1 Introduction to Neo4j

As with the Cassandra Lab, we are going to use Vagrant, Docker, and CoreOS to bootstrap our environment. The instructions are repeated below to ensure that the lab is self-contained. Those instructions that are repeated are marked as such, and those that differ are also marked.

The aim is that this process should work on your computers, as well as on the terminals in the CS labs that we cannot currently access. The instructions have been tested on a computer running macOS. (If lab access is restored in time, you can find a terminal in Lab E, and if it is running Linux, you can reboot the machine and choose macOS from the startup menu that will be presented to you).

We will use Vagrant to create a virtual machine (VM) that we will use for our experimentation. That VM will be running CoreOS, which is a cut-down version of Linux suitable for hosting "containers", using Docker. (CoreOS is soon at end-of-life in terms of support, but we are only using it as a compatibility layer, and it can be replaced by many other such layers.) We will use Docker to download and create instances of Neo4j, within the VM.

After you've cloned the correct git repo (see Get a CoreOS VM up and running), have run `vagrant up` and are connected, you can skip ahead to the section Getting Started with Neo4j to proceed with the lab work.

1.1 Setting up the Lab E Environment (same as for Cassandra lab)

Skip this section if you are using your own computer! Continue from "Ensure that Vagrant is working".

These steps only apply to working through this exercise in Lab E. We need to ensure that Vagrant places its (large) cache files somewhere other than your home directory. There are at least two ways to achieve this:

1. You can set the `VAGRANT_HOME` variable before *every* command. So if the instructions say to run `vagrant --version` you'd add `VAGRANT_HOME=/scratch/vagrant` before the command to turn it into `VAGRANT_HOME=/scratch/vagrant vagrant --version`.
2. You can set the `VAGRANT_HOME` variable to the value `/scratch/vagrant` inside your `.profile` (or other appropriate startup file) in the manner that you set up your Oracle variables in the DBA labs. You can test whether you have set the variable correctly by starting a new terminal window and running `echo $VAGRANT_HOME`, which should print back `/scratch/vagrant`.

Personally, I (David) prefer option 2, as almost all of you will be using bash by default. You can edit the startup file using `nano`:

```
nano ~/.profile
```

Add a line at the bottom of the file that reads `export VAGRANT_HOME=/scratch/vagrant`. Save the file (control-x, and follow the prompts), and then type `source ~/.profile` in the terminal window.

NOTE: you may find that `/scratch/vagrant` already exists (and that you don't have permission to create files inside it). In this case use your CS login name as a suffix. For example, `/scratch/vagrant-dme`.

1.2 Ensure that Vagrant is working (same as for Cassandra lab)

Check that Vagrant is installed. You can do so by running the following command from a terminal:

```
vagrant --version
```

If the above does not work (it complains about the command not being found), and you are on a computer that you administer, you can install Vagrant.

1.3 Get a CoreOS VM up and running (different from Cassandra lab)

NOTE: This step is slightly different to to the Cassandra lab's instructions. **The git repository URL is different!**

Gather the files needed to setup the lab. To do this, change directory to where you want a `neo4j-intro` subdirectory to be created and then run the command:

```
git clone https://altitude.otago.ac.nz/cosc430/neo4j-intro.git
```

(Note that this document is contained within the above git repository, and that you can browse the repository within a web browser by visiting the above URL.)

Now the latest version of the configuration files have been copied, `cd` into your git working copy directory, e.g. `cd neo4j-intro` and then copy the configuration file `config.rb.cosc430` into `config.rb`. The configuration here just changes VM to be created with 4 GiB of RAM, 2 virtual CPUs, and requests the stable version of CoreOS.

Get Vagrant to create the CoreOS VM using `vagrant up`. Note that the first time you run this command Vagrant will acquire the base image for Ubuntu, but it will only need to do that once.

1.4 Connect to your CoreOS VM (same as for Cassandra lab)

The terminal window that you are using will be connected to the terminal of your CoreOS VM if you run the command `vagrant ssh`.

Quick points to note:

- The CoreOS shell prompt is likely to be something similar to `core@core-01 ~ $`
- You can run `vagrant ssh` to open multiple SSH connections to your CoreOS VM in different terminal windows.

