the Neo4j instance and view its status and log file.

## Before you begin

You should disable the *neo4j* service `[sudo] systemctl disable neo4j`, if you are using a system that utilizes the *neo4j* service (for example, Debian).

## Exercise steps:

1. Open a terminal on your system.
2. View the status of the Neo4j instance.
3. Stop the Neo4j instance.
4. View the status of the Neo4j instance.
5. Examine the Neo4j log file.
6. Examine the files and folders created for this Neo4j instance.

# Using cypher-shell

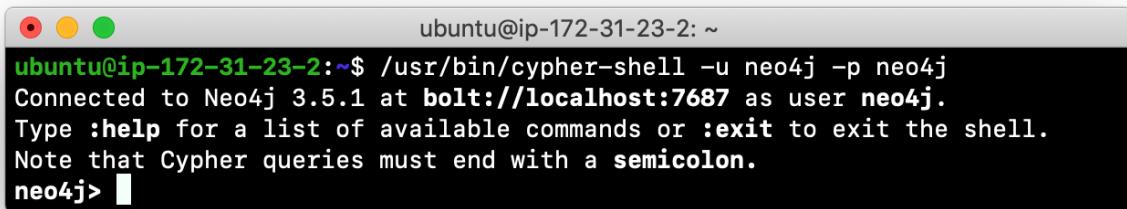
`cypher-shell` enables you to access the Neo4j database from a terminal window. You simply log into the database using `cypher-shell` with your credentials:

```
/usr/bin/cypher-shell -u <username> -p <password>
```

Once authenticated, you enter Cypher statements to execute just as you would in a Neo4j Browser session. One caveat with `cypher-shell`, however is that all Cypher commands must end with `;`. You exit `cypher-shell` with the command `:exit`.

## Example: Using cypher-shell

Here is an example showing that we can successfully log in to the database for the Neo4j instance, providing the default credentials `neo4j/neo4j`:



A screenshot of a terminal window on an Ubuntu system. The title bar says "ubuntu@ip-172-31-23-2: ~". The command entered is "/usr/bin/cypher-shell -u neo4j -p neo4j". The response shows the connection to Neo4j 3.5.1 at bolt://localhost:7687 as user neo4j. It also provides instructions for help and exiting the shell, and a note about Cypher queries ending with a semicolon. The prompt "neo4j>" is visible at the bottom.

```
ubuntu@ip-172-31-23-2:~$ /usr/bin/cypher-shell -u neo4j -p neo4j
Connected to Neo4j 3.5.1 at bolt://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.
neo4j> █
```

**NOTE** If you set the environment variables `NEO4J_USER` and `NEO4J_PASSWORD` with their respective values, then you need not enter your credentials when logging into `cypher-shell`.

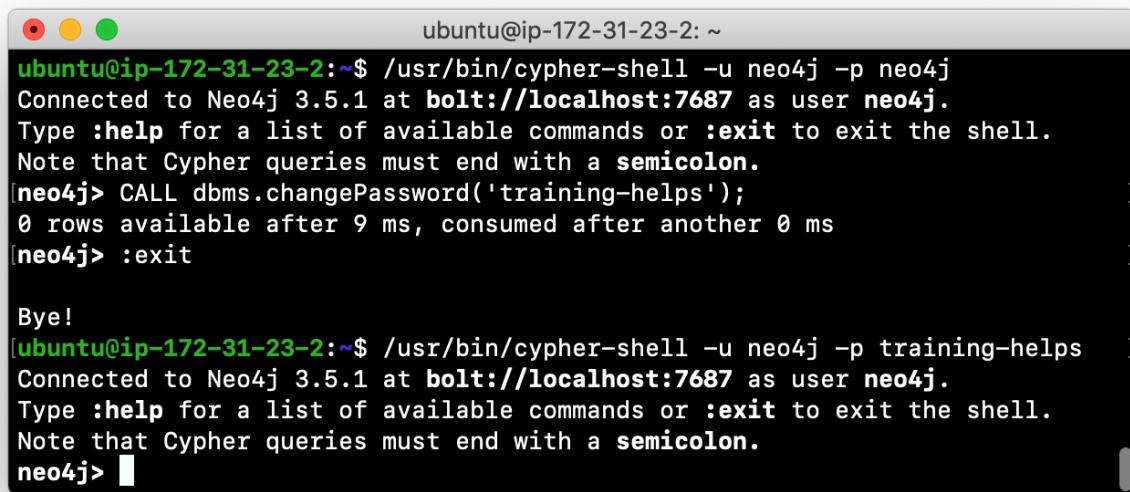
# Changing the default password (Debian only)

If we were to attempt to access the database for the first time, we would receive an error. This is because the default credentials `neo4j/neo4j` must be changed. As an administrator, you want to control who can manage this Neo4j instance and its database. To do so, you change the default password for the `neo4j` user. Later in this training, you will learn more about securing Neo4j by managing users and their access.

While logged into the database in `cypher-shell`, you execute the procedure to change the password:

```
CALL dbms.changePassword('newPassword');
```

In this example, we log into `cypher-shell` with our credentials. Then we execute the Cypher command to change the password. Finally, we specify `:exit` to log out of `cypher-shell`.



The screenshot shows a terminal window with a black background and white text. At the top, it says "ubuntu@ip-172-31-23-2: ~". The terminal output is as follows:

```
ubuntu@ip-172-31-23-2:~$ /usr/bin/cypher-shell -u neo4j -p neo4j
Connected to Neo4j 3.5.1 at bolt://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.

[neo4j> CALL dbms.changePassword('training-helps');
0 rows available after 9 ms, consumed after another 0 ms

[neo4j> :exit

Bye!
[ubuntu@ip-172-31-23-2:~$ /usr/bin/cypher-shell -u neo4j -p training-helps
Connected to Neo4j 3.5.1 at bolt://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.

neo4j> ]
```

After changing the default password for the Neo4j instance (database), we are now able to access the database after logging in with the new credentials.

# Accessing the database

Here is an example where we execute a Cypher statement against the empty database where we list all active queries:

```
ubuntu@ip-172-31-23-2:~$ /usr/bin/cypher-shell -u neo4j -p training-helps
Connected to Neo4j 3.5.1 at bolt://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.
neo4j> CALL dbms.listQueries();
+-----+
| queryId | username | metaData | query                                | parameters | planner      | runtime     | indexes | sta
rtTime   | protocol | clientAddress | requestUri    | status      | resourceInformation | activeLock
Count | elapsedTimeMillis | cpuTimeMillis | waitTimeMillis | idleTimeMillis | allocatedBytes | pageHits | pageFaults | connectionId |
+-----+
+-----+
| "query-7" | "neo4j"  | "bolt"      | "CALL dbms.listQueries();" | "procedure" | "procedure" | []        | 0       | "20
18-12-28T17:27:06.64Z" | "20"      | "127.0.0.1:39200" | "127.0.0.1:7687" | "running"  | NULL       | 0         | 0       | 0           |
"bolt-2"  |           | NULL       | 0          | NULL       | NULL       | 0         | 0       | 0           |
+-----+
1 row available after 31 ms, consumed after another 1 ms
neo4j> :exit
Bye!
ubuntu@ip-172-31-23-2:~$ ]
```

When you are done with `cypher-shell`, you enter `:exit` to exit.

## Exercise #2: Using cypher-shell to change the password

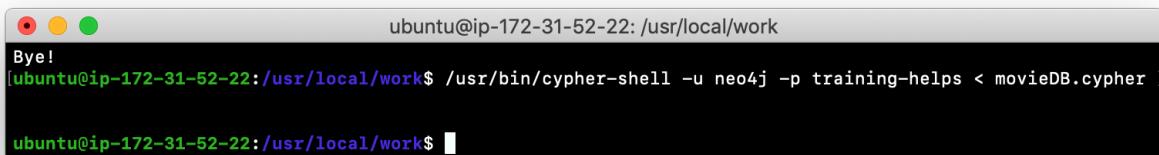
In this Exercise, you will log in to the database with `cypher-shell`, change the password for the database, and execute a Cypher statement to load the database. You can perform this Exercise regardless of the type of system you are using.

### Before you begin

You should ensure that the Neo4j instance is started.

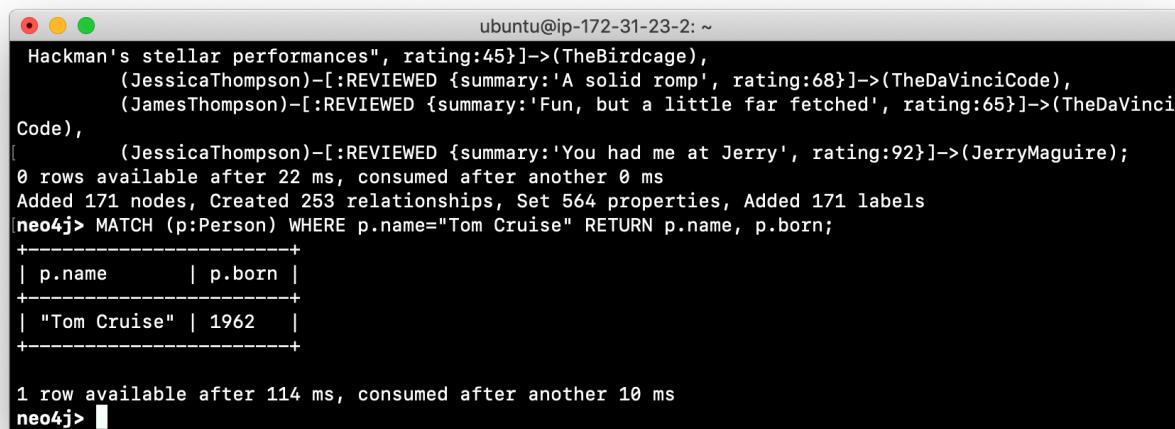
### Exercise steps:

1. Open a terminal on your system.
2. Log into the database with `cypher-shell` using the default credentials of `neo4j/neo4j`. (or different credentials if you changed the password previously with `neo4j-admin set-initial-password`)
3. Execute the Cypher statement, `CALL dbms.listQueries();`. Do you get an error? Note you will not get an error if you previously changed the password.
4. Execute the Cypher statement to change the password to something you will remember.
5. Exit out of `cypher-shell`.
6. Log into the database with `cypher-shell` using the new credentials.
7. Execute the Cypher statement, `CALL dbms.listQueries();`.
8. Exit out of `cypher-shell`.
9. Download this [file](#). This file contains the Cypher statements to load the database with movie data.
10. Invoke `cypher-shell` sending `movieDB.cypher` as input. You should see something like the following:



```
ubuntu@ip-172-31-52-22: /usr/local/work
Bye!
[ubuntu@ip-172-31-52-22:/usr/local/work$ /usr/bin/cypher-shell -u neo4j -p training-helps < movieDB.cypher ]
ubuntu@ip-172-31-52-22:/usr/local/work$
```

11. The database is now populated with the *Movie* data. Log in to `cypher-shell` and execute a Cypher statement to retrieve data from the database, for example: `MATCH (p:Person) WHERE p.name='Tom Cruise' RETURN p.name, p.born;` You should see the following:



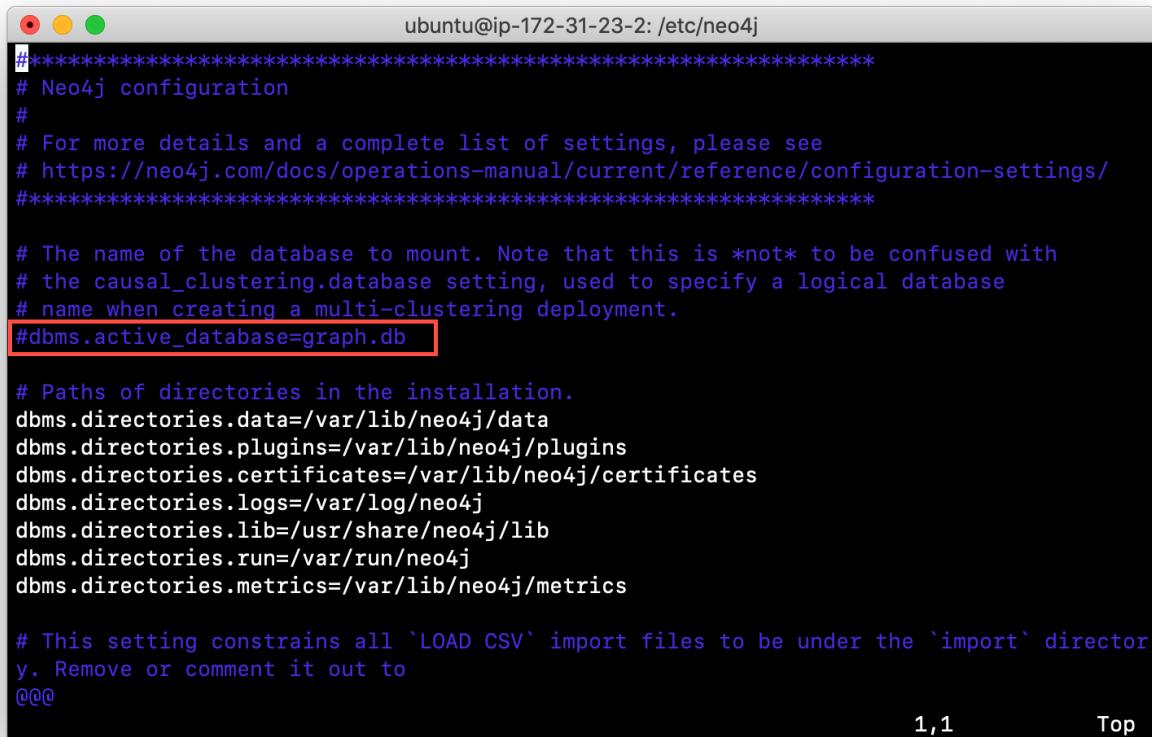
```
ubuntu@ip-172-31-23-2: ~
Hackman's stellar performances", rating:45})->(TheBirdcage),
(JessicaThompson)-[:REVIEWED {summary:'A solid romp', rating:68}]->(TheDaVinciCode),
(JamesThompson)-[:REVIEWED {summary:'Fun, but a little far fetched', rating:65}]->(TheDaVinci
Code),
(JessicaThompson)-[:REVIEWED {summary:'You had me at Jerry', rating:92}]->(JerryMaguire);
0 rows available after 22 ms, consumed after another 0 ms
Added 171 nodes, Created 253 relationships, Set 564 properties, Added 171 labels
[neo4j]> MATCH (p:Person) WHERE p.name="Tom Cruise" RETURN p.name, p.born;
+-----+
| p.name      | p.born   |
+-----+
| "Tom Cruise" | 1962    |
+-----+
1 row available after 114 ms, consumed after another 10 ms
neo4j> █
```

12. Exit `cypher-shell`.

# Renaming a Neo4j database

By default, the Neo4j database (on Debian) is located in the `/var/lib/neo4j/data/databases` folder. The database is represented by a subfolder with the default name, `graph.db`. You should never modify, copy, or move any files or folders at or under `graph.db`.

A key file for a Neo4j instance is `/etc/neo4j/neo4j.conf`. This file contains all settings used by the Neo4j instance at runtime. Here is a portion of the default `neo4j.conf` file that is installed with Neo4j. The setting for the name of the database is the property `dbms.active_database`, which, by default, is `graph.db`. Since this is the default configuration as installed, this setting is commented out in the configuration file because Neo4j uses the default at runtime.



```
ubuntu@ip-172-31-23-2: /etc/neo4j
*****
# Neo4j configuration
#
# For more details and a complete list of settings, please see
# https://neo4j.com/docs/operations-manual/current/reference/configuration-settings/
*****

# The name of the database to mount. Note that this is *not* to be confused with
# the causal_clustering.database setting, used to specify a logical database
# name when creating a multi-clustering deployment.
#dbms.active_database=graph.db

# Paths of directories in the installation.
dbms.directories.data=/var/lib/neo4j/data
dbms.directories.plugins=/var/lib/neo4j/plugins
dbms.directories.certificates=/var/lib/neo4j/certificates
dbms.directories.logs=/var/log/neo4j
dbms.directories.lib=/usr/share/neo4j/lib
dbms.directories.run=/var/run/neo4j
dbms.directories.metrics=/var/lib/neo4j/metrics

# This setting constrains all `LOAD CSV` import files to be under the `import` director
y. Remove or comment it out to
@@@
```

1,1      [Top](#)

If you wanted to change the name of the Neo4j database, you could change the folder name `graph.db` to another name, but if you do so, you must uncomment the line in `neo4j.conf` for `dbms.active_database` to match what you have renamed the database folder to. You should make this type of change in the configuration when the Neo4j instance is stopped.

# Deleting a Neo4j database

You would want to delete a Neo4j database for a couple of reasons:

- The database is no longer needed or usable and you want to recreate a fresh database.
- The database is no longer needed and you want to remove it so that a new database can be used. To do this you would load a new database which you will learn about next in this module.

To delete a Neo4j database used by a Neo4j instance you must:

1. Stop the Neo4j instance.
2. Remove the folder for the active database.

For example, delete the **graph.db** database:

```
[sudo] rm -rf /var/lib/neo4j/data/databases/graph.db
```

After deleting the Neo4j database, if you were to start the Neo4j instance, it would recreate an empty database. If you want to copy an existing database for use with this Neo4j instance, you dump and load an existing database to be used as the active database. Then you can start the Neo4j instance. You will learn about dumping and loading a database next.

# Copying a Neo4j database

The structure of a Neo4j database is proprietary and could change from one release to another. You should never copy the database from one location in the filesystem/network to another location. You copy a Neo4j database by creating an offline backup.

To create an offline backup of a database that, perhaps you want to have as an additional copy or you want to give to another user for use on their system, you must:

1. Stop the Neo4j instance.
2. Ensure that the folder where you will dump the database exists.
3. Use the `dump` command of the `neo4j-admin` tool to create the dump file.
4. You can now copy the dump file between systems.

Then, if you want to create a database from any offline backup file to use for a Neo4j instance, you must:

1. Stop the Neo4j instance.
2. Determine what you will call the new database and adjust `neo4j.conf` to use this database as the active database.
3. Use the `load` command of the `neo4j-admin` tool to create the database from the dump file using the same name you specify in the `neo4j.conf` file.
4. Start the Neo4j instance.

**NOTE**

Dumping and loading a database is done when the Neo4j instance is stopped. Later in this module, you will learn about online backup and restore. Offline backup is typically done for initial setup and development purposes. Online backup and restore is done in a production environment.

# Creating and offline backup

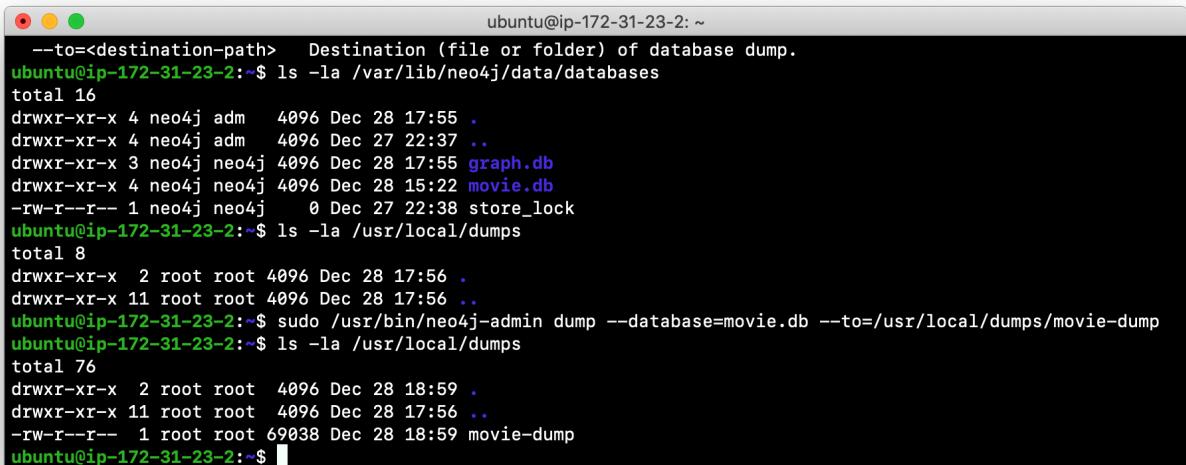
To create an offline backup, the Neo4j instance must be stopped. Here is how to use the `dump` command of the `neo4j-admin` tool to dump a database to a file:

```
[sudo] neo4j-admin dump --database=db-folder --to=db-target-folder/db-dump-file
```

where:

<code>db-folder</code>	is the name of the folder representing source database to be dumped.
<code>db-target-folder</code>	is the folder in the filesystem where you want to place the dumped database. This folder must exist.
<code>db-dump-file</code>	is the name of the dump file that will be created.

Here is an example where we have previously renamed the database to be `movie.db` and we have created a folder named `dumps`. We dump the `movie.db` using `neo4j-admin`:



The screenshot shows a terminal window on an Ubuntu system. The user has run the command `ls -la /var/lib/neo4j/data/databases`, which lists several database files including `graph.db` and `movie.db`. The user then runs `sudo /usr/bin/neo4j-admin dump --database=movie.db --to=/usr/local/dumps/movie-dump`. Finally, the user runs `ls -la /usr/local/dumps`, which shows a new file named `movie-dump`.

```
ubuntu@ip-172-31-23-2: ~
--to=<destination-path> Destination (file or folder) of database dump.
ubuntu@ip-172-31-23-2:~$ ls -la /var/lib/neo4j/data/databases
total 16
drwxr-xr-x 4 neo4j adm 4096 Dec 28 17:55 .
drwxr-xr-x 4 neo4j adm 4096 Dec 27 22:37 ..
drwxr-xr-x 3 neo4j neo4j 4096 Dec 28 17:55 graph.db
drwxr-xr-x 4 neo4j neo4j 4096 Dec 28 15:22 movie.db
-rw-r--r-- 1 neo4j neo4j 0 Dec 27 22:38 store.lock
ubuntu@ip-172-31-23-2:~$ ls -la /usr/local/dumps
total 8
drwxr-xr-x 2 root root 4096 Dec 28 17:56 .
drwxr-xr-x 11 root root 4096 Dec 28 17:56 ..
ubuntu@ip-172-31-23-2:~$ sudo /usr/bin/neo4j-admin dump --database=movie.db --to=/usr/local/dumps/movie-dump
ubuntu@ip-172-31-23-2:~$ ls -la /usr/local/dumps
total 76
drwxr-xr-x 2 root root 4096 Dec 28 18:59 .
drwxr-xr-x 11 root root 4096 Dec 28 17:56 ..
-rw-r--r-- 1 root root 69038 Dec 28 18:59 movie-dump
ubuntu@ip-172-31-23-2:~$
```

After the dump file, `movie-dump` is created, you can move it anywhere on the filesystem or network.

# Creating a database from an offline backup

Assuming that you have a dump file to use, you must first determine what the name of the target database will be. If you use an existing database name, the `load` command, can overwrite the database. If you want to create a new database, then you specify a database name that does not already exist. To perform the `load` command, the Neo4j instance must be stopped. In addition, the user:group permissions of the files created must be `neo4j:neo4j`.

**NOTE**

You must either perform the `load` operation as the `neo4j` user, or after the load, you must change the owner of all files and folders created to `neo4j:neo4j`.

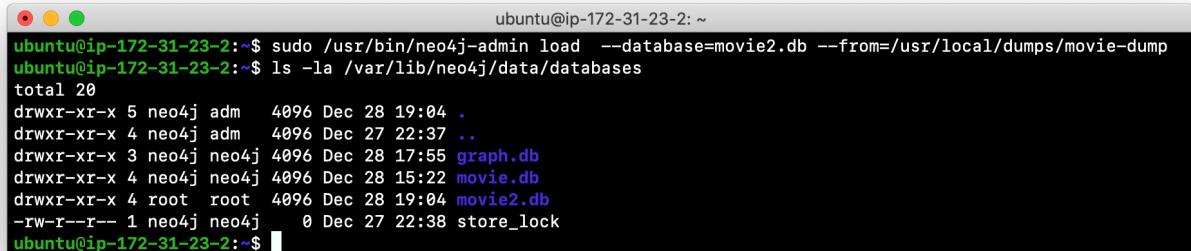
Here is how to use the `load` command of the `neo4j-admin` tool to load a database from a file:

```
[sudo] neo4j-admin load --from=path/db-dump-file --database=db-folder [--force=true]
```

where:

<code>path</code>	is a folder in the filesystem where the dump file resides.
<code>db-dump-file</code>	is the file previously created with the <code>dump</code> command of <code>neo4j-admin</code> .
<code>db-folder</code>	is the name of the database that will be created. The database is overwritten if <code>--force</code> is specified as <code>true</code> .

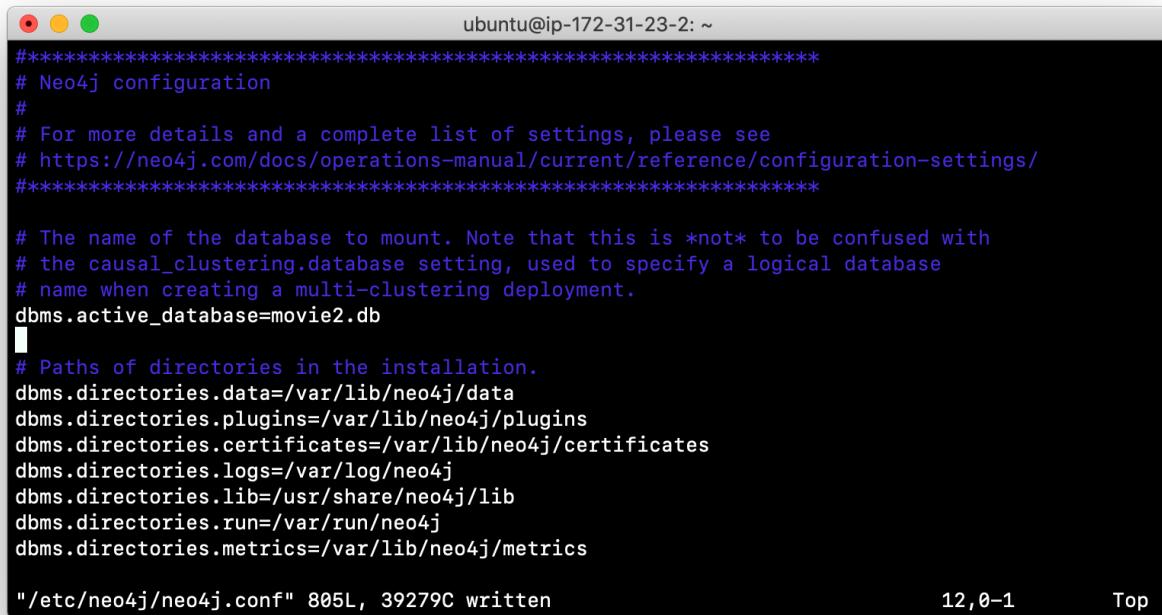
Here is an example where we load the contents of `movie-dump` into a database named `movie2.db`.



```
ubuntu@ip-172-31-23-2:~$ sudo /usr/bin/neo4j-admin load --database=movie2.db --from=/usr/local/dumps/movie-dump
ubuntu@ip-172-31-23-2:~$ ls -la /var/lib/neo4j/data/databases
total 20
drwxr-xr-x 5 neo4j adm 4096 Dec 28 19:04 .
drwxr-xr-x 4 neo4j adm 4096 Dec 27 22:37 ..
drwxr-xr-x 3 neo4j neo4j 4096 Dec 28 17:55 graph.db
drwxr-xr-x 4 neo4j neo4j 4096 Dec 28 15:22 movie.db
drwxr-xr-x 4 root root 4096 Dec 28 19:04 movie2.db
-rw-r--r-- 1 neo4j neo4j 0 Dec 27 22:38 store_lock
ubuntu@ip-172-31-23-2:~$
```

## Modifying config for new database

In order to access this newly created and loaded database, we must modify **neo4j.conf** to use **movie2.db** as the active database before starting the Neo4j instance:



```
ubuntu@ip-172-31-23-2: ~
*****
# Neo4j configuration
#
# For more details and a complete list of settings, please see
# https://neo4j.com/docs/operations-manual/current/reference/configuration-settings/
*****


# The name of the database to mount. Note that this is *not* to be confused with
# the causal_clustering.database setting, used to specify a logical database
# name when creating a multi-clustering deployment.
dbms.active_database=movie2.db

#
# Paths of directories in the installation.
dbms.directories.data=/var/lib/neo4j/data
dbms.directories.plugins=/var/lib/neo4j/plugins
dbms.directories.certificates=/var/lib/neo4j/certificates
dbms.directories.logs=/var/log/neo4j
dbms.directories.lib=/usr/share/neo4j/lib
dbms.directories.run=/var/run/neo4j
dbms.directories.metrics=/var/lib/neo4j/metrics

"/etc/neo4j/neo4j.conf" 805L, 39279C written          12,0-1      Top
```

In addition, we must change the owner:group for the database folder and its sub-folders to **neo4j:neo4j** before we start the Neo4j instance.

A best practice is to examine the log file for the Neo4j instance after you have made any configuration changes to ensure that the instance starts with no errors.

# Exercise #3: Copying a database

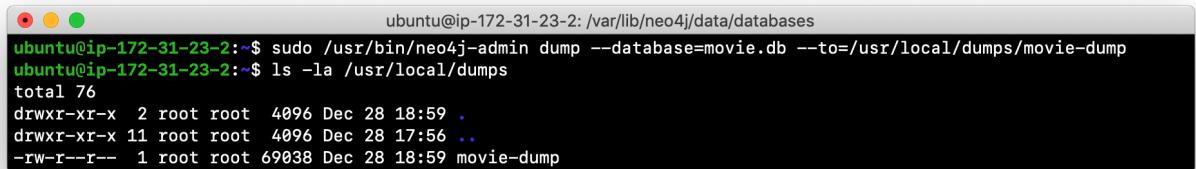
In this Exercise, you will make a copy of your active database that has the movie data in it and use the dump file to create a database.

## Before you begin

You should have loaded the **graph.db** database with the movie data (Exercise #2) and stopped the Neo4j instance.

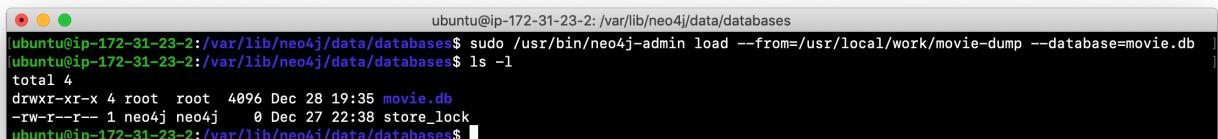
## Exercise steps:

1. Open a terminal on your system.
2. Create a folder named **/usr/local/work**.
3. Use the **neo4j-admin** script to dump the **graph.db** database to the **work** folder. You should do something like this:



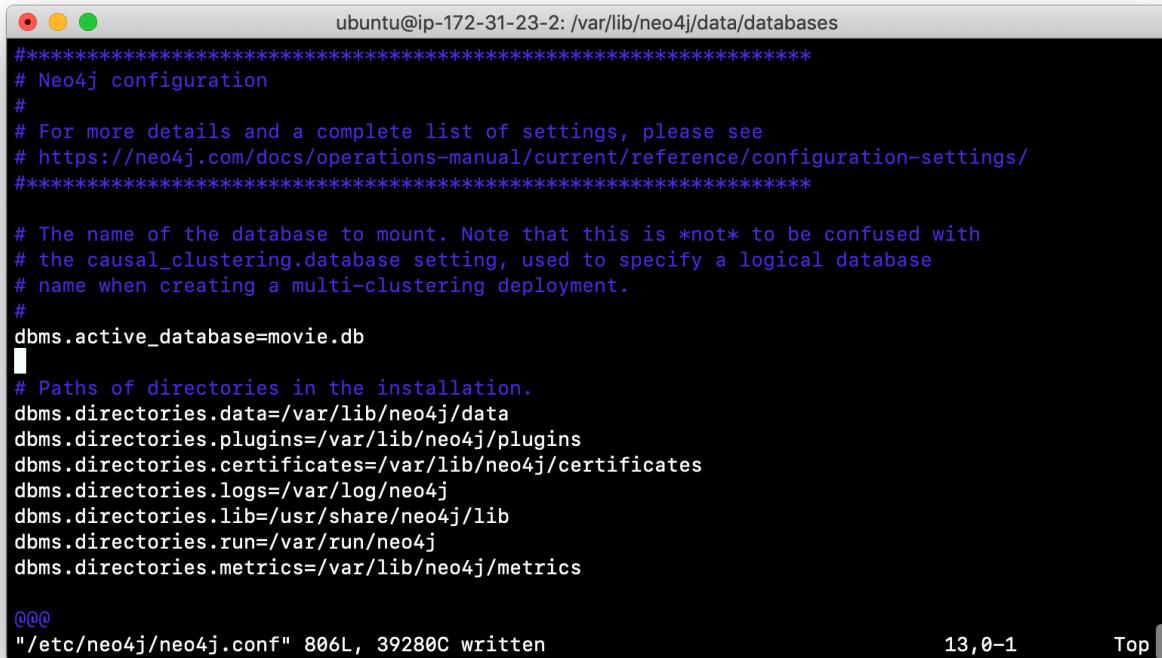
```
ubuntu@ip-172-31-23-2: /var/lib/neo4j/data/databases
ubuntu@ip-172-31-23-2:~$ sudo /usr/bin/neo4j-admin dump --database=movie.db --to=/usr/local/dumps/movie-dump
ubuntu@ip-172-31-23-2:~$ ls -la /usr/local/dumps
total 76
drwxr-xr-x  2 root root  4096 Dec 28 18:59 .
drwxr-xr-x 11 root root  4096 Dec 28 17:56 ..
-rw-r--r--  1 root root 69038 Dec 28 18:59 movie-dump
```

4. Notice that this dump file is simply a file that can be copied to any location.
5. Delete the **graph.db** database by removing the **graph.db** folder and its subfolders.
6. Use the **neo4j-admin** script to load the database from the dump file you just created. Name the database **movie.db**. You should do something like this:



```
ubuntu@ip-172-31-23-2: /var/lib/neo4j/data/databases
[ubuntu@ip-172-31-23-2:~/var/lib/neo4j/data/databases]$ sudo /usr/bin/neo4j-admin load --from=/usr/local/work/movie-dump --database=movie.db
[ubuntu@ip-172-31-23-2:~/var/lib/neo4j/data/databases]$ ls -l
total 4
drwxr-xr-x 4 root root 4096 Dec 28 19:35 movie.db
-rw-r--r-- 1 neo4j neo4j 0 Dec 27 22:38 store.lock
ubuntu@ip-172-31-23-2:~/var/lib/neo4j/data/databases$ ]
```

7. Modify **neo4j.conf** to use **movie.db** as the active database.



```
ubuntu@ip-172-31-23-2: /var/lib/neo4j/data/databases
*****
# Neo4j configuration
#
# For more details and a complete list of settings, please see
# https://neo4j.com/docs/operations-manual/current/reference/configuration-settings/
*****


# The name of the database to mount. Note that this is *not* to be confused with
# the causal_clustering.database setting, used to specify a logical database
# name when creating a multi-clustering deployment.
#
dbms.active_database=movie.db
# Paths of directories in the installation.
dbms.directories.data=/var/lib/neo4j/data
dbms.directories.plugins=/var/lib/neo4j/plugins
dbms.directories.certificates=/var/lib/neo4j/certificates
dbms.directories.logs=/var/log/neo4j
dbms.directories.lib=/usr/share/neo4j/lib
dbms.directories.run=/var/run/neo4j
dbms.directories.metrics=/var/lib/neo4j/metrics

@@@
"/etc/neo4j/neo4j.conf" 806L, 39280C written
```

13, 0-1

Top

8. If you did not perform the load as the user **neo4j**, you must change the owner:group of all files and folders under **movie.db** to be **neo4j:neo4j**. For example, change directory to the **movie.db** folder and then enter the command: **[sudo] chown -R neo4j:neo4j movie.db** This will recursively change the owner and group to all files and folders including and under the **movie.db** folder.
9. Start the Neo4j instance.
10. Examine the log file to ensure that the instance started with no errors.

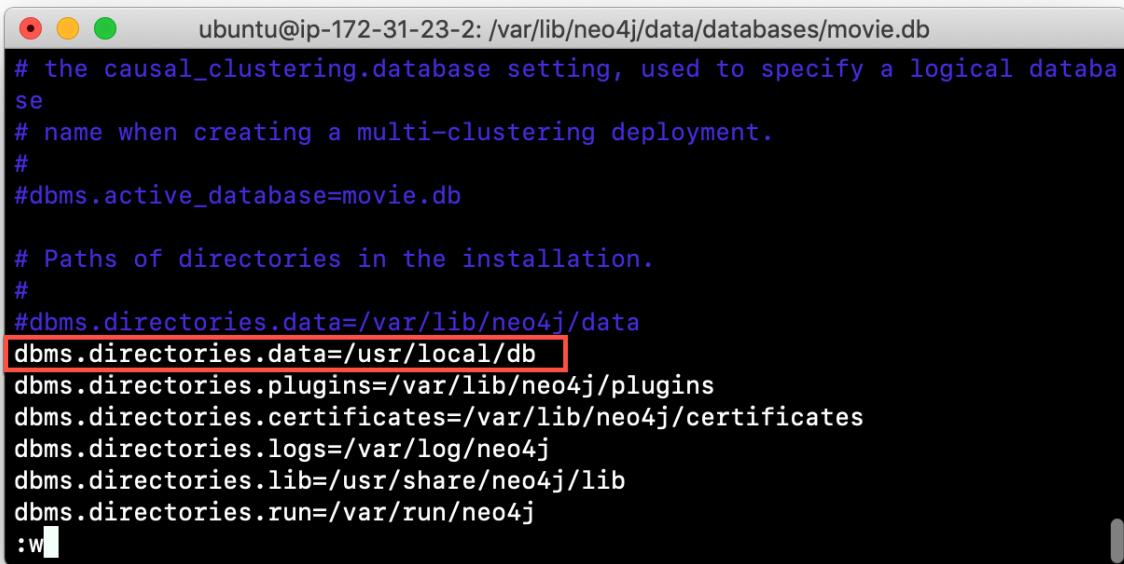
11. Access the database using **cypher-shell**. Can you see the movie data in the database?

```
ubuntu@ip-172-31-23-2: /var/lib/neo4j/data/databases/movie.db
[ubuntu@ip-172-31-23-2:/var/lib/neo4j/data/databases/movie.db$ /usr/bin/cypher-shell -u neo4j -p training-helps
Connected to Neo4j 3.5.1 at bolt://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.
[neo4j> MATCH (p:Person) WHERE p.name='Tom Hanks' RETURN p.name, p.born;
+-----+
| p.name      | p.born   |
+-----+
| "Tom Hanks" | 1956    |
+-----+
1 row available after 132 ms, consumed after another 23 ms
[neo4j> :exit
Bye!
ubuntu@ip-172-31-23-2:/var/lib/neo4j/data/databases/movie.db$ ]
```

# Modifying the location of the database

If you do not want the database used by the Neo4j instance to reside in the same location as the Neo4j installation, you can modify its location in the **neo4j.conf** file. If you specify a new location for the data, it must exist in the filesystem and the folder must be owned by *neo4j:neo4j*.

Here we have specified a new location for the data in the configuration file:



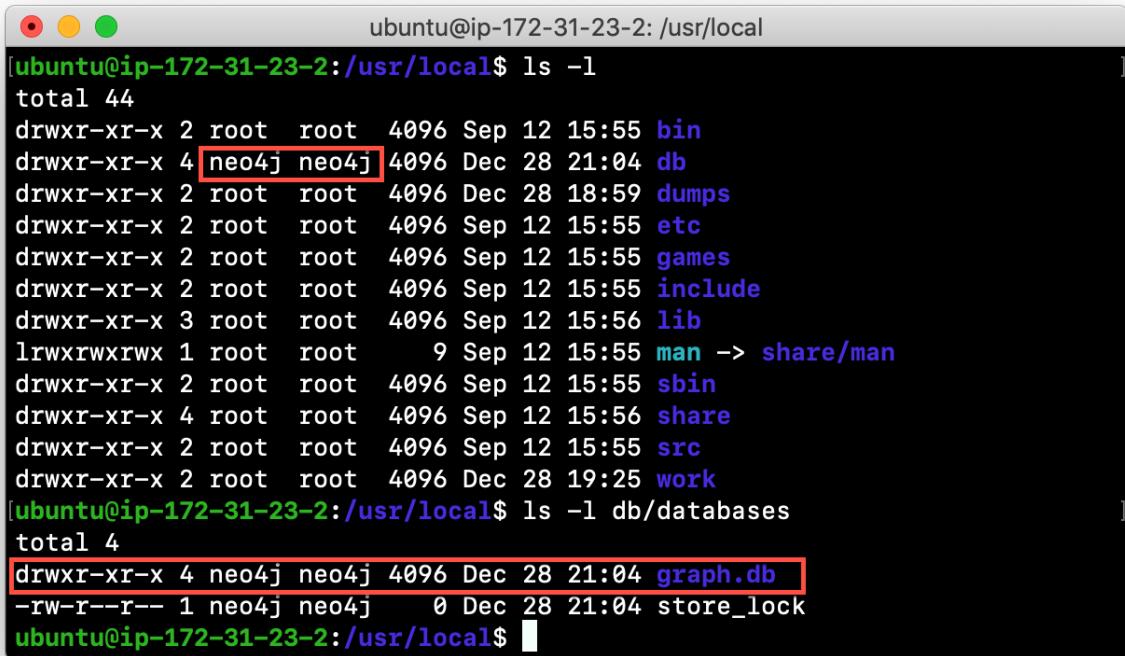
The screenshot shows a terminal window on an Ubuntu system (version 12.04 LTS) with the title bar "ubuntu@ip-172-31-23-2: /var/lib/neo4j/data/databases/movie.db". The window contains the contents of the neo4j.conf file. A red box highlights the line "dbms.directories.data=/usr/local/db". The file also includes other configuration settings like causal clustering, active database, and directory paths for plugins, certificates, logs, and libraries. At the bottom, there is a command ":w" indicating the file is being saved.

```
ubuntu@ip-172-31-23-2: /var/lib/neo4j/data/databases/movie.db
# the causal_clustering.database setting, used to specify a logical database
# name when creating a multi-clustering deployment.
#
#dbms.active_database=movie.db

# Paths of directories in the installation.
#
#dbms.directories.data=/var/lib/neo4j/data
dbms.directories.data=/usr/local/db
dbms.directories.plugins=/var/lib/neo4j/plugins
dbms.directories.certificates=/var/lib/neo4j/certificates
dbms.directories.logs=/var/log/neo4j
dbms.directories.lib=/usr/share/neo4j/lib
dbms.directories.run=/var/run/neo4j
:w
```

# Starting Neo4j instance with a new location

We ensure that the location for the data exists and then we can start the Neo4j instance. If this is the first time Neo4j has been started for this location, a new database named **graph.db** will be created. This is because we are using the default database name in the configuration file.



```
ubuntu@ip-172-31-23-2: /usr/local
[ubuntu@ip-172-31-23-2:/usr/local$ ls -l
total 44
drwxr-xr-x 2 root root 4096 Sep 12 15:55 bin
drwxr-xr-x 4 neo4j neo4j 4096 Dec 28 21:04 db
drwxr-xr-x 2 root root 4096 Dec 28 18:59 dumps
drwxr-xr-x 2 root root 4096 Sep 12 15:55 etc
drwxr-xr-x 2 root root 4096 Sep 12 15:55 games
drwxr-xr-x 2 root root 4096 Sep 12 15:55 include
drwxr-xr-x 3 root root 4096 Sep 12 15:56 lib
lrwxrwxrwx 1 root root 9 Sep 12 15:55 man -> share/man
drwxr-xr-x 2 root root 4096 Sep 12 15:55 sbin
drwxr-xr-x 4 root root 4096 Sep 12 15:56 share
drwxr-xr-x 2 root root 4096 Sep 12 15:55 src
drwxr-xr-x 2 root root 4096 Dec 28 19:25 work
[ubuntu@ip-172-31-23-2:/usr/local$ ls -l db/databases
total 4
drwxr-xr-x 4 neo4j neo4j 4096 Dec 28 21:04 graph.db
-rw-r--r-- 1 neo4j neo4j 0 Dec 28 21:04 store_lock
ubuntu@ip-172-31-23-2:/usr/local$ ]
```

If you have an existing database that you want to reside in a different location for the Neo4j instance, remember that you must dump and load the database to safely copy it to the new location.

# Using a different location for the database

If you are starting the Neo4j instance with a new location and do not want to use the default **graph.db** database, you must follow these steps to ensure that the folders for the database are set up properly:

1. Specify the new location in the configuration file, but do not specify the active database name.
2. Start or restart the Neo4j instance. A new **graph.db** folder will be created as well as the **dbms** folder required by the instance (contains important authentication information).
3. Examine the log file to ensure that it started without errors.
4. Stop the Neo4j instance.
5. Specify the name of the active database in the configuration file.
6. Load the data into the database name that will be the active database.
7. Ensure that the database folder and its subfolders are owned by *neo4j:neo4j*.
8. Start the Neo4j instance.
9. Examine the log file to ensure it started without errors.
10. Optionally, you can remove the **graph.db** folder as you will be working with a different database you loaded.

# Exercise #4: Modifying the location of the database

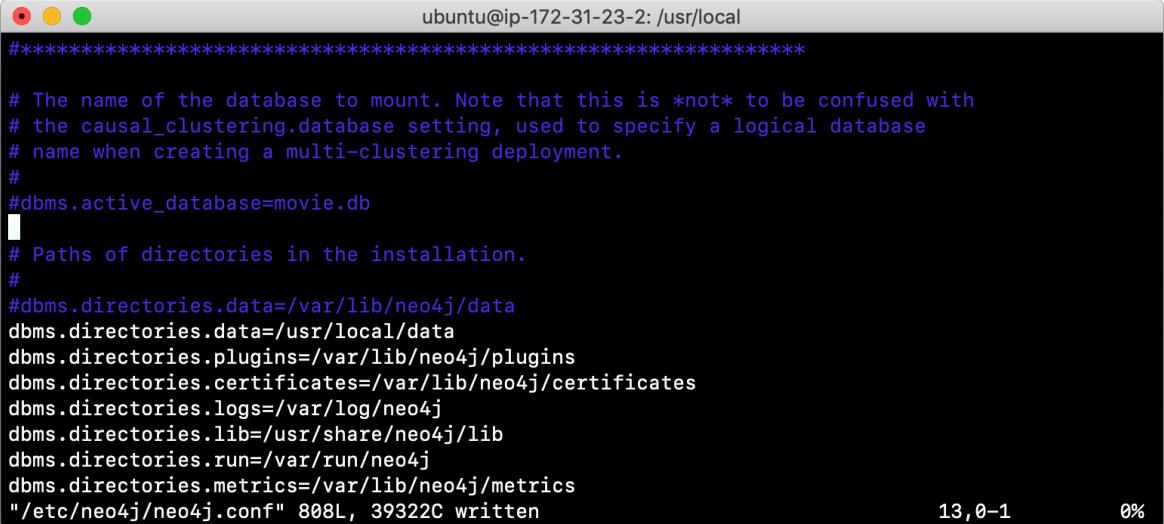
In this Exercise, you will set up a different location for the database in your local filesystem and start the Neo4j instance using the database at this new location.

## Before you begin

1. You should have created the dump file for the movie database (Exercise #3).
2. Stop the Neo4j instance.

## Exercise steps:

1. Open a terminal on your system.
2. Create a folder named **/usr/local/data**. This is the folder where the database will reside which is different from the default location used by Neo4j.
3. Make sure that this **data** folder is owned by *neo4j:neo4j*. For example, navigate to the **/usr/local** folder and enter **[sudo ]chown neo4j:neo4j data**.
4. Modify the **neo4j.conf** file to use **/usr/local/data** as the data directory. Also ensure that there is no active database specified. Your **neo4j.conf** file should look something like this:



The screenshot shows a terminal window on a Linux system (Ubuntu) with the command prompt `ubuntu@ip-172-31-23-2: /usr/local`. The window displays the configuration file `neo4j.conf` with the following content:

```
# ****
# The name of the database to mount. Note that this is *not* to be confused with
# the causal_clustering.database setting, used to specify a logical database
# name when creating a multi-clustering deployment.
#
#dbms.active_database=movie.db
#
# Paths of directories in the installation.
#
#dbms.directories.data=/var/lib/neo4j/data
dbms.directories.data=/usr/local/data
dbms.directories.plugins=/var/lib/neo4j/plugins
dbms.directories.certificates=/var/lib/neo4j/certificates
dbms.directories.logs=/var/log/neo4j
dbms.directories.lib=/usr/share/neo4j/lib
dbms.directories.run=/var/run/neo4j
dbms.directories.metrics=/var/lib/neo4j/metrics
"/etc/neo4j/neo4j.conf" 808L, 39322C written
```

The status bar at the bottom right of the terminal window shows `13,0-1` and `0%`.

5. Start the Neo4j instance.
6. Examine the log file to ensure that the instance started without errors.

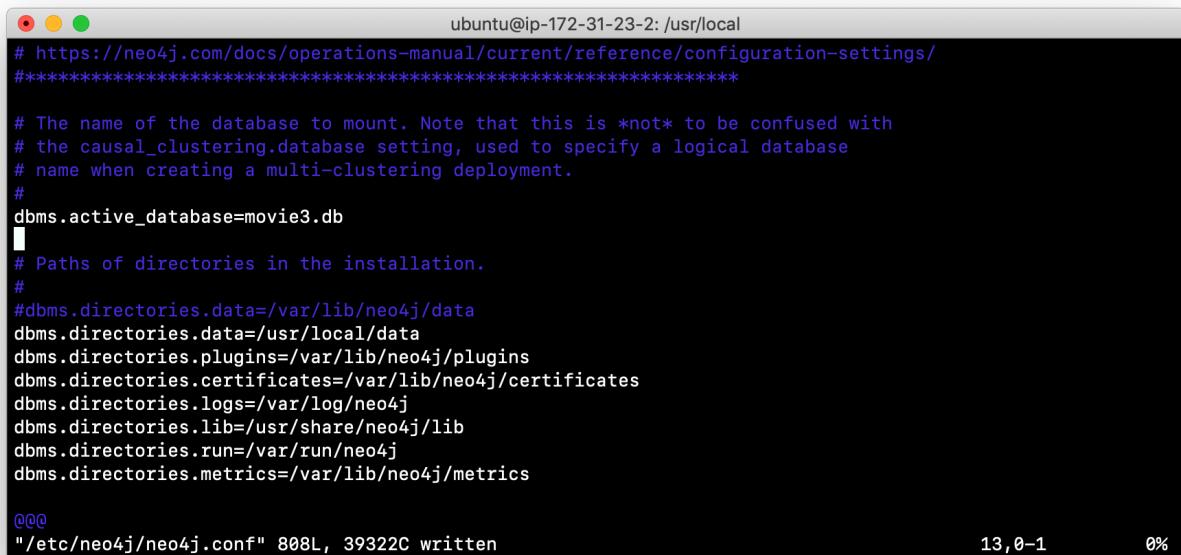
7. Examine the files in the `/usr/local/data` location. The instance should have created the **databases** and **dbms** folders. They should look as follows:



```
ubuntu@ip-172-31-23-2: /usr/local$ ls -l data
total 8
drwxr-xr-x 3 neo4j neo4j 4096 Dec 28 22:18 databases
drwxr-xr-x 2 neo4j neo4j 4096 Dec 28 22:18 dbms
ubuntu@ip-172-31-23-2: /usr/local$ ls -l data/databases
total 4
drwxr-xr-x 4 neo4j neo4j 4096 Dec 28 22:18 graph.db
-rw-r--r-- 1 neo4j neo4j     0 Dec 28 22:18 store_lock
ubuntu@ip-172-31-23-2: /usr/local$
```

8. Stop the Neo4j instance.

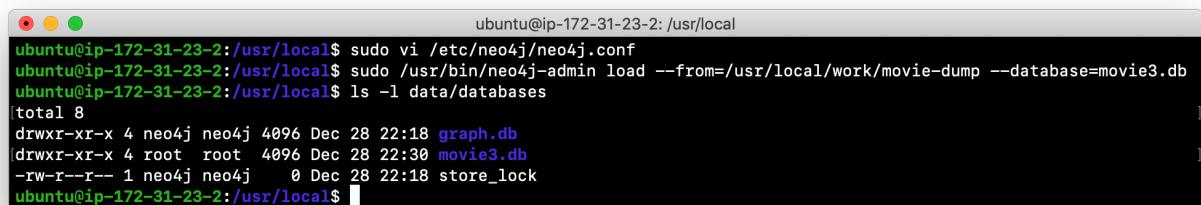
9. Modify the **neo4j.conf** file to use **movie3.db** as the active database. Your **neo4j.conf** file should look something like this:



```
ubuntu@ip-172-31-23-2: /usr/local
# https://neo4j.com/docs/operations-manual/current/reference/configuration-settings/
#*****#
# The name of the database to mount. Note that this is *not* to be confused with
# the causal_clustering.database setting, used to specify a logical database
# name when creating a multi-clustering deployment.
#
dbms.active_database=movie3.db
#
# Paths of directories in the installation.
#
#dbms.directories.data=/var/lib/neo4j/data
dbms.directories.data=/usr/local/data
dbms.directories.plugins=/var/lib/neo4j/plugins
dbms.directories.certificates=/var/lib/neo4j/certificates
dbms.directories.logs=/var/log/neo4j
dbms.directories.lib=/usr/share/neo4j/lib
dbms.directories.run=/var/run/neo4j
dbms.directories.metrics=/var/lib/neo4j/metrics

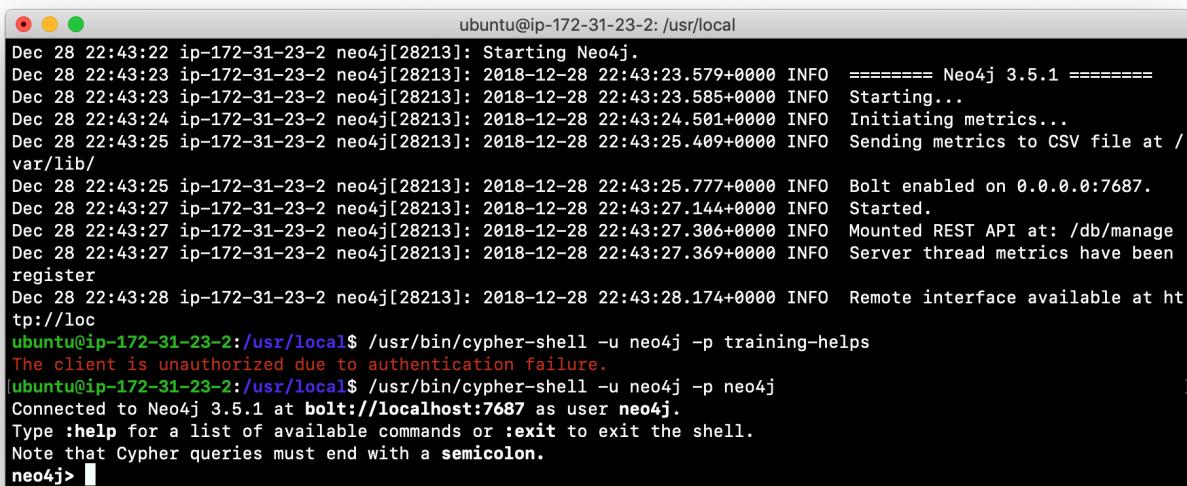
@@@
"/etc/neo4j/neo4j.conf" 808L, 39322C written
13,0-1          0%
```

10. Use the **neo4j-admin** script to load the database from the dump file you created in Exercise 3. Name the database **movie3.db** You should do something like this:



```
ubuntu@ip-172-31-23-2: /usr/local$ sudo vi /etc/neo4j/neo4j.conf
ubuntu@ip-172-31-23-2: /usr/local$ sudo /usr/bin/neo4j-admin load --from=/usr/local/work/movie-dump --database=movie3.db
ubuntu@ip-172-31-23-2: /usr/local$ ls -l data/databases
total 8
drwxr-xr-x 4 neo4j neo4j 4096 Dec 28 22:18 graph.db
drwxr-xr-x 4 root  root  4096 Dec 28 22:30 movie3.db
-rw-r--r-- 1 neo4j neo4j     0 Dec 28 22:18 store_lock
ubuntu@ip-172-31-23-2: /usr/local$
```

11. Ensure that all files and folders including and under **movie3.db** are owned by *neo4j:neo4j*. For example, change directory to the **databases** folder and then enter the command: `[sudo] chown -R neo4j:neo4j movie3.db` This will recursively change the owner and group to all files and folders under **movie3.db**.
12. Start the Neo4j instance.
13. Examine the log file to ensure that no errors occurred.
14. Access the database using **cypher-shell**. Do you get an authentication error? This is because the database is now located in a different location and the default credentials of *neo4j/neo4j* are used.



```
ubuntu@ip-172-31-23-2: /usr/local
Dec 28 22:43:22 ip-172-31-23-2 neo4j[28213]: Starting Neo4j.
Dec 28 22:43:23 ip-172-31-23-2 neo4j[28213]: 2018-12-28 22:43:23.579+0000 INFO ===== Neo4j 3.5.1 =====
Dec 28 22:43:23 ip-172-31-23-2 neo4j[28213]: 2018-12-28 22:43:23.585+0000 INFO Starting...
Dec 28 22:43:24 ip-172-31-23-2 neo4j[28213]: 2018-12-28 22:43:24.501+0000 INFO Initiating metrics...
Dec 28 22:43:25 ip-172-31-23-2 neo4j[28213]: 2018-12-28 22:43:25.409+0000 INFO Sending metrics to CSV file at /
var/lib/
Dec 28 22:43:25 ip-172-31-23-2 neo4j[28213]: 2018-12-28 22:43:25.777+0000 INFO Bolt enabled on 0.0.0.0:7687.
Dec 28 22:43:27 ip-172-31-23-2 neo4j[28213]: 2018-12-28 22:43:27.144+0000 INFO Started.
Dec 28 22:43:27 ip-172-31-23-2 neo4j[28213]: 2018-12-28 22:43:27.306+0000 INFO Mounted REST API at: /db/manage
Dec 28 22:43:27 ip-172-31-23-2 neo4j[28213]: 2018-12-28 22:43:27.369+0000 INFO Server thread metrics have been
register
Dec 28 22:43:28 ip-172-31-23-2 neo4j[28213]: 2018-12-28 22:43:28.174+0000 INFO Remote interface available at ht
tp://loc
ubuntu@ip-172-31-23-2:/usr/local$ /usr/bin/cypher-shell -u neo4j -p training-helps
The client is unauthorized due to authentication failure.
[ubuntu@ip-172-31-23-2:/usr/local$ /usr/bin/cypher-shell -u neo4j -p neo4j
Connected to Neo4j 3.5.1 at bolt://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.
neo4j> ]
```

15. Enter the Cypher statement to change the password: `CALL dbms.changePassword('newPassword');`
16. Enter a Cypher statement to retrieve some data: `MATCH (p:Person) WHERE p.name='Meg Ryan' RETURN p.name, p.born;`

17. Exit `cypher-shell`.

```
ubuntu@ip-172-31-23-2: /usr/local$ /usr/bin/cypher-shell -u neo4j -p training-helps
The client is unauthorized due to authentication failure.
ubuntu@ip-172-31-23-2: /usr/local$ /usr/bin/cypher-shell -u neo4j -p neo4j
Connected to Neo4j 3.5.1 at bolt://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.
neo4j> CALL dbms.changePassword('training-helps');
0 rows available after 9 ms, consumed after another 0 ms
neo4j> MATCH (p:Person) WHERE p.name="Meg Ryan" RETURN p.name, p.born;
+-----+
| p.name      | p.born |
+-----+
| "Meg Ryan" | 1961   |
+-----+
1 row available after 137 ms, consumed after another 24 ms
neo4j> :exit

Bye!
ubuntu@ip-172-31-23-2: /usr/local$ /usr/bin/cypher-shell -u neo4j -p training-helps
Connected to Neo4j 3.5.1 at bolt://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.
neo4j> :exit

Bye!
ubuntu@ip-172-31-23-2: /usr/local$
```

# Checking the consistency of a database

A database's consistency could be compromised if a software or hardware failure has occurred that affects the Neo4j instance. You will learn later in this module about live backups, but if you have reason to believe that a specific database has been corrupted, you can perform a consistency check on it.

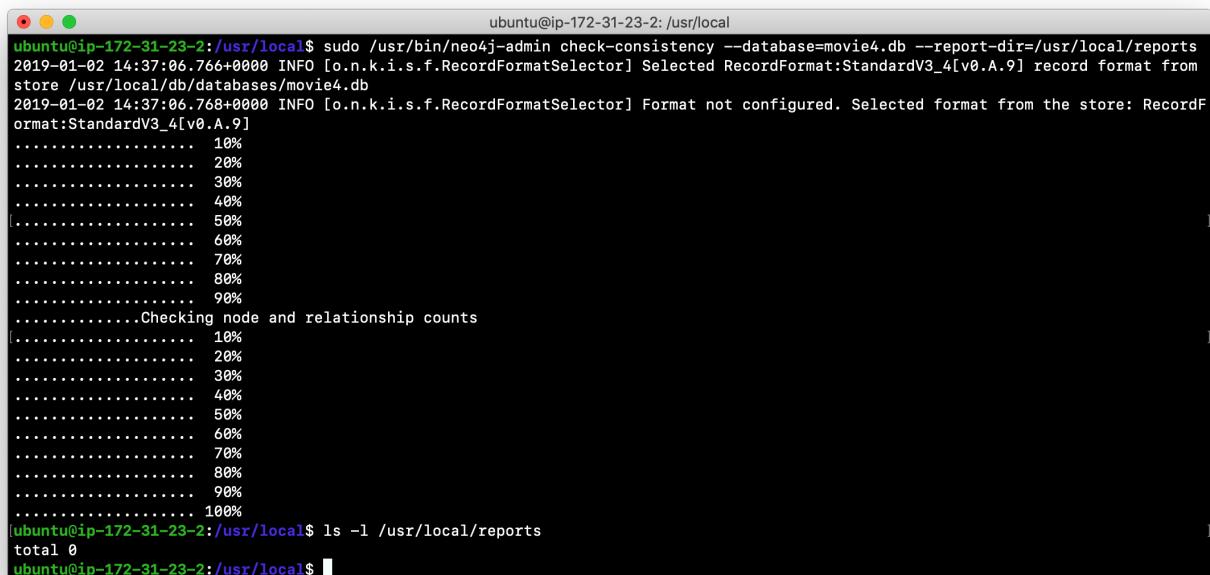
The Neo4j instance must be stopped to perform the consistency check.

Here is how you use the `neo4j-admin` tool to check the consistency of the database:

```
[sudo] neo4j-admin check-consistency --database=db-name --report-dir=report-location [--verbose=true]
```

The database named *db-name* is found in the data location specified in **neo4j.conf** file. If the tool comes back with no error, then the database is consistent. Otherwise, an error is returned and a report is written to *report-location*. You can specify verbose reporting. See the [Neo4j Operations Manual](#) for more options. For example, you can check the consistency of a backup which is a best practice.

Suppose we had loaded the **movie4.db** database with `neo4j-admin`. Here is what a successful run of the consistency checker should produce:

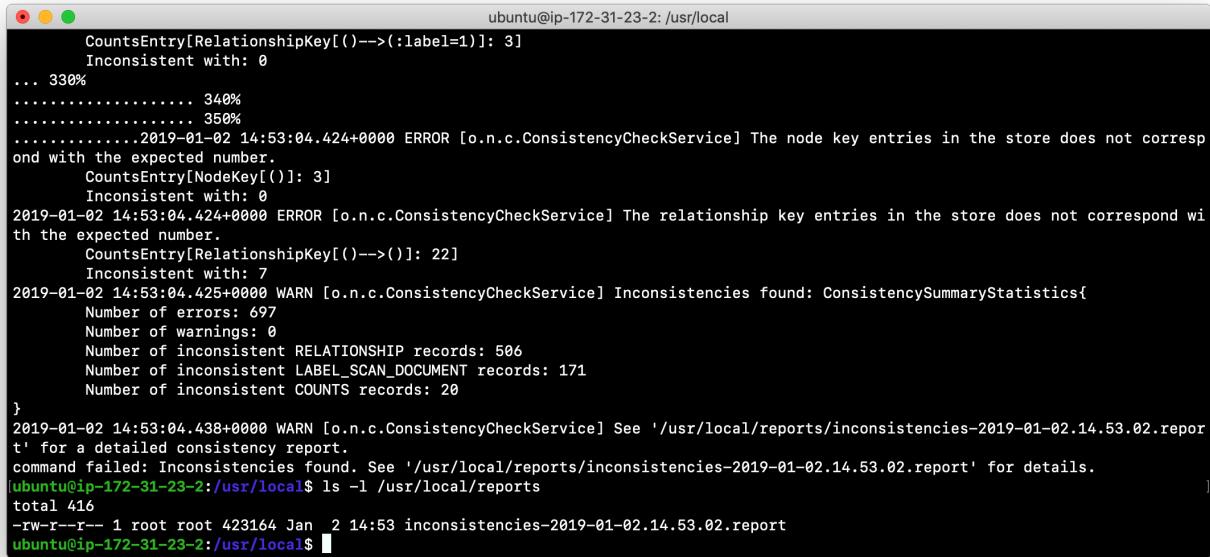


```
ubuntu@ip-172-31-23-2:/usr/local$ sudo /usr/bin/neo4j-admin check-consistency --database=movie4.db --report-dir=/usr/local/reports
2019-01-02 14:37:06.766+0000 INFO [o.n.k.i.s.f.RecordFormatSelector] Selected RecordFormat:StandardV3_4[v0.A.9] record format from
store /usr/local/db/databases/movie4.db
2019-01-02 14:37:06.768+0000 INFO [o.n.k.i.s.f.RecordFormatSelector] Format not configured. Selected format from the store: RecordF
ormat:StandardV3_4[v0.A.9]
..... 10%
..... 20%
..... 30%
..... 40%
..... 50%
..... 60%
..... 70%
..... 80%
..... 90%
..... Checking node and relationship counts
..... 10%
..... 20%
..... 30%
..... 40%
..... 50%
..... 60%
..... 70%
..... 80%
..... 90%
..... 100%
[ubuntu@ip-172-31-23-2:/usr/local$ ls -l /usr/local/reports
total 0
ubuntu@ip-172-31-23-2:/usr/local$ ]
```

No report is written to the reports folder because the consistency check passed.

# Inconsistencies found

Here is an example of what an unsuccessful run of the consistency checker should produce:



```
ubuntu@ip-172-31-23-2: /usr/local
CountsEntry[RelationshipKey[()-->(:label=1)]: 3]
Inconsistent with: 0
... 330%
..... 340%
.... 350%
2019-01-02 14:53:04.424+0000 ERROR [o.n.c.ConsistencyCheckService] The node key entries in the store does not correspond with the expected number.
CountsEntry[NodeKey[()]: 3]
Inconsistent with: 0
2019-01-02 14:53:04.424+0000 ERROR [o.n.c.ConsistencyCheckService] The relationship key entries in the store does not correspond with the expected number.
CountsEntry[RelationshipKey[()-->()): 22]
Inconsistent with: 7
2019-01-02 14:53:04.425+0000 WARN [o.n.c.ConsistencyCheckService] Inconsistencies found: ConsistencySummaryStatistics{
    Number of errors: 697
    Number of warnings: 0
    Number of inconsistent RELATIONSHIP records: 506
    Number of inconsistent LABEL_SCAN_DOCUMENT records: 171
    Number of inconsistent COUNTS records: 20
}
2019-01-02 14:53:04.438+0000 WARN [o.n.c.ConsistencyCheckService] See '/usr/local/reports/inconsistencies-2019-01-02.14.53.02.report' for a detailed consistency report.
command failed: Inconsistencies found. See '/usr/local/reports/inconsistencies-2019-01-02.14.53.02.report' for details.
[ubuntu@ip-172-31-23-2:/usr/local]$ ls -l /usr/local/reports
total 416
-rw-r--r-- 1 root root 423164 Jan  2 14:53 inconsistencies-2019-01-02.14.53.02.report
ubuntu@ip-172-31-23-2:/usr/local$
```

If inconsistencies are found, a report is generated and placed in the folder specified for the report location.

Inconsistencies in a database are a serious matter that should be looked into with the help of Neo4j Technical Support.

# Exercise #5: Checking consistency of a database

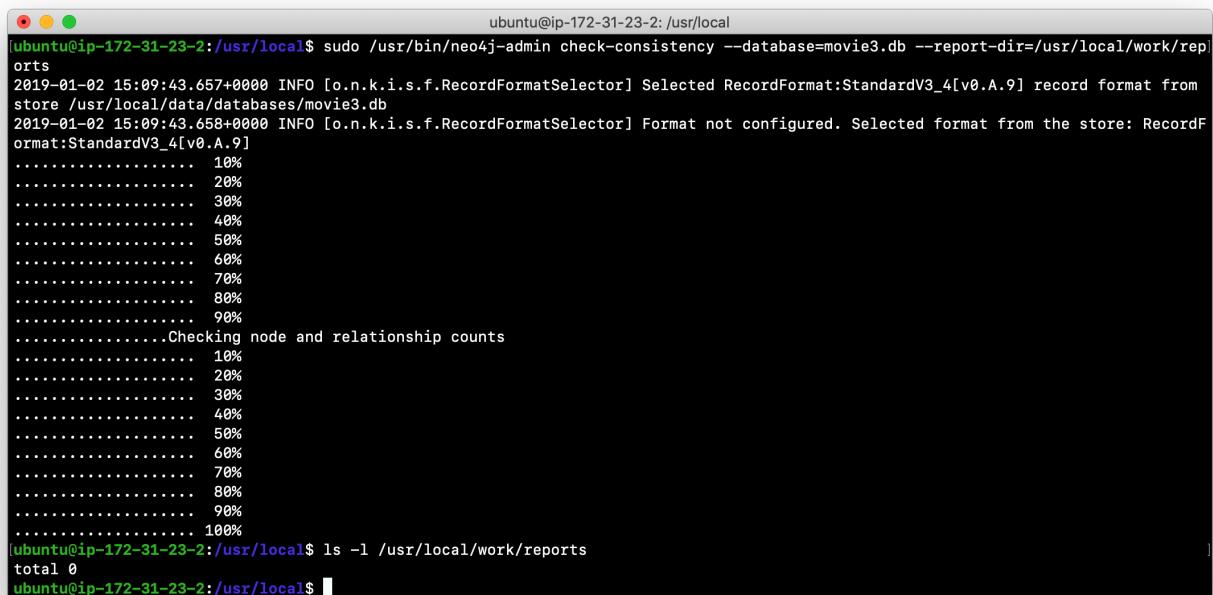
In this Exercise, you check the consistency of a database that is consistent. Then you modify a file that causes the database to become corrupt and then check its consistency.

## Before you begin

1. You should have created and started the **movie3.db** database (Exercise #4).
2. Stop the Neo4j instance.
3. Create a folder named **/usr/local/work/reports**.

## Exercise steps:

1. Open a terminal on your system.
2. Run the consistency check tool on **movie3.db** using **neo4j-admin** specifying **reports** as the folder where the report will be written. The consistency check tool should return the following:



```
ubuntu@ip-172-31-23-2:/usr/local$ sudo /usr/bin/neo4j-admin check-consistency --database=movie3.db --report-dir=/usr/local/work/reports
2019-01-02 15:09:43.657+0000 INFO [o.n.k.i.s.f.RecordFormatSelector] Selected RecordFormat:StandardV3_4[v0.A.9] record format from store /usr/local/data/databases/movie3.db
2019-01-02 15:09:43.658+0000 INFO [o.n.k.i.s.f.RecordFormatSelector] Format not configured. Selected format from the store: RecordFormat:StandardV3_4[v0.A.9]
..... 10%
..... 20%
..... 30%
..... 40%
..... 50%
..... 60%
..... 70%
..... 80%
..... 90%
..... Checking node and relationship counts
..... 10%
..... 20%
..... 30%
..... 40%
..... 50%
..... 60%
..... 70%
..... 80%
..... 90%
..... 100%
[ubuntu@ip-172-31-23-2:/usr/local$ ls -l /usr/local/work/reports
total 0
ubuntu@ip-172-31-23-2:/usr/local$ ]
```

3. Modify the Neo4j configuration to use a database named **movie3-copy.db**, rather than **movie3.db**.
4. Use **neo4j-admin** to create and load **movie3-copy.db** from the movie dump file you created earlier.
5. Ensure that the owner of the **movie3-copy.db** is **neo4j:neo4j**.
6. Next, you will corrupt the database. Modify the file **movie3-copy.db/neostore.nodesstore.db** by adding some text to the file.

7. Run the consistency check tool on **movie3-copy.db** using **neo4j-admin** specifying **/usr/local/work/reports** as the folder where the report will be written. The consistency check tool should return something like the following:

```
ubuntu@ip-172-31-23-2:/usr/local
.....2019-01-02 15:15:23.861+0000 ERROR [o.n.c.ConsistencyCheckService] The node key entries in the store does not correspond wi
th the expected number.
CountsEntry[NodeKey[()]: 3]
Inconsistent with: 8
2019-01-02 15:15:23.861+0000 ERROR [o.n.c.ConsistencyCheckService] The relationship key entries in the store does not correspond wi
th the expected number.
CountsEntry[RelationshipKey[()-->()): 22]
Inconsistent with: 43
.. 50%
..... 60%
..... 70%
..... 80%
..... 90%
..... 100%
2019-01-02 15:15:23.862+0000 WARN [o.n.c.ConsistencyCheckService] Inconsistencies found: ConsistencySummaryStatistics{
    Number of errors: 364
    Number of warnings: 0
    Number of inconsistent NODE records: 11
    Number of inconsistent RELATIONSHIP records: 238
    Number of inconsistent LABEL_SCAN_DOCUMENT records: 95
    Number of inconsistent COUNTS records: 20
}
2019-01-02 15:15:23.876+0000 WARN [o.n.c.ConsistencyCheckService] See '/usr/local/work/reports/inconsistencies-2019-01-02.15.15.22.
report' for a detailed consistency report.
command failed: Inconsistencies found. See '/usr/local/work/reports/inconsistencies-2019-01-02.15.15.22.report' for details.
[ubuntu@ip-172-31-23-2:/usr/local$ ls -l /usr/local/work/reports
total 228
-rw-r--r-- 1 root root 230564 Jan  2 15:15 inconsistencies-2019-01-02.15.15.22.report
ubuntu@ip-172-31-23-2:/usr/local$ ]
```

# Scripting with cypher-shell

As a database administrator, you may need to automate changes to the database. The most common types of changes that administrators may want to perform are operations such as adding/dropping constraints or indexes. Note that you will need to work with the developers and architects of your application to determine what indexes must be created. You can create scripts that forward the Cypher statements to `cypher-shell`. The number of supporting script files you create will depend upon the tasks you want to perform against the database.

## Examples: Adding constraints

Suppose that we use `bash`. We create 3 files:

1. **AddConstraints.cypher** that contains the Cypher statements to execute in `cypher-shell`:

```
CREATE CONSTRAINT ON (m:Movie) ASSERT m.title IS UNIQUE;  
CREATE CONSTRAINT ON (p:Person) ASSERT p.name IS UNIQUE;  
CALL db.constraints();
```

Each Cypher statement must end with a `;`.

2. **AddConstraints.sh** that invokes `cypher-shell` using a set of Cypher statements and specifies verbose output:

```
cat /usr/local/work/AddConstraints.cypher | /usr/bin/cypher-shell -u neo4j -p  
training-helps --format verbose
```

3. **PrepareDB.sh** that initializes the log file, **PrepareDB.log**, and calls the script to add the constraints:

```
rm -rf /usr/local/work/PrepareDB.log  
/usr/local/work/AddConstraints.sh 2>&1 >> /usr/local/work/  
PrepareDB.log  
# Other scripts here
```

When the **PrepareDB.sh** script runs its scripts, all output will be written to the log file, including error output. Then you can simply check the log file to make sure it ran as expected.

# Exercise #6: Scripting changes to the database

In this Exercise, you will gain experience scripting with **cypher-shell**. You will create three files in the **/usr/local/work** folder:

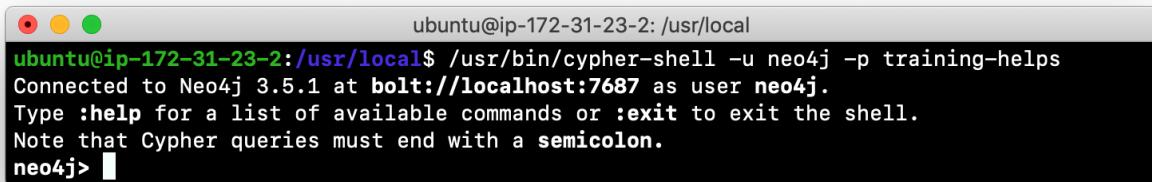
1. **AddConstraints.cypher**
2. **AddConstraints.sh**
3. **MaintainDB.sh**

## Before you begin

1. Remove the **databases/movie3-copy.db** folder as this database is now corrupt.
2. Ensure that the Neo4j configuration uses **movie3.db** for the database.
3. Start or restart the Neo4j instance.

## Exercise steps:

1. Open a terminal on your system.
2. Start **cypher-shell**, providing the credentials for the *neo4j* user.



A screenshot of a terminal window titled "ubuntu@ip-172-31-23-2: /usr/local". The window shows the command `/usr/bin/cypher-shell -u neo4j -p training-helps` being run. The output indicates a connection to Neo4j 3.5.1 at bolt://localhost:7687 as user neo4j. It provides help information and notes about Cypher queries ending with a semicolon. The prompt ends with `neo4j>`.

3. Enter some simple Cypher statements to confirm that you can access the database. For example:
  - a. `CALL db.schema();`
  - b. `CALL db.constraints();`
4. Exit Cypher-shell by typing `:exit`.
5. Create a Cypher script in the **/usr/local/work** folder named **AddConstraints.cypher** with the following statements:

```
CREATE CONSTRAINT ON (m:Movie) ASSERT m.title IS UNIQUE;  
CREATE CONSTRAINT ON (p:Person) ASSERT p.name IS UNIQUE;
```

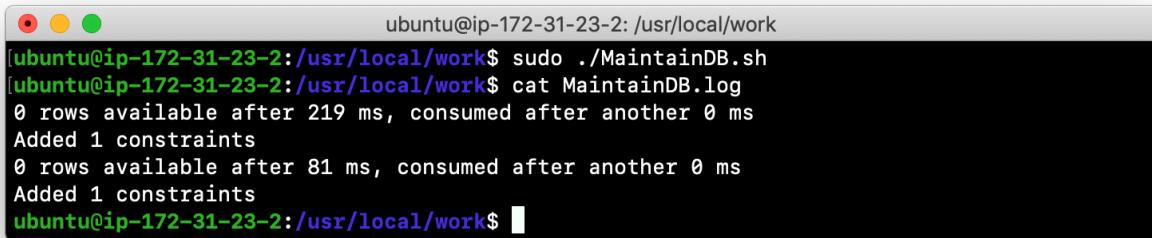
6. Create a shell script in the **/usr/local/work** folder named **AddConstraints.sh** that will forward **AddConstraints.cypher** to **cypher-shell**. This file should have the following contents:

```
cat /usr/local/work/AddConstraints.cypher | /usr/bin/cypher-shell -u neo4j -p  
training-helps --format verbose
```

7. Create a shell script in the `/usr/local/work` folder named **MaintainDB.sh** that will initialize the log file and then call **AddConstraints.sh**. This file should have the following contents:

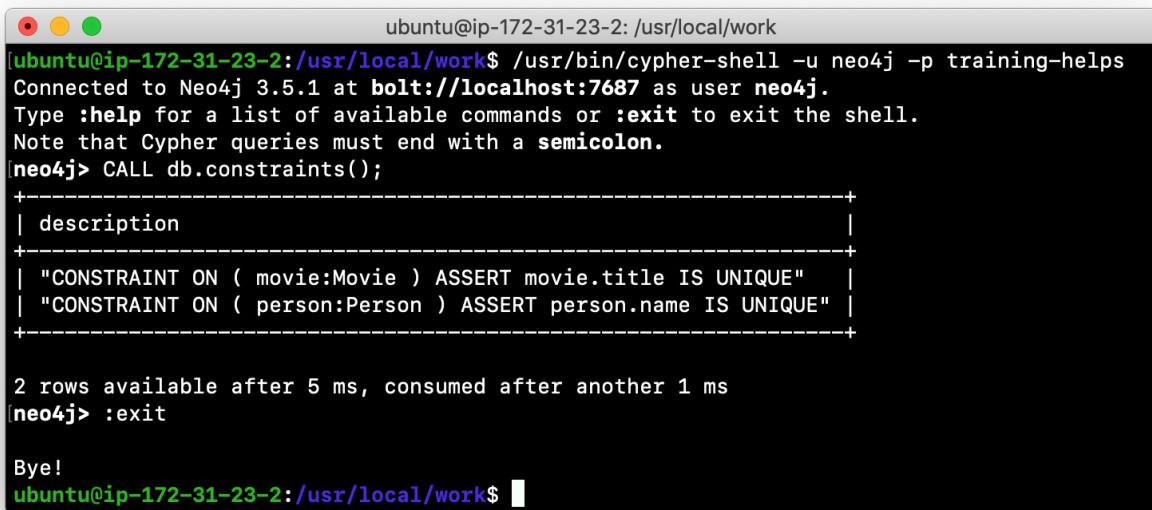
```
rm -rf /usr/local/work/MaintainDB.log  
/usr/local/work/AddConstraints.sh 2>&1 >> /usr/local/work/MaintainDB.log
```

8. Ensure that the scripts you created have execute permissions.  
9. Run the **MaintainDB.sh** script and view the log file.



```
ubuntu@ip-172-31-23-2: /usr/local/work$ sudo ./MaintainDB.sh  
ubuntu@ip-172-31-23-2: /usr/local/work$ cat MaintainDB.log  
0 rows available after 219 ms, consumed after another 0 ms  
Added 1 constraints  
0 rows available after 81 ms, consumed after another 0 ms  
Added 1 constraints  
ubuntu@ip-172-31-23-2: /usr/local/work$
```

10. Confirm that it created the constraints in the database. (Check using cypher-shell (`CALL db.constraints();`))



```
ubuntu@ip-172-31-23-2: /usr/local/work$ /usr/bin/cypher-shell -u neo4j -p training-helps  
Connected to Neo4j 3.5.1 at bolt://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.  
neo4j> CALL db.constraints();  
+-----+  
| description |  
+-----+  
| "CONSTRAINT ON ( movie:Movie ) ASSERT movie.title IS UNIQUE" |  
| "CONSTRAINT ON ( person:Person ) ASSERT person.name IS UNIQUE" |  
+-----+  
  
2 rows available after 5 ms, consumed after another 1 ms  
neo4j> :exit  
Bye!  
ubuntu@ip-172-31-23-2: /usr/local/work$
```

# Managing plugins

Some applications can use Neo4j out-of-the-box, but many applications require additional functionality that could be:

- A library supported by Neo4j such as GraphQL or GRAPH ALGORITHMS.
- A community-supported library, such as APOC.
- Custom functionality that has been written by the developers of your application.

We refer to this additional functionality as a *plugin* that contains specific procedures. A *plugin* is typically specific to a particular release of Neo4j. In many cases, if you upgrade to a later version of Neo4j, you may also need to install a new *plugin*. First, you should understand how to view the procedures available for use with the Neo4j instance. You do so by executing the Cypher statement `CALL db.procedures();`

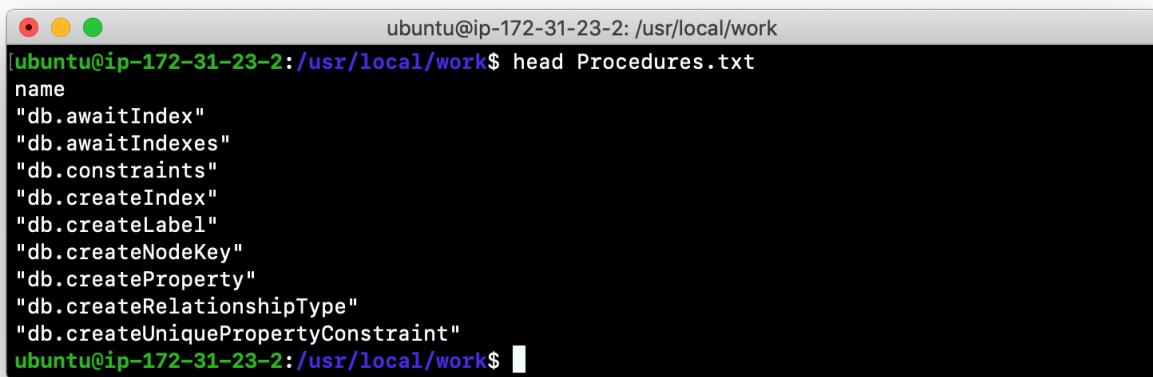
## Retrieving available procedures

Here is an example of a script you can run to produce a file, **Procedures.txt** that contain the names of the procedures currently available for the Neo4j instance:

```
echo "CALL dbms.procedures() YIELD name;" | /usr/bin/cypher-shell -u neo4j -p  
training-helps --format plain > /usr/local/work/Procedures.txt
```

This script calls `dbms.procedures()` to return the name of each procedure in the list returned.

Here is a view of **Procedures.txt**:



The screenshot shows a terminal window on an Ubuntu system. The command `head Procedures.txt` is run, displaying the first few lines of the `Procedures.txt` file. The output lists various Neo4j built-in procedures, all starting with "db." followed by a procedure name like "awaitIndex", "awaitIndexes", etc.

```
ubuntu@ip-172-31-23-2: /usr/local/work$ head Procedures.txt
name
"db.awaitIndex"
"db.awaitIndexes"
"db.constraints"
"db.createIndex"
"db.createLabel"
"db.createNodeKey"
"db.createProperty"
"db.createRelationshipType"
"db.createUniquePropertyConstraint"
ubuntu@ip-172-31-23-2: /usr/local/work$
```

By default, the procedures available to the Neo4j instance are the built-in procedures that are named `db.*` and `dbms.*`.

# Adding a plugin to the Neo4j instance

To add a plugin to your Neo4j instance, you must first obtain the **.jar** file. It is important to confirm that the **.jar** file you will use is compatible with the version of Neo4j that you are using. For example, a plugin released for release 3.4 of Neo4j can be used by a Neo4j 3.5 instance, but the converse may not be true. You must check with the developers of the plugin for compatibility.

Some plugins require a configuration change. You should understand the configuration changes required for any plugin you are installing.

**NOTE** When you install Neo4j, the **plugins** folder contains a **README.txt** folder that contains instructions related to sandboxing and whitelisting. These instructions will change in future releases of Neo4j.

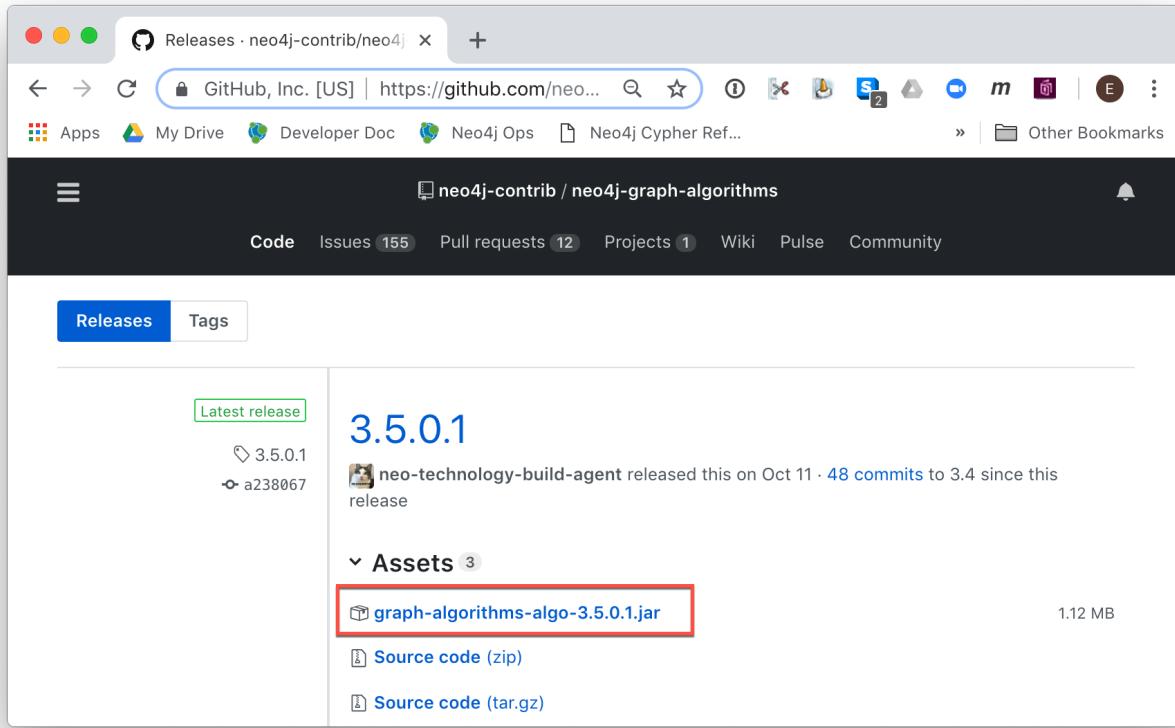
## Sandboxing and whitelisting

Neo4j provides *sandboxing* to ensure that procedures do not inadvertently use insecure APIs. For example, when writing custom code it is possible to access Neo4j APIs that are not publicly supported, and these internal APIs are subject to change, without notice. Additionally, their use comes with the risk of performing insecure actions. The sandboxing functionality limits the use of extensions to publicly supported APIs, which exclusively contain safe operations, or contain security checks.

Neo4j *whitelisting* can be used to allow loading only a few extensions from a larger library. The configuration setting `dbms.security.procedures.whitelist` is used to name certain procedures that should be available from a library. It defines a comma-separated list of procedures that are to be loaded. The list may contain both fully-qualified procedure names, and partial names with the wildcard `*`.

# Example: Installing the Graph Algorithms plugin

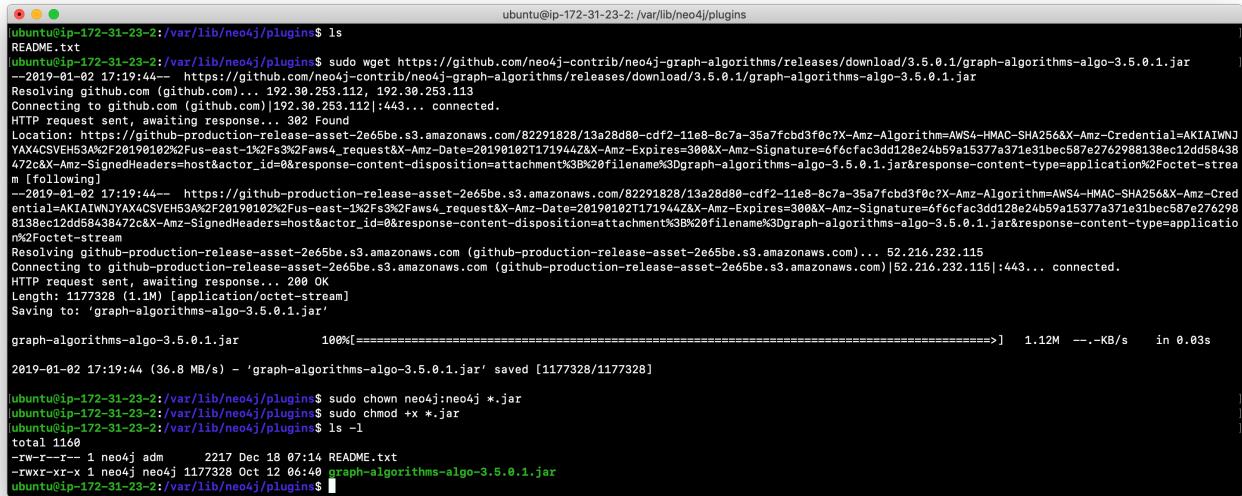
Suppose we wanted to install the Graph Algorithms library that is compatible with Neo4j 3.5. We find the library in GitHub and simply download the **.jar** file. Here is the [release area](#) in GitHub for the graph algorithms library:



The main page for [Graph Algorithms](#) in GitHub contains details about the plugin and instructions for installing it.

## Example: Download and ownership of plugin

You download any plugins that your application will use to the /var/lib/neo4j/plugins folder:



```
ubuntu@ip-172-31-23-2:/var/lib/neo4j/plugins$ ls
README.txt
ubuntu@ip-172-31-23-2:/var/lib/neo4j/plugins$ sudo wget https://github.com/neo4j-contrib/neo4j-graph-algorithms/releases/download/3.5.0.1/graph-algorithms-algo-3.5.0.1.jar
--2019-01-02 17:19:44-- https://github.com/neo4j-contrib/neo4j-graph-algorithms/releases/download/3.5.0.1/graph-algorithms-algo-3.5.0.1.jar
Resolving github.com (github.com)... 192.30.253.112, 192.30.253.113
Connecting to github.com (github.com)|192.30.253.112|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://github-production-release-asset-2e65be.s3.amazonaws.com/82291828/13a28d80-cdf2-11e8-8c7a-35a7fcb3f0c?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNUXAX4CSVEH53A%2F20190102%2Fus-east-1%2F%2Faws4_request&X-Amz-Date=20190102T171944Z&X-Amz-Expires=3008X&X-Amz-Signature=6fcfa3dd128e24b59a1537a371e31bec87e2762988138ec12dd58438472c&X-Amz-SignedHeaders=host&actor_id=0&response-content-disposition=attachment%3B%20filename%3Dgraph-algorithms-algo-3.5.0.1.jar&response-content-type=application%2Foctet-stream [following]
--2019-01-02 17:19:44-- https://github-production-release-asset-2e65be.s3.amazonaws.com/82291828/13a28d80-cdf2-11e8-8c7a-35a7fcb3f0c?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNUXAX4CSVEH53A%2F20190102%2Fus-east-1%2F%2Faws4_request&X-Amz-Date=20190102T171944Z&X-Amz-Expires=3008X&X-Amz-Signature=6fcfa3dd128e24b59a1537a371e31bec87e2762988138ec12dd58438472c&X-Amz-SignedHeaders=host&actor_id=0&response-content-disposition=attachment%3B%20filename%3Dgraph-algorithms-algo-3.5.0.1.jar&response-content-type=application%2Foctet-stream
Resolving github-production-release-asset-2e65be.s3.amazonaws.com (github-production-release-asset-2e65be.s3.amazonaws.com)... 52.216.232.115
Connecting to github-production-release-asset-2e65be.s3.amazonaws.com (github-production-release-asset-2e65be.s3.amazonaws.com)|52.216.232.115|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1177328 (1.1M) [application/octet-stream]
Saving to: 'graph-algorithms-algo-3.5.0.1.jar'

graph-algorithms-algo-3.5.0.1.jar      100%[=====]  1.12M --.-KB/s   in 0.03s

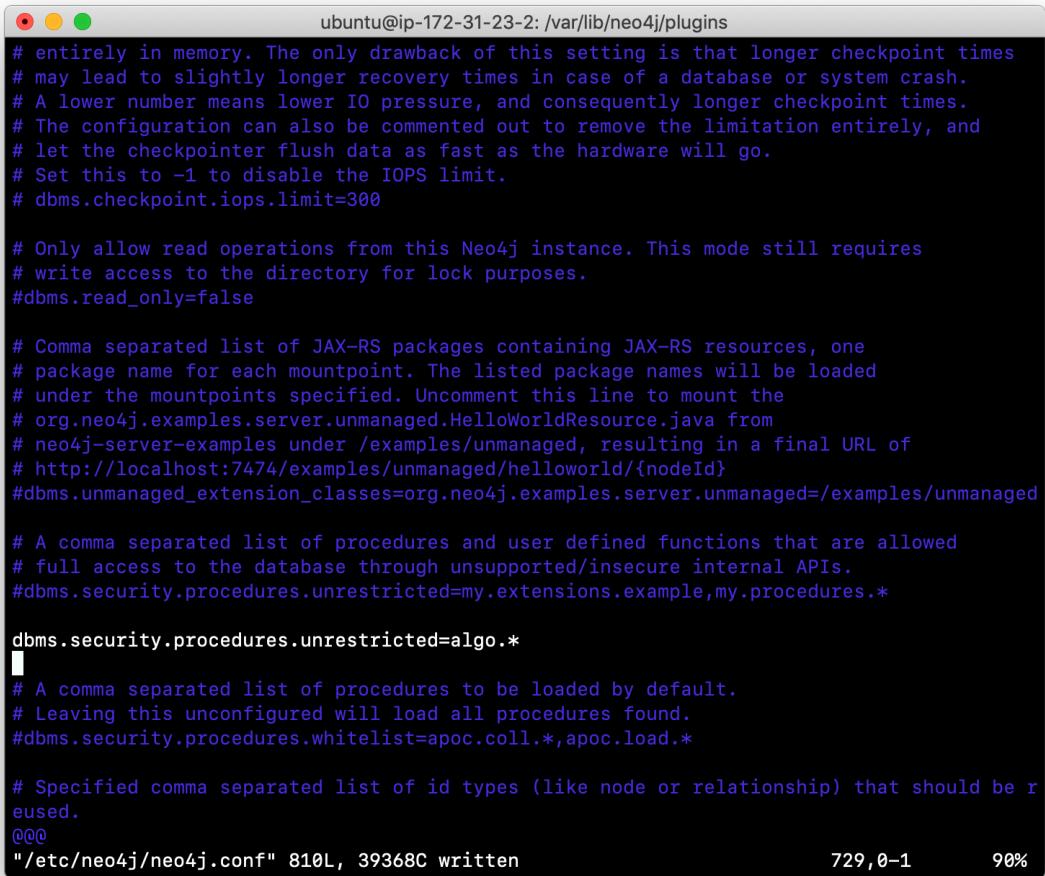
2019-01-02 17:19:44 (36.8 MB/s) - 'graph-algorithms-algo-3.5.0.1.jar' saved [1177328/1177328]

ubuntu@ip-172-31-23-2:/var/lib/neo4j/plugins$ sudo chown neo4j:neo4j *.jar
ubuntu@ip-172-31-23-2:/var/lib/neo4j/plugins$ sudo chmod +x *.jar
ubuntu@ip-172-31-23-2:/var/lib/neo4j/plugins$ ls -l
total 1160
-rw-r--r-- 1 neo4j adm    2217 Dec 18 07:14 README.txt
-rwxr-xr-x 1 neo4j neo4j 1177328 Oct 12 06:48 graph-algorithms-algo-3.5.0.1.jar
ubuntu@ip-172-31-23-2:/var/lib/neo4j/plugins$
```

Ensure that the .jar file is owned by *neo4j:neo4j* and that it has execute permissions.

## Example: Sandboxing

The graph algorithms plugin requires *sandboxing*. Here is how we enable the procedures in the graph algorithms plugin. We modify the *Miscellaneous Configuration* section of the **neo4j.conf** file as follows:



A screenshot of a terminal window titled "ubuntu@ip-172-31-23-2: /var/lib/neo4j/plugins". The window contains the configuration file for the Neo4j database, specifically the "neo4j.conf" file located at "/etc/neo4j/neo4j.conf". The configuration includes settings for checkpointing, security, and procedure loading. The terminal shows the configuration file with syntax highlighting for comments and keywords. The bottom right corner of the terminal window displays the status "729, 0-1 90%".

```
# entirely in memory. The only drawback of this setting is that longer checkpoint times
# may lead to slightly longer recovery times in case of a database or system crash.
# A lower number means lower IO pressure, and consequently longer checkpoint times.
# The configuration can also be commented out to remove the limitation entirely, and
# let the checkpointer flush data as fast as the hardware will go.
# Set this to -1 to disable the IOPS limit.
#dbms.checkpoint.iops.limit=300

# Only allow read operations from this Neo4j instance. This mode still requires
# write access to the directory for lock purposes.
#dbms.read_only=false

# Comma separated list of JAX-RS packages containing JAX-RS resources, one
# package name for each mountpoint. The listed package names will be loaded
# under the mountpoints specified. Uncomment this line to mount the
# org.neo4j.examples.server.unmanaged.HelloWorldResource.java from
# neo4j-server-examples under /examples/unmanaged, resulting in a final URL of
# http://localhost:7474/examples/unmanaged/helloworld/{nodeId}
#dbms.unmanaged_extension_classes=org.neo4j.examples.server.unmanaged=/examples/unmanaged

# A comma separated list of procedures and user defined functions that are allowed
# full access to the database through unsupported/insecure internal APIs.
#dbms.security.procedures.unrestricted=my.extensions.example,my.procedures./*

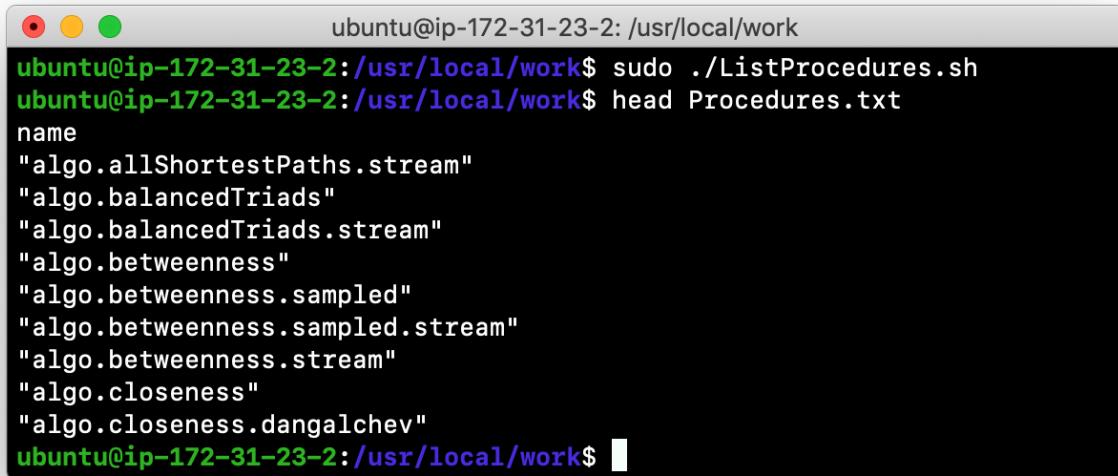
dbms.security.procedures.unrestricted=algo./*

# A comma separated list of procedures to be loaded by default.
# Leaving this unconfigured will load all procedures found.
#dbms.security.procedures.whitelist=apoc.coll.*,apoc.load./*

# Specified comma separated list of id types (like node or relationship) that should be reused.
@@@
"/etc/neo4j/neo4j.conf" 810L, 39368C written
```

## Example: Restart with plugin

You must then start or restart the Neo4j instance. Once started, you can then run the script to return the names of the procedures that are available to the Neo4j instance. Here we see that we have the additional procedures for the graph algorithms plugin:

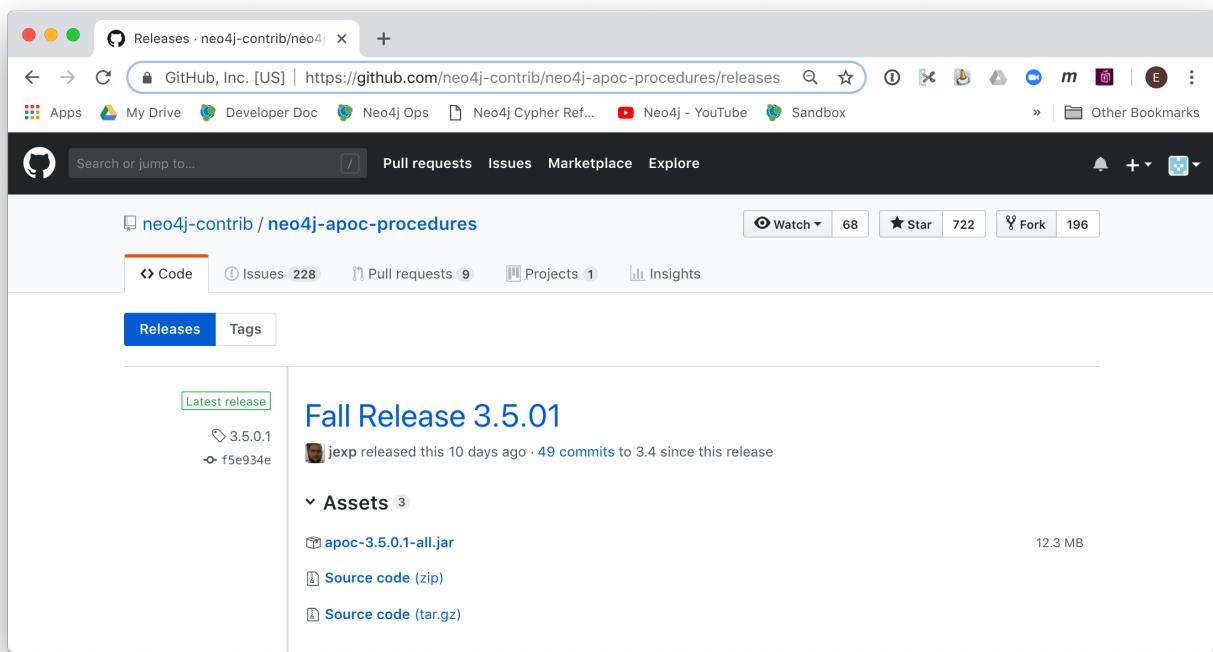


```
ubuntu@ip-172-31-23-2: /usr/local/work
ubuntu@ip-172-31-23-2:/usr/local/work$ sudo ./ListProcedures.sh
ubuntu@ip-172-31-23-2:/usr/local/work$ head Procedures.txt
name
"algo.allShortestPaths.stream"
"algo.balancedTriads"
"algo.balancedTriads.stream"
"algo.betweenness"
"algo.betweenness.sampled"
"algo.betweenness.sampled.stream"
"algo.betweenness.stream"
"algo.closeness"
"algo.closeness.dangalchev"
ubuntu@ip-172-31-23-2:/usr/local/work$ █
```

# Example: Installing the APOC plugin

APOC (Awesome Procedures on Cypher) is a very popular plugin used by many applications. It contains over 450 user-defined procedures that make accessing a graph incredibly efficient and much easier than writing your own Cypher statements to do the same thing.

You obtain the plugin from the APOC [releases](#) page:



## Example: Download and ownership of plugin

Here we download the .jar file, change its permissions to execute, and change the owner to be `neo4j:neo4j`.

```
ubuntu@ip-172-31-23-2: /var/lib/neo4j/plugins$ sudo wget https://github.com/neo4j-contrib/neo4j-apoc-procedures/releases/download/3.5.0.1/apoc-3.5.0.1-all.jar
--2019-01-02 17:40:25-- https://github.com/neo4j-contrib/neo4j-apoc-procedures/releases/download/3.5.0.1/apoc-3.5.0.1-all.jar
Resolving github.com (github.com)... 192.30.253.112, 192.30.253.113
Connecting to github.com (github.com)|192.30.253.112|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://github-production-release-asset-2e65be.s3.amazonaws.com/52509220/5fe94600-f257-11e8-8298-df34f5b8d7f0?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNYAX4CSVEH53A%2F20190102%2Fsus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20190102T174025Z&X-Amz-Expires=300&X-Amz-Signature=0014673ec5729ba2e0a6985d8c958248e960f925d9c9c6d41d6265c94bcfb14c&X-Amz-SignedHeaders=host&actor_id=0&response-content-disposition=attachment%3B%20filename%3Dapo
c-3.5.0.1-all.jar&response-content-type=application%2Foctet-stream [following]
--2019-01-02 17:40:25-- https://github-production-release-asset-2e65be.s3.amazonaws.com/52509220/5fe94600-f257-11e8-8298-df34f5b8d7f0?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNYAX4CSVEH53A%2F20190102%2Fsus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20190102T174025Z&X-Amz-Expires=300&X-Amz-Signature=0014673ec5729ba2e0a6985d8c958248e960f925d9c9c6d41d6265c94bcfb14c&X-Amz-SignedHeaders=host&actor_id=0&response-content-disposition=attachment%3B%20filename%3Dapo
c-3.5.0.1-all.jar&response-content-type=application%2Foctet-stream
Resolving github-production-release-asset-2e65be.s3.amazonaws.com (github-production-release-asset-2e65be.s3.amazonaws.com)... 52.216.9.115
Connecting to github-production-release-asset-2e65be.s3.amazonaws.com (github-production-release-asset-2e65be.s3.amazonaws.com)|52.216.9.115|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 12880869 (12M) [application/octet-stream]
Saving to: 'apoc-3.5.0.1-all.jar'

apoc-3.5.0.1-all. 100%[=====] 12.28M --.-KB/s   in 0.08s

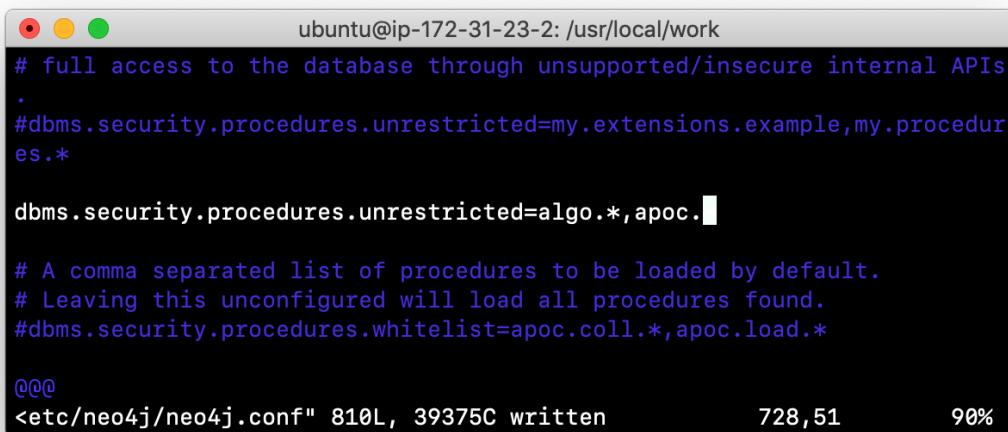
2019-01-02 17:40:25 (146 MB/s) - 'apoc-3.5.0.1-all.jar' saved [12880869/12880869]

[ubuntu@ip-172-31-23-2: /var/lib/neo4j/plugins$ sudo chown neo4j:neo4j *.jar
[ubuntu@ip-172-31-23-2: /var/lib/neo4j/plugins$ sudo chmod +x *.jar
[ubuntu@ip-172-31-23-2: /var/lib/neo4j/plugins$ ls -l
total 13744
-rw-r--r-- 1 neo4j adm      2217 Dec 18 07:14 README.txt
-rwxr-xr-x 1 neo4j neo4j 12880869 Nov 27 14:16 apoc-3.5.0.1-all.jar
-rw-rxr-xr-x 1 neo4j neo4j 1177328 Oct 12 06:40 graph-algorithms-algo-3.5.0.1.jar
ubuntu@ip-172-31-23-2: /var/lib/neo4j/plugins$ ]
```

## Example: Sandboxing

After you have placed the **.jar** file into the **plugins** folder, you must modify the configuration for the instance as described in the main page for APOC. As described on this page, you have an option of either *sandboxing* or *whitelisting* the procedures of the plugin. How much of the APOC library is used by your application is determined by the developers so you should use them as a resource for this type of configuration change.

Suppose we want to allow all APOC procedures to be available to this Neo4j instance. We would sandbox the plugin in the **neo4j.conf** file as follows, similar to how we sandboxed the graph algorithms where we specify **dbms.security.procedures.unrestricted=algo.,apoc..**



```
ubuntu@ip-172-31-23-2: /usr/local/work
# full access to the database through unsupported/insecure internal APIs
.
#dbms.security.procedures.unrestricted=my.extensions.example,my.procedures.*

dbms.security.procedures.unrestricted=algo.*,apoc.* █

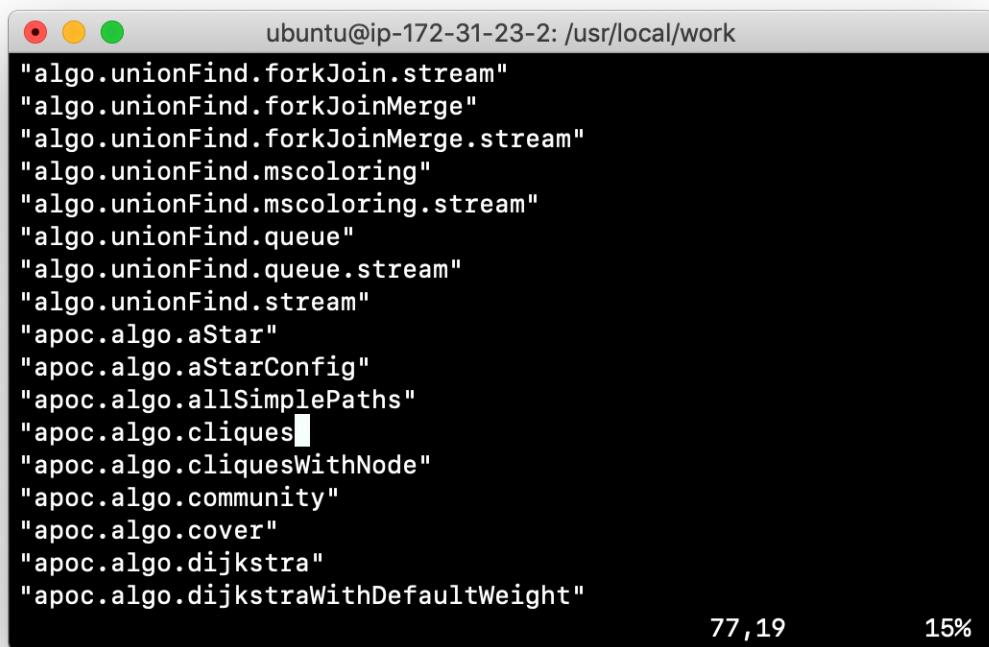
# A comma separated list of procedures to be loaded by default.
# Leaving this unconfigured will load all procedures found.
#dbms.security.procedures.whitelist=apoc.coll.*,apoc.load.*

@QQ
<etc/neo4j/neo4j.conf" 810L, 39375C written          728,51      90%
```

Since APOC is large, you will most likely want to whitebox specific procedures so that only the procedures needed by the application are loaded into the Neo4j instance at runtime.

## Example: Restart with plugin

And here we see the results after restarting the Neo4j instance and running the script to list the procedures loaded in the instance:



A screenshot of a terminal window titled "ubuntu@ip-172-31-23-2: /usr/local/work". The window contains a list of Neo4j procedures, all starting with double quotes and followed by a procedure name. The list includes:

- "algo.unionFind.forkJoin.stream"
- "algo.unionFind.forkJoinMerge"
- "algo.unionFind.forkJoinMerge.stream"
- "algo.unionFind.mscoloring"
- "algo.unionFind.mscoloring.stream"
- "algo.unionFind.queue"
- "algo.unionFind.queue.stream"
- "algo.unionFind.stream"
- "apoc.algo.aStar"
- "apoc.algo.aStarConfig"
- "apoc.algo.allSimplePaths"
- "apoc.algo.cliques"
- "apoc.algo.cliquesWithNode"
- "apoc.algo.community"
- "apoc.algo.cover"
- "apoc.algo.dijkstra"
- "apoc.algo.dijkstraWithDefaultWeight"

The terminal window has three colored status indicators at the top left (red, yellow, green). At the bottom right, there are two small numbers: "77,19" and "15%".

# Exercise #7: Install a plugin

In this Exercise, you will install the Spatial library for use by your Neo4j instance and you will create and execute a script to report all of the procedures available to the Neo4j instance.

## Before you begin:

1. Stop the Neo4j instance.
2. Make sure you have a terminal window open for executing test commands.

## Exercise steps:

1. In a Web browser, go to the GitHub repository for the [Neo4j Spacial Library](#).
2. On the main page for this repository, find the latest release of the library that is compatible with your version of Neo4j Enterprise Edition.
3. Download the already-built **.jar** file into the **/var/lib/neo4j/plugins** folder.
4. Ensure that the file size is correct and that the file name ends with **.jar**.
5. Change the owner of the **.jar** file to **neo4j:neo4j** and add execute permissions to the file.
6. Restart the Neo4j instance.
7. Follow the steps on the GitHub page for testing the library.

For example, you should see the following in the repository main page:

- o v0.25.5 for Neo4j 3.3.5
- o v0.25.6 for Neo4j 3.4.5
- o v0.25.7 for Neo4j 3.4.9

For versions up to 0.15-neo4j-2.3.4:

```
#install the plugin
unzip neo4j-spatial-XXXX-server-plugin.zip -d $NEO4J_HOME/plugins

#start the server
$NEO4J_HOME/bin/neo4j start

#list REST API (edit to correct password)
curl -u neo4j:neo4j http://localhost:7474/db/data/
```

For versions for neo4j 3.0 and later:

```
#install the plugin
cp neo4j-spatial-XXXX-server-plugin.jar $NEO4J_HOME/plugins/

#start the server
$NEO4J_HOME/bin/neo4j start

#list REST API (edit to correct password)
curl -u neo4j:neo4j http://localhost:7474/db/data/

#list spatial procedures (edit to correct password)
curl -u neo4j:neo4j -H "Content-Type: application/json" -X POST -d '{"query":"CALL spatial.procedures"}'
```

Here is how you download the .jar file into the /var/lib/neo4j/plugins folder. You should confirm that the file size is correct and that the owner is neo4j:neo4j with execute permissions.

```
ubuntu@ip-172-31-23-2:~$ cd /var/lib/neo4j/plugins
ubuntu@ip-172-31-23-2:/var/lib/neo4j/plugins$ sudo wget https://github.com/neo4j-contrib/m2/blob/master/releases/org/neo4j/neo4j-spatial/0.25.7-neo4j-3.4.9/neo4j-spatial-0.25.7-neo4j-3.4.9-server-plugin.jar?raw=true
--2019-01-02 18:14:45-- https://github.com/neo4j-contrib/m2/blob/master/releases/org/neo4j/neo4j-spatial/0.25.7-neo4j-3.4.9/neo4j-spatial-0.25.7-neo4j-3.4.9-server-plugin.jar?raw=true
Resolving github.com (github.com)... 192.30.253.113, 192.30.253.112
Connecting to github.com (github.com)|192.30.253.113|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://github.com/neo4j-contrib/m2/raw/master/releases/org/neo4j/neo4j-spatial/0.25.7-neo4j-3.4.9/neo4j-spatial-0.25.7-neo4j-3.4.9-server-plugin.jar [following]
--2019-01-02 18:14:45-- https://github.com/neo4j-contrib/m2/raw/master/releases/org/neo4j/neo4j-spatial/0.25.7-neo4j-3.4.9/neo4j-spatial-0.25.7-neo4j-3.4.9-server-plugin.jar
Reusing existing connection to github.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/neo4j-contrib/m2/master/releases/org/neo4j/neo4j-spatial/0.25.7-neo4j-3.4.9/neo4j-spatial-0.25.7-neo4j-3.4.9-server-plugin.jar [following]
--2019-01-02 18:14:45-- https://raw.githubusercontent.com/neo4j-contrib/m2/master/releases/org/neo4j/neo4j-spatial/0.25.7-neo4j-3.4.9/neo4j-spatial-0.25.7-neo4j-3.4.9-server-plugin.jar
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.248.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.248.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 16135449 (15M) [application/octet-stream]
Saving to: 'neo4j-spatial-0.25.7-neo4j-3.4.9-server-plugin.jar?raw=true'

neo4j-spatial-0.25.7-neo4j-3. 100%[=====] 15.39M --.-KB/s   in 0.1s

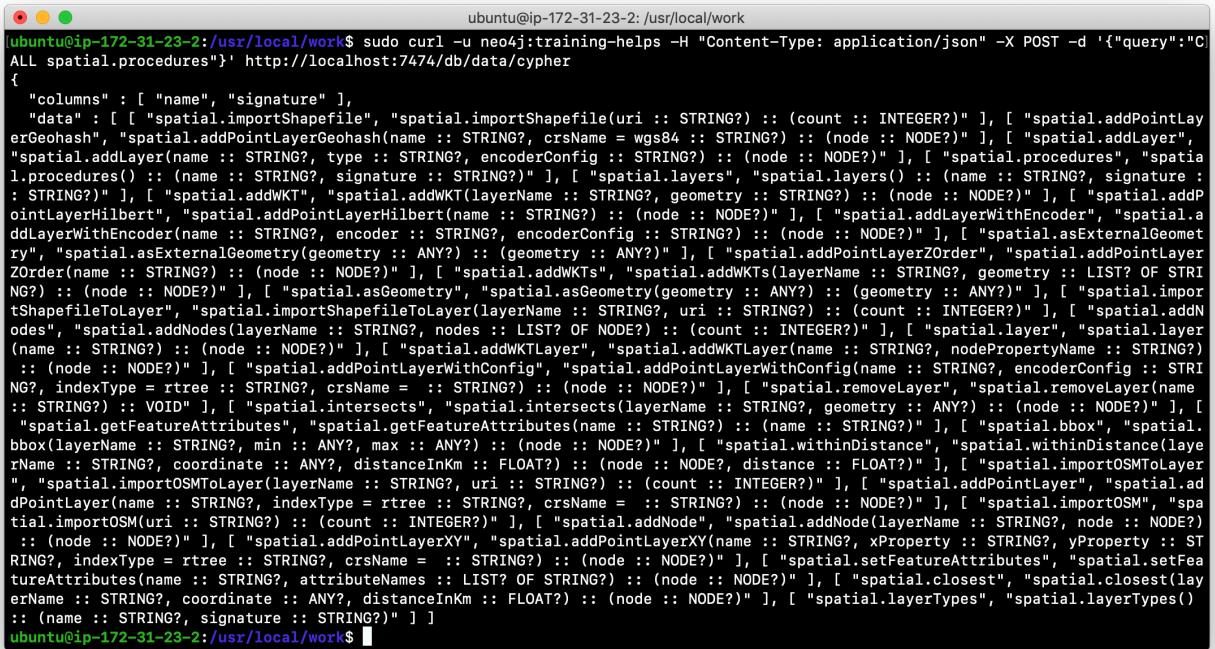
2019-01-02 18:14:45 (141 MB/s) - 'neo4j-spatial-0.25.7-neo4j-3.4.9-server-plugin.jar?raw=true' saved [16135449/16135449]

ubuntu@ip-172-31-23-2:~$ ls
README.txt          graph-algorithms-algo-3.5.0.1.jar
apoc-3.5.0.1-all.jar 'neo4j-spatial-0.25.7-neo4j-3.4.9-server-plugin.jar?raw=true'
ubuntu@ip-172-31-23-2:/var/lib/neo4j/plugins$ sudo mv 'neo4j-spatial-0.25.7-neo4j-3.4.9-server-plugin.jar?raw=true' neo4j-spatial-0.25.7-neo4j-3.4.9-server-plugin.jar
ubuntu@ip-172-31-23-2:/var/lib/neo4j/plugins$ sudo chown neo4j:neo4j *.jar
ubuntu@ip-172-31-23-2:/var/lib/neo4j/plugins$ sudo chmod +x *.jar
ubuntu@ip-172-31-23-2:/var/lib/neo4j/plugins$ ls -l
total 29508
-rw-r--r-- 1 neo4j adm      2217 Dec 18 07:14 README.txt
-rwxr-xr-x 1 neo4j neo4j 12880869 Nov 27 14:16 apoc-3.5.0.1-all.jar
-rwxr-xr-x 1 neo4j neo4j 1177328 Oct 12 06:40 graph-algorithms-algo-3.5.0.1.jar
-rwxr-xr-x 1 neo4j neo4j 16135449 Jan  2 18:14 neo4j-spatial-0.25.7-neo4j-3.4.9-server-plugin.jar
ubuntu@ip-172-31-23-2:/var/lib/neo4j/plugins$
```

Here is what you should see when you execute the first `curl` command:

```
ubuntu@ip-172-31-23-2:~$ curl -u neo4j:training-helps http://localhost:7474/db/data/
{
  "extensions" : {
    "SpatialPlugin" : {
      "addSimplePointLayer" : "http://localhost:7474/db/data/ext/SpatialPlugin/graphdb/addSimplePointLayer",
      "addNodesToLayer" : "http://localhost:7474/db/data/ext/SpatialPlugin/graphdb/addNodesToLayer",
      "findClosestGeometries" : "http://localhost:7474/db/data/ext/SpatialPlugin/graphdb/findClosestGeometries",
      "addGeometryWKTToLayer" : "http://localhost:7474/db/data/ext/SpatialPlugin/graphdb/addGeometryWKTToLayer",
      "addEditableLayer" : "http://localhost:7474/db/data/ext/SpatialPlugin/graphdb/addEditableLayer",
      "findGeometriesWithinDistance" : "http://localhost:7474/db/data/ext/SpatialPlugin/graphdb/findGeometriesWithinDistance",
      "addNodeToLayer" : "http://localhost:7474/db/data/ext/SpatialPlugin/graphdb/addNodeToLayer",
      "addCQLDynamicLayer" : "http://localhost:7474/db/data/ext/SpatialPlugin/graphdb/addCQLDynamicLayer",
      "getLayer" : "http://localhost:7474/db/data/ext/SpatialPlugin/graphdb/getLayer",
      "findGeometriesInBBox" : "http://localhost:7474/db/data/ext/SpatialPlugin/graphdb/findGeometriesInBBox",
      "findGeometriesIntersectingBBox" : "http://localhost:7474/db/data/ext/SpatialPlugin/graphdb/findGeometriesIntersectingBBox"
    }
  },
  "node" : "http://localhost:7474/db/data/node",
  "relationship" : "http://localhost:7474/db/data/relationship",
  "node_index" : "http://localhost:7474/db/data/index/node",
  "relationship_index" : "http://localhost:7474/db/data/index/relationship",
  "extensions_info" : "http://localhost:7474/db/data/ext",
  "relationship_types" : "http://localhost:7474/db/data/relationship/types",
  "batch" : "http://localhost:7474/db/data/batch",
  "cypher" : "http://localhost:7474/db/data/cypher",
  "indexes" : "http://localhost:7474/db/data/schema/index",
  "constraints" : "http://localhost:7474/db/data/schema/constraint",
  "transaction" : "http://localhost:7474/db/data/transaction",
  "node_labels" : "http://localhost:7474/db/data/labels",
  "neo4j_version" : "3.5.1"
}
```

Here is what you should see when you execute the second `curl` command:



```
ubuntu@ip-172-31-23-2:/usr/local/work$ sudo curl -u neo4j:training-helps -H "Content-Type: application/json" -X POST -d '{"query":"CALL spatial.procedures"}' http://localhost:7474/db/data/cypher
{
  "columns" : [ "name", "signature" ],
  "data" : [ [ "spatial.importShapefile", "spatial.importShapefile(uri :: STRING?) :: (count :: INTEGER?)"], [ "spatial.addPointLayerGeoHash", "spatial.addPointLayerGeoHash(name :: STRING?, crsName = wgs84 :: STRING?) :: (node :: NODE?)"], [ "spatial.addLayer", "spatial.addLayer(name :: STRING?, type :: STRING?, encoderConfig :: STRING?) :: (node :: NODE?)"], [ "spatial.procedures", "spatial.procedures() :: (name :: STRING?, signature :: STRING?)"], [ "spatial.layers", "spatial.layers() :: (name :: STRING?, signature :: STRING?)"], [ "spatial.addWKT", "spatial.addWKT(layerName :: STRING?, geometry :: STRING?) :: (node :: NODE?)"], [ "spatial.addPointLayerHilbert", "spatial.addPointLayerHilbert(name :: STRING?) :: (node :: NODE?)"], [ "spatial.addLayerWithEncoder", "spatial.addLayerWithEncoder(name :: STRING?, encoder :: STRING?, encoderConfig :: STRING?) :: (node :: NODE?)"], [ "spatial.asExternalGeometry", "spatial.asExternalGeometry(geometry :: ANY?) :: (geometry :: ANY?)"], [ "spatial.addPointLayerZOrder", "spatial.addPointLayerZOrder(name :: STRING?) :: (node :: NODE?)"], [ "spatial.addWKTs", "spatial.addWKTs(layerName :: STRING?, geometry :: LIST? OF STRING?) :: (node :: NODE?)"], [ "spatial.asGeometry", "spatial.asGeometry(geometry :: ANY?) :: (geometry :: ANY?)"], [ "spatial.importShapefileToLayer", "spatial.importShapefileToLayer(layerName :: STRING?, uri :: STRING?) :: (count :: INTEGER?)"], [ "spatial.addNode", "spatial.addNode(layerName :: STRING?, nodes :: LIST? OF NODE?) :: (count :: INTEGER?)"], [ "spatial.layer", "spatial.layer(name :: STRING?) :: (node :: NODE?)"], [ "spatial.addWKTLayer", "spatial.addWKTLayer(name :: STRING?, nodePropertyName :: STRING?) :: (node :: NODE?)"], [ "spatial.addPointLayerWithConfig", "spatial.addPointLayerWithConfig(name :: STRING?, encoderConfig :: STRING?, indexType = rtree :: STRING?, crsName = :: STRING?) :: (node :: NODE?)"], [ "spatial.removeLayer", "spatial.removeLayer(name :: STRING?) :: (VOID?)"], [ "spatial.intersects", "spatial.intersects(layerName :: STRING?, geometry :: ANY?) :: (node :: NODE?)"], [ "spatial.getFeatureAttributes", "spatial.getFeatureAttributes(name :: STRING?) :: (name :: STRING?)"], [ "spatial.bbox", "spatial.bbox(layerName :: STRING?, min :: ANY?, max :: ANY?) :: (node :: NODE?)"], [ "spatial.withinDistance", "spatial.withinDistance(layerName :: STRING?, coordinate :: ANY?, distanceInKm :: FLOAT?) :: (node :: NODE?, distance :: FLOAT?)"], [ "spatial.importOSMToLayer", "spatial.importOSMToLayer(layerName :: STRING?, uri :: STRING?) :: (count :: INTEGER?)"], [ "spatial.addPointLayer", "spatial.addPointLayer(name :: STRING?, indexType = rtree :: STRING?, crsName = :: STRING?) :: (node :: NODE?)"], [ "spatial.importOSM", "spatial.importOSM(uri :: STRING?) :: (count :: INTEGER?)"], [ "spatial.addNode", "spatial.addNode(layerName :: STRING?, node :: NODE?) :: (node :: NODE?)"], [ "spatial.addPointLayerXY", "spatial.addPointLayerXY(name :: STRING?, xProperty :: STRING?, yProperty :: STRING?) :: (node :: NODE?)"], [ "spatial.setFeatureAttributes", "spatial.setFeatureAttributes(name :: STRING?, attributeNames :: LIST? OF STRING?) :: (node :: NODE?)"], [ "spatial.closest", "spatial.closest(layerName :: STRING?, coordinate :: ANY?, distanceInKm :: FLOAT?) :: (node :: NODE?)"], [ "spatial.layerTypes", "spatial.layerTypes() :: (name :: STRING?, signature :: STRING?)"]
ubuntu@ip-172-31-23-2:/usr/local/work$
```

8. In the `/usr/local/work` folder, create a script named `ListProcedures.sh` that will write the list of procedures available to the Neo4j instance to the `/usr/local/work/Procedures.txt` file.

9. Run the **ListProcedures.sh** script and examine the contents to also verify that the plugin has been installed. The **Procedures.txt** file should contain these items:



The screenshot shows a terminal window with a black background and white text. At the top, it displays the session information: "ubuntu@ip-172-31-23-2: /usr/local/work". Below this, a large block of text lists various database procedures. The text is as follows:

```
"dbms.setTXMetaData"
"dbms.showCurrentUser"
"spatial.addLayer"
"spatial.addLayerWithEncoder"
"spatial.addNode"
"spatial.addNodes"
"spatial.addPointLayer"
"spatial.addPointLayerGeohash"
"spatial.addPointLayerHilbert"
"spatial.addPointLayerWithConfig"
"spatial.addPointLayerXY"
"spatial.addPointLayerZOrder"
"spatial.addWKT"
"spatial.addWKTLayer"
"spatial.addWKTs"
"spatial.asExternalGeometry"
"spatial.asGeometry"
"spatial.bbox"
"spatial.closest"
"spatial.getFeatureAttributes"
"spatial.importOSM"
"spatial.importOSMToLayer"
"spatial.importShapefile"
"spatial.importShapefileToLayer"
"spatial.intersects"
"spatial.layer"
"spatial.layerTypes"
"spatial.layers"
"spatial.procedures"
"spatial.removeLayer"
"spatial.setFeatureAttributes"
"spatial.withinDistance"
```

At the bottom right of the terminal window, there are two small status indicators: "458,19" and "Bot".

# Configuring connector ports for the Neo4j instance

The Neo4j instance uses [default port numbers](#) that may conflict with other processes on your system. The ports frequently used are the connector ports:

Name	Port Number	Description
HTTP	7474	Used by Neo4j Browser and REST API. It is <b>not</b> encrypted so it should never be exposed externally.
HTTPS	7473	Used by REST API. Requires additional SSL configuration.
Bolt	7687	Bolt connection used by Neo4j Browser, cypher-shell, and client applications.

## Modifying the default connector ports

If any of these ports conflict with ports already used on your system, you can change these connector ports by modifying these property values in the **neo4j.conf** file:

```
# Bolt connector
dbms.connector.bolt.enabled=true
#dbms.connector.bolt.tls_level=OPTIONAL
#dbms.connector.bolt.listen_address=:*7687*

# HTTP Connector. There can be zero or one HTTP connectors.
dbms.connector.http.enabled=true
#dbms.connector.http.listen_address=:*7474*

# HTTPS Connector. There can be zero or one HTTPS connectors.
dbms.connector.https.enabled=true
#dbms.connector.https.listen_address=:*7473*
```

As you learn more about some of the other administrative tasks for a Neo4j instance, you will work with other ports.

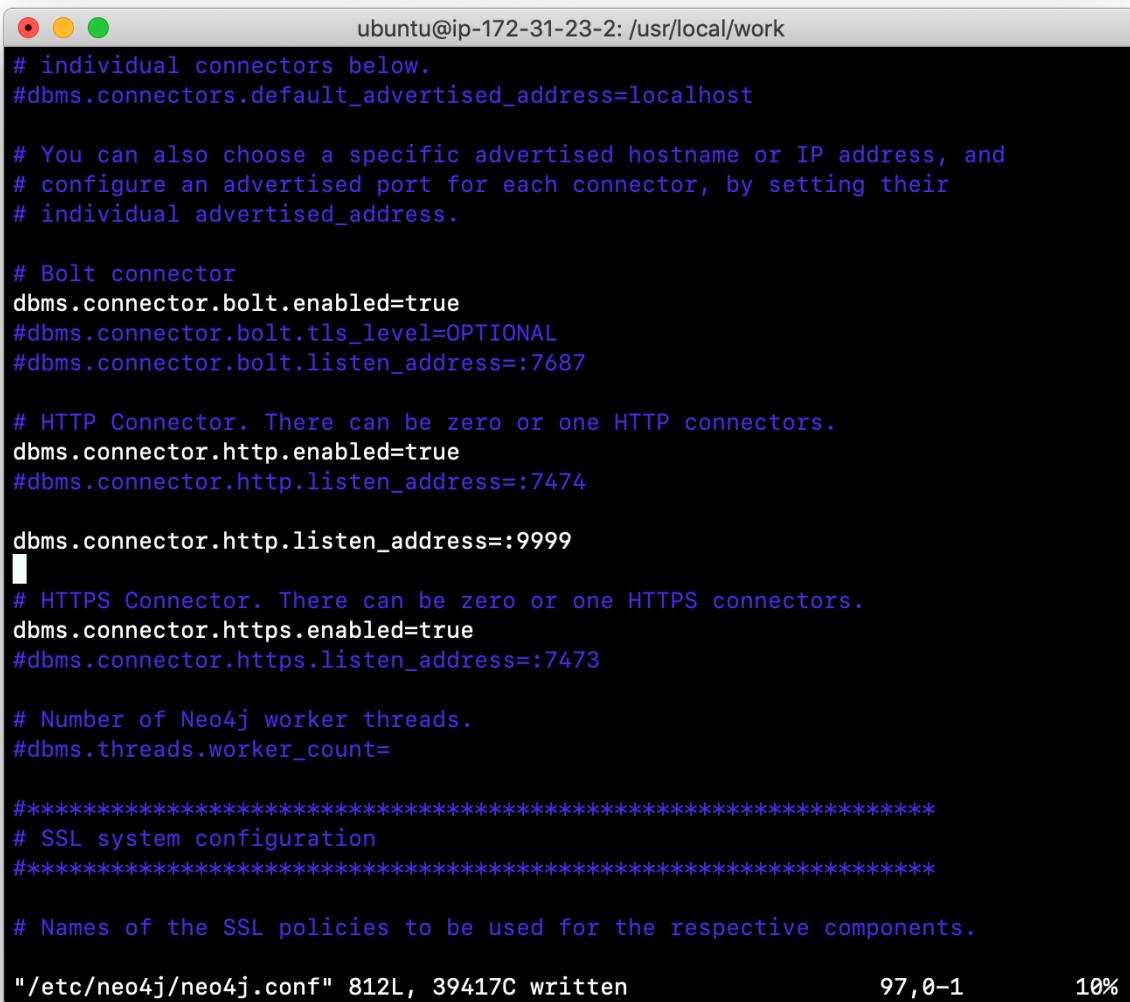
**NOTE** The REST API will be deprecated in Neo4j 4.0.

## Exercise #8: Modify the HTTP port

In this Exercise, you will modify the default HTTP port used by the HTTP instance and use the new port.

### Exercise steps:

1. Change the HTTP port to a value that is not in use on your system, for example **9999**. For example, your **neo4j.conf** file should look something like this:



```
ubuntu@ip-172-31-23-2: /usr/local/work
# individual connectors below.
#dbms.connectors.default_advertised_address=localhost

# You can also choose a specific advertised hostname or IP address, and
# configure an advertised port for each connector, by setting their
# individual advertised_address.

# Bolt connector
dbms.connector.bolt.enabled=true
#dbms.connector.bolt.tls_level=OPTIONAL
#dbms.connector.bolt.listen_address=:7687

# HTTP Connector. There can be zero or one HTTP connectors.
dbms.connector.http.enabled=true
#dbms.connector.http.listen_address=:7474

dbms.connector.http.listen_address=:9999
#
# HTTPS Connector. There can be zero or one HTTPS connectors.
dbms.connector.https.enabled=true
#dbms.connector.https.listen_address=:7473

# Number of Neo4j worker threads.
#dbms.threads.worker_count=

*****
# SSL system configuration
*****


# Names of the SSL policies to be used for the respective components.

"/etc/neo4j/neo4j.conf" 812L, 39417C written          97,0-1          10%
```

2. Restart the Neo4j instance.

3. Confirm that the port works by entering the following `curl` command that uses the Neo4j HTTP API to create a node, where it will ask you for the password the `neo4j` user:

```
curl -v -H "Content-Type: application/json" -d '{ "statements" : [ { "statement" : "CREATE (n) RETURN id(n)" }]}' http://localhost:9999/db/data/transaction/commit -u neo4j
```



A terminal window titled "Terminal" showing the execution of a curl command. The command is used to create a node in a Neo4j database via its REST API. It includes a password prompt for the 'neo4j' user and displays the response from the server, which includes headers like Content-Type and Access-Control-Allow-Origin.

```
ubuntu@ip-172-31-52-22:/usr/local/db/databases$ curl -v -H "Content-Type: application/json" -d '{ "statements" : [ { "statement" : "CREATE (n) RETURN id(n)" }]}' http://localhost:9999/db/data/transaction/commit -u neo4j
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 9999 (#0)
* Server auth using Basic with user 'neo4j'
> POST /db/data/transaction/commit HTTP/1.1
> Host: localhost:9999
> Authorization: Basic bmVvNGo6dHJhaW5pbmctaGVscHM=
> User-Agent: curl/7.58.0
> Accept: */*
> Content-type: application/json
> Content-Length: 65
>
* upload completely sent off: 65 out of 65 bytes
< HTTP/1.1 200 OK
< Date: Tue, 26 Feb 2019 19:51:39 GMT
< Access-Control-Allow-Origin: *
< Content-Type: application/json
< Content-Length: 82
<
* Connection #0 to host localhost left intact
ubuntu@ip-172-31-52-22:/usr/local/db/databases$ |
```

4. Change the HTTP port back to its default (7474).  
5. Restart the Neo4j instance.

# Performing online backup and restore

Online backup is used in production where the application cannot tolerate the database being unavailable. In this part of the training, you will learn how to back up and restore a stand-alone Neo4j database. Later in this training, you will learn about backup and restore in a Neo4j Causal Cluster environment.

## Enabling online backup

To enable a Neo4j instance to be backed up online, you must add these two properties to your **neo4j.conf** file:

```
dbms.backup.enabled=true  
dbms.backup.address=<host-address>:<6362-6372>
```

Where *host-address* is the address of a server from which you will run the backup tool from. You must specify a port number that will not conflict with existing ports used on the backup server.

A best practice for online backup of a stand-alone production database is to perform the backup on a different server. This is because the backup process and consistency checking is expensive and you want to offload this to another server.

A common practice for many enterprises is to back up their databases to Amazon S3 sites. In addition, if any backups are to be stored in S3, they should be encrypted as well as the channel used to create send the backup to S3.

## Performing the backup

After you restart the Neo4j instance, you can then perform the backup on the server you specified in *host\_address* as follows with consistency checking:

```
neo4j-admin backup --backup-dir=<backup-folder>  
                  --name=<backup-instance-folder-name>  
                  --from=<Neo4j-instance-host-address:<port>>  
                  --check-consistency=true  
                  --cc-report-dir=<report-directory>
```

This will perform a full backup to *backup-instance-folder-name* for the Neo4j instance running on *Neo4j-instance-host-address*.

# Restoring from a backup

If you need to restore a database from a backup, you must first stop the Neo4j instance. Since the instance is down, you can restore the database on the same server that runs the instance, provided the server has access to the backup location in the network.

Here is how you restore the database from a backup:

```
neo4j-admin restore  
  --from=<absolute-path-to-backup-instance-folder-name>  
  --database=<database-name>  
  --force=true
```

You specify *true* for force so that the existing database will be replaced.

**NOTE** If you restore a database as *root*, make sure that you change the ownership (recursively) of the database directory to *neo4j:neo4j* before starting the Neo4j instance.

There are many ways for performing online backups, including incremental backups. See the [Neo4j Operations Manual](#) for details.

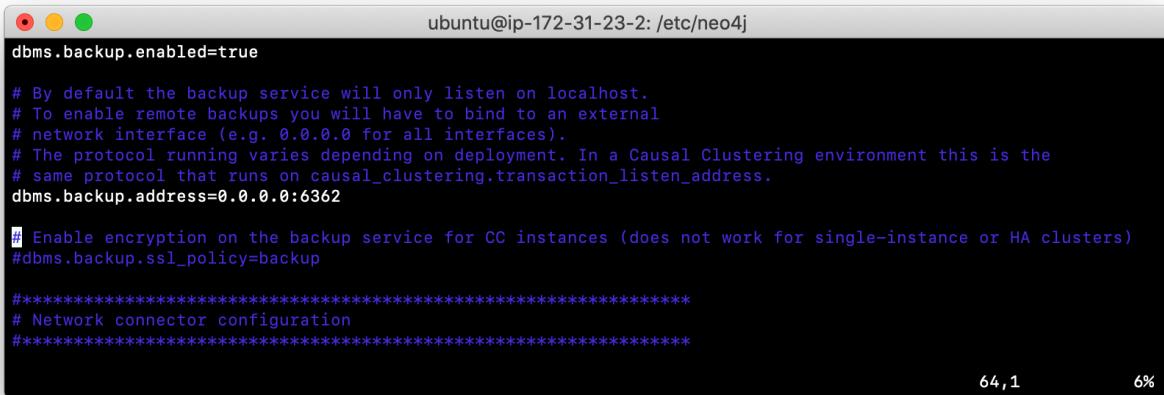
# Exercise #9: Performing online backup and restore

In this Exercise, you will perform an online backup of your database where you use the same host for the backup process. Then you will modify the database. Finally, you will restore the database from the backup.

**NOTE** In your real application, if you were to back up a production stand-alone Neo4j instance, you would use a different host from the host that is running the Neo4j instance.

## Exercise steps:

1. Stop the Neo4j instance.
2. Modify the Neo4j configuration so that online backup is enabled and will be done on this same host. For example, your **neo4j.conf** file should look something like this:



```
ubuntu@ip-172-31-23-2: /etc/neo4j
dbms.backup.enabled=true

# By default the backup service will only listen on localhost.
# To enable remote backups you will have to bind to an external
# network interface (e.g. 0.0.0.0 for all interfaces).
# The protocol running varies depending on deployment. In a Causal Clustering environment this is the
# same protocol that runs on causal_clustering.transaction_listen_address.
dbms.backup.address=0.0.0.0:6362

# Enable encryption on the backup service for CC instances (does not work for single-instance or HA clusters)
#dbms.backup.ssl_policy=backup

*****#
# Network connector configuration
*****#
```

3. Restart the Neo4j instance.
4. Create a folder named **/usr/local/backup** and ensure that it is owned by **neo4j:neo4j**.

5. Perform an online backup of the active database (**movie3.db**). The result of the backup should look something like this:

```
ubuntu@ip-172-31-52-22:~$ sudo /usr/bin/neo4j-admin backup --backup-dir=/usr/local/backups --name=movie3DB-backup --from=localhost:6362 --check-consistency=true --cc-report-dir=/usr/local/reports
2019-01-15 15:59:55.132+0000 INFO [o.n.b.i.BackupOutputMonitor] Start receiving store files
2019-01-15 15:59:55.132+0000 INFO [o.n.b.i.BackupOutputMonitor] Finish receiving store file /usr/local/backups/movie3DB-backup/temp-copy/neostore.nodesstore.db.labels
2019-01-15 15:59:55.332+0000 INFO [o.n.b.i.BackupOutputMonitor] Finish receiving store file /usr/local/backups/movie3DB-backup/temp-copy/neostore.nodesstore.db.labels
2019-01-15 15:59:55.335+0000 INFO [o.n.b.i.BackupOutputMonitor] Start receiving store file /usr/local/backups/movie3DB-backup/temp-copy/neostore.nodesstore.db
2019-01-15 15:59:55.335+0000 INFO [o.n.b.i.BackupOutputMonitor] Finish receiving store file /usr/local/backups/movie3DB-backup/temp-copy/neostore.nodesstore.db
2019-01-15 15:59:55.336+0000 INFO [o.n.b.i.BackupOutputMonitor] Start receiving store file /usr/local/backups/movie3DB-backup/temp-copy/neostore.propertystore.db.index.keys
2019-01-15 15:59:55.336+0000 INFO [o.n.b.i.BackupOutputMonitor] Finish receiving store file /usr/local/backups/movie3DB-backup/temp-copy/neostore.propertystore.db.index.keys
2019-01-15 15:59:55.337+0000 INFO [o.n.b.i.BackupOutputMonitor] Start receiving store file /usr/local/backups/movie3DB-backup/temp-copy/neostore.propertystore.db.index
2019-01-15 15:59:55.337+0000 INFO [o.n.b.i.BackupOutputMonitor] Finish receiving store file /usr/local/backups/movie3DB-backup/temp-copy/neostore.propertystore.db.index
2019-01-15 15:59:55.338+0000 INFO [o.n.b.i.BackupOutputMonitor] Start receiving store file /usr/local/backups/movie3DB-backup/temp-copy/neostore.propertystore.db.strings
2019-01-15 15:59:55.338+0000 INFO [o.n.b.i.BackupOutputMonitor] Finish receiving store file /usr/local/backups/movie3DB-backup/temp-copy/neostore.propertystore.db.strings
2019-01-15 15:59:55.340+0000 INFO [o.n.b.i.BackupOutputMonitor] Start receiving store file /usr/local/backups/movie3DB-backup/temp-copy/neostore.propertystore.db.arrays
2019-01-15 15:59:55.340+0000 INFO [o.n.b.i.BackupOutputMonitor] Finish receiving store file /usr/local/backups/movie3DB-backup/temp-copy/neostore.propertystore.db.arrays
2019-01-15 15:59:55.340+0000 INFO [o.n.b.i.BackupOutputMonitor] Start receiving store file /usr/local/backups/movie3DB-backup/temp-copy/neostore.relationshipstypestore.db
2019-01-15 15:59:55.340+0000 INFO [o.n.b.i.BackupOutputMonitor] Finish receiving store file /usr/local/backups/movie3DB-backup/temp-copy/neostore.relationshipstypestore.db
2019-01-15 15:59:55.341+0000 INFO [o.n.b.i.BackupOutputMonitor] Start receiving store file /usr/local/backups/movie3DB-backup/temp-copy/neostore.relationshipstypestore.db.names
2019-01-15 15:59:55.342+0000 INFO [o.n.b.i.BackupOutputMonitor] Start receiving store file /usr/local/backups/movie3DB-backup/temp-copy/neostore.relationshipstypestore.db.names
2019-01-15 15:59:55.343+0000 INFO [o.n.b.i.BackupOutputMonitor] Start receiving store file /usr/local/backups/movie3DB-backup/temp-copy/neostore.relationshipstypestore.db
2019-01-15 15:59:55.344+0000 INFO [o.n.b.i.BackupOutputMonitor] Start receiving store file /usr/local/backups/movie3DB-backup/temp-copy/neostore.relationshipstypestore.db
2019-01-15 15:59:55.344+0000 INFO [o.n.b.i.BackupOutputMonitor] Start receiving store file /usr/local/backups/movie3DB-backup/temp-copy/neostore.labelsokenstore.db.names
```

```
ubuntu@ip-172-31-52-22:~$ 2019-01-15 15:59:57.510+0000 INFO [o.n.b.i.BackupOutputMonitor] Finish recovering store
18%
20%
38%
46%
56%
66%
76%
86%
96%
100%
Checking node and relationship counts
18%
28%
38%
46%
56%
66%
76%
86%
96%
100%
Backup complete.
ubuntu@ip-172-31-52-22:~$
```

6. Stop the Neo4j instance.
7. Corrupt the database like you did earlier in this module. Modify the file **movie3.db/neostore.nodesstore.db** by adding some text to the file.
8. Run the consistency check tool on **movie3.db** using **neo4j-admin** specifying **/usr/local/work/reports** as the folder where the report will be written.

```
neo4j-admin check-consistency --database=movie3.db --report-dir=/usr/local/reports
--verbose=true
```

9. The consistency check tool should return that inconsistencies were found.

```
ubuntu@ip-172-31-52-22:~$ .....2019-01-15 16:01:44.107+0000 ERROR [o.n.c.ConsistencyCheckService] The node key entries in the store does not correspond with the expected number.
CountsEntry[NodeKey(): 3]
Inconsistent with: 5
2019-01-15 16:01:44.107+0000 ERROR [o.n.c.ConsistencyCheckService] The relationship key entries in the store does not correspond with the expected number.
CountsEntry[RelationshipKey()-->[]: 22]
Inconsistent with: 17
2019-01-15 16:01:44.108+0000 WARN [o.n.c.ConsistencyCheckService] Inconsistencies found: ConsistencySummaryStatistics{
    Number of errors: 689
    Number of warnings: 1
    Number of inconsistent NODE records: 5
    Number of inconsistent RELATIONSHIP records: 491
    Number of inconsistent LABEL_SCAN_DOCUMENT records: 173
    Number of inconsistent COUNTS records: 20
}
2019-01-15 16:01:44.113+0000 WARN [o.n.c.ConsistencyCheckService] See '/usr/local/reports/inconsistencies-2019-01-15.16.01.41.report' for a detailed consistency report.
command failed: Inconsistencies found. See '/usr/local/reports/inconsistencies-2019-01-15.16.01.41.report' for details.
ubuntu@ip-172-31-52-22:~$
```



# Using the import tool to create a database

The course, *Introduction to Neo4j*, teaches you how to import .csv data using `LOAD CSV` in Cypher. `LOAD CSV` works fine for datasets containing fewer than 10M nodes. For large datasets, it may also be possible to import the data with some of the APOC procedures.

Data import for a graph database is resource-intensive because it needs to pre-compute joins (relationships) between records (nodes). For large datasets, a best practice is to import the data using the `import` command of the `neo4j-admin` tool. This tool creates the database from a set of .csv files.

You can read details about using the import tool in the [Neo4j Operations Manual](#).

## Creating CSV files for the import

The format of the .csv files is important. For both nodes and relationships, header information must be associated with the data. Header information contains an ID to uniquely identify the record, optional node labels or relationship types, and names for the properties representing the imported data. A .csv can have a header row, or you can place the header information in a separate file.

In this training, you will use data that has been created for you that represents crimes.

### CSV files for nodes

Here is portion of the `beats.csv` file with embedded header information for loading nodes of type `Beat`:

```
:ID(Beat),id,:LABEL  
1132,1132,Beat  
0813,0813,Beat  
0513,0513,Beat
```

The `beats.csv` records represent data that will be loaded into a node with the label `Beat`. In this example the record ID is the same as the `id` property value that will be used to create the node in the graph.

Here is an example of the `crimes_header.csv` header file for loading nodes of type `Crime`:

```
:ID(Crime),id,:LABEL,date,description
```

The nodes loaded with `Crimes_header.csv` will have the label, `:LABEL`. In addition, the data in the associated `crimes.csv` file will have values for the ID of the record, and property values for `id`, `date`, and `description`.

And here is a portion of the associated **crimes.csv** file for loading nodes of type *Crime*:

```
8920441,8920441,Crime,12/07/2012 07:50:00 AM,AUTOMOBILE  
4730813,4730813,Crime,05/09/2006 08:20:00 AM,POCKET-PICKING  
7150780,7150780,Crime,09/28/2009 01:00:00 AM,CHILD ABANDONMENT  
4556970,4556970,Crime,12/16/2005 08:39:24 PM,POSS: CANNABIS 30GMS OR LESS  
9442492,9442492,Crime,12/28/2013 12:15:00 PM,OVER $500
```

In addition, this dataset includes information about the types of crimes. These nodes are created without a label for the node, but their ID, *PrimaryType* will be used to link them to *Crime* nodes. Here is a portion of the **primaryTypes.csv** file for loading these nodes:

```
:ID(PrimaryType),crimeType  
ARSON,ARSON  
OBSCENITY,OBSCENITY  
ROBBERY,ROBBERY  
THEFT,THEFT  
CRIM SEXUAL ASSAULT,CRIM SEXUAL ASSAULT  
BURGLARY,BURGLARY
```

## CSV files for relationships

.csv files for loading relationships contain a row for every relationship where the ID for the starting and ending node is specified, as well as the relationship type. If you do not specify the relationship in the file, then you must specify it in the arguments to the import tool.

Here is a portion of the **crimesBeats.csv** file that will be used to create the :*ON\_BEAT* relationships between *Crime* and *Beat* nodes:

```
:START_ID(Crime),:END_ID(Beat),:TYPE  
6978096,0911,ON_BEAT  
3170923,2511,ON_BEAT  
3073515,1012,ON_BEAT  
8157905,0113,ON_BEAT
```

Here is a portion of a portion of the **crimesPrimaryTypes.csv** file that will be used to create the relationships between the *Crime* nodes and the nodes that contain the *CrimeType* data:

```
:START_ID(Crime),:END_ID(PrimaryType)
5221115,NARCOTICS
4522835,DECEPTIVE PRACTICE
3432518,BATTERY
6439993,CRIMINAL TRESPASS
```

The relationship, *:TYPE* is not specified in this file so it will be specified in the arguments when you load the data from this file.

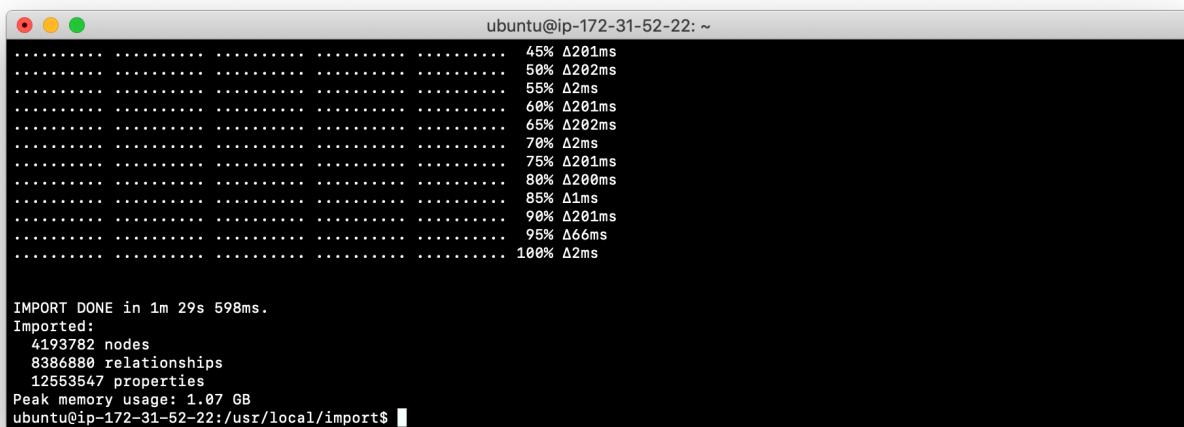
## Importing the data

After you have created or obtained the **.csv** files for the data, you import the data. The data import creates a database and you must run the import tool with the Neo4j instance stopped.

Here is the simplified syntax for creating a database from **.csv** files:

```
neo4j-admin import --database <database-name>
  --nodes[:Label1:Label2] [<rheader-csv-file-1>,<csv-file-1>
  --nodes[:Label1:Label2] [<rheader-csv-file-2>,<csv-file-2>
  --nodes[:Label1:Label2] [<rheader-csv-file-n>,<csv-file-n>
  --relationships[:REL_TYPE] [<jheader-csv-file-1>,<join-csv-file-
1>
  --relationships[:REL_TYPE] [<jheader-csv-file-2>,<join-csv-file-
2>
  --relationships[:REL_TYPE] [<jheader-csv-file-n>,<join-csv-file-
n>
  --report-file <report-file-path>
```

Here is the result of using the `import` command of `neo4j-admin` to create a database and import .csv files.



```
ubuntu@ip-172-31-52-22: ~
..... 45% Δ201ms
..... 50% Δ202ms
..... 55% Δ2ms
..... 60% Δ201ms
..... 65% Δ202ms
..... 70% Δ2ms
..... 75% Δ201ms
..... 80% Δ200ms
..... 85% Δ1ms
..... 90% Δ201ms
..... 95% Δ66ms
..... 100% Δ2ms

IMPORT DONE in 1m 29s 598ms.
Imported:
4193782 nodes
8386880 relationships
12553547 properties
Peak memory usage: 1.07 GB
ubuntu@ip-172-31-52-22:/usr/local/import$ █
```

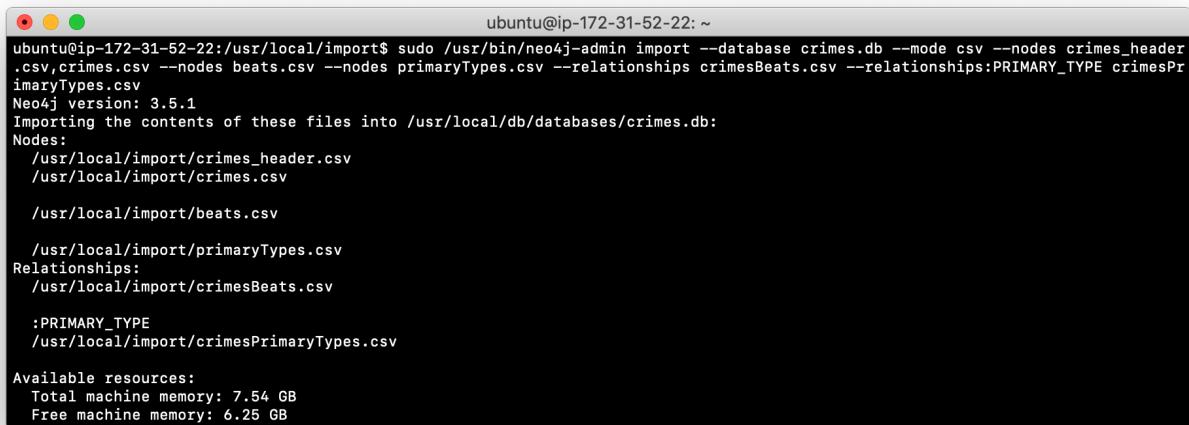
# Exercise #10: Importing data with the import command

In this Exercise, you create a new database by importing data using .csv files. Data import is very common when preparing a database for production where the data originally comes from relational tables.

## Exercise steps:

1. Stop the Neo4j instance.
2. In a terminal window, create the /usr/local/import folder.
3. Navigate to the **import** folder and download this file: <https://s3-us-west-1.amazonaws.com/data.neo4j.com/admin-neo4j/crime-data.zip>. Hint: use `curl -O` or `wget`.
4. Unzip the file. You should have six .csv files.
5. Examine the contents of the files to become familiar with their format and data.
6. Use the **import** command to import the data into a new database named **crimes.db**, using these guidelines:

```
--nodes crimes_header.csv,crimes.csv  
--nodes beats.csv  
--nodes primaryTypes.csv  
--relationships crimesBeats.csv  
--relationships:PRIMARY_TYPE crimesPrimaryTypes.csv
```



The screenshot shows a terminal window on an Ubuntu system. The command entered is:

```
ubuntu@ip-172-31-52-22:~$ sudo /usr/bin/neo4j-admin import --database crimes.db --mode csv --nodes crimes_header.csv,crimes.csv --nodes beats.csv --nodes primaryTypes.csv --relationships crimesBeats.csv --relationships:PRIMARY_TYPE crimesPrimaryTypes.csv
```

Output from the command:

```
Neo4j version: 3.5.1  
Importing the contents of these files into /usr/local/db/databases/crimes.db:  
Nodes:  
/usr/local/import/crimes_header.csv  
/usr/local/import/crimes.csv  
  
/usr/local/import/beats.csv  
  
/usr/local/import/primaryTypes.csv  
Relationships:  
/usr/local/import/crimesBeats.csv  
  
:PRIMARY_TYPE  
/usr/local/import/crimesPrimaryTypes.csv  
  
Available resources:  
Total machine memory: 7.54 GB  
Free machine memory: 6.25 GB
```

```
ubuntu@ip-172-31-52-22: ~
..... 45% Δ201ms
..... 50% Δ202ms
..... 55% Δ2ms
..... 60% Δ291ms
..... 65% Δ202ms
..... 70% Δ2ms
..... 75% Δ201ms
..... 80% Δ200ms
..... 85% Δ1ms
..... 90% Δ201ms
..... 95% Δ66ms
..... 100% Δ2ms

IMPORT DONE in 1m 29s 598ms.
Imported:
 4193782 nodes
 8386880 relationships
 12553547 properties
Peak memory usage: 1.07 GB
ubuntu@ip-172-31-52-22:/usr/local/import$
```

7. Modify the **neo4j.conf** file to use **crimes.db** as the active database.
8. Ensure that the ownership of the **crimes.db** directory and everything under it is owned by **neo4j:neo4j**.
9. Start the Neo4j instance.
10. Run **cyphe-shell** to retrieve the schema of the database and also count the number of *Crime* nodes in the graph.

```
ubuntu@ip-172-31-52-22: ~
[ubuntu@ip-172-31-52-22:/etc/neo4j$ /usr/bin/cypher-shell -u neo4j -p training-helps --format plain
[neo4j> CALL db.schema();
nodes, relationships
[{:Crime {name: "Crime", indexes: [], constraints: []}}, {:Beat {name: "Beat", indexes: [], constraints: []}}], [[:ON_BEAT]]
[neo4j> MATCH (c:Crime) RETURN count(c);
count(c)
4193440
[neo4j> :exit
ubuntu@ip-172-31-52-22:/etc/neo4j$
```

# Check your understanding

## Question 1

Suppose that you have installed Neo4j Enterprise Edition and have modified the name of the active database in the Neo4j configuration file. What tool and command do you run to create the new database?

Select the correct answer.

- `neo4j-admin create-database`
- `neo4j-admin initialize`
- `neo4j create-database`
- `neo4j start`

## Question 2

Suppose that you want the existing Neo4j database to have the name **ABCRecommendations.db**. Assuming that you have stopped the Neo4j instance, what steps must you perform to modify the name of the database, which currently has a default name of **graph.db**?

Select the correct answers.

- Rename the `NEO4J_HOME/graph.db` folder to `NEO4J_HOME/ABCRecommendations.db`.
- Modify `neo4j.conf` to use `dbms.active_database=ABCRecommendations.db`.
- Run `neo4j-admin rename graph.db ABCRecommendations.db`.
- Run `neo4j-admin move graph.db ABCRecommendations.db`.

## Question 3

How do you copy a database that you want to give to another user?

Select the correct answer.

- With the Neo4j instance started, run `neo4j-admin copy` providing the location where the copy will be created.
- With the Neo4j instance stopped, run `neo4j-admin copy` providing the location where the copy will be created.
- With the Neo4j instance started, run `neo4j-admin dump` providing the location where the dump file will be created.
- With the Neo4j instance stopped, run `neo4j-admin dump` providing the location where the dump file will be created.

# Summary

You should now be able to:

- Start a Neo4j instance.
- Stop the Neo4j instance.
- Set the password for the *neo4j* user.
- Copy a Neo4j database.
- Modify the location for a Neo4j database.
- Check the consistency of a Neo4j database.
- Create scripts for modifying a Neo4j database.
- Manage plugins for a Neo4j database.
- Configure ports used by the Neo4j instance.
- Perform an online backup of a Neo4j database.
- Create a database with the import tool.