2 Getting Started with Neo4j

Once you have connected (via `vagrant ssh`) to the vagrant container, execute the following command to start Neo4j running.

```
docker run --publish=7474:7474 --publish=7687:7687 neo4j
```

In the terminal you should see something resembling the listing below (after any information displayed by Docker, if you are using the neo4j image for the first time).

```
Active database: graph.db
Directories in use:
  home:         /var/lib/neo4j
  config:       /var/lib/neo4j/conf
  logs:         /var/lib/neo4j/logs
  plugins:      /var/lib/neo4j/plugins
  import:       /var/lib/neo4j/import
  data:         /var/lib/neo4j/data
  certificates: /var/lib/neo4j/certificates
  run:          /var/lib/neo4j/run
Starting Neo4j.
2020-05-09 00:56:05.082+0000 INFO  ======== Neo4j 4.0.4 ========
2020-05-09 00:56:05.096+0000 INFO  Starting...
2020-05-09 00:56:13.668+0000 INFO  Bolt enabled on 0.0.0.0:7687.
2020-05-09 00:56:13.669+0000 INFO  Started.
2020-05-09 00:56:15.924+0000 INFO  Remote interface available at
... http://localhost:7474/
```

It is probably safe to ignore lines in the above that may appear that contain `WARN` for this lab—Neo4j

used to complain about a lack of configuration options in the context of this COSC430 lab. The important point is that the database has started and that the `Remote interface` is available.

Open that link in a browser and you should then be presented with a login page (the password is `neo4j`), it will ask you to change the password so choose something appropriate.

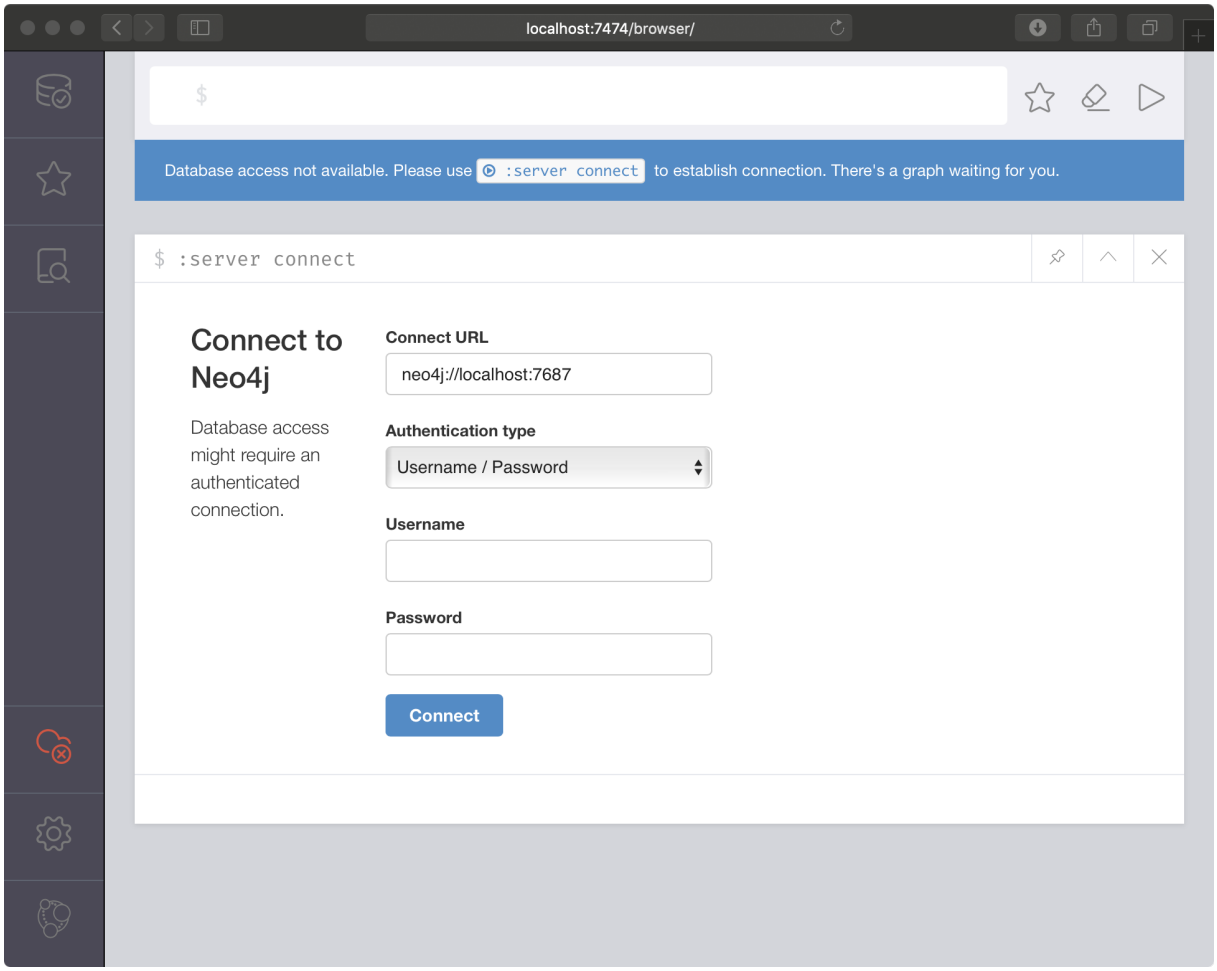

Figure 1: Neo4j Login

Once you've clicked through the password change prompt, you'll see a welcome screen.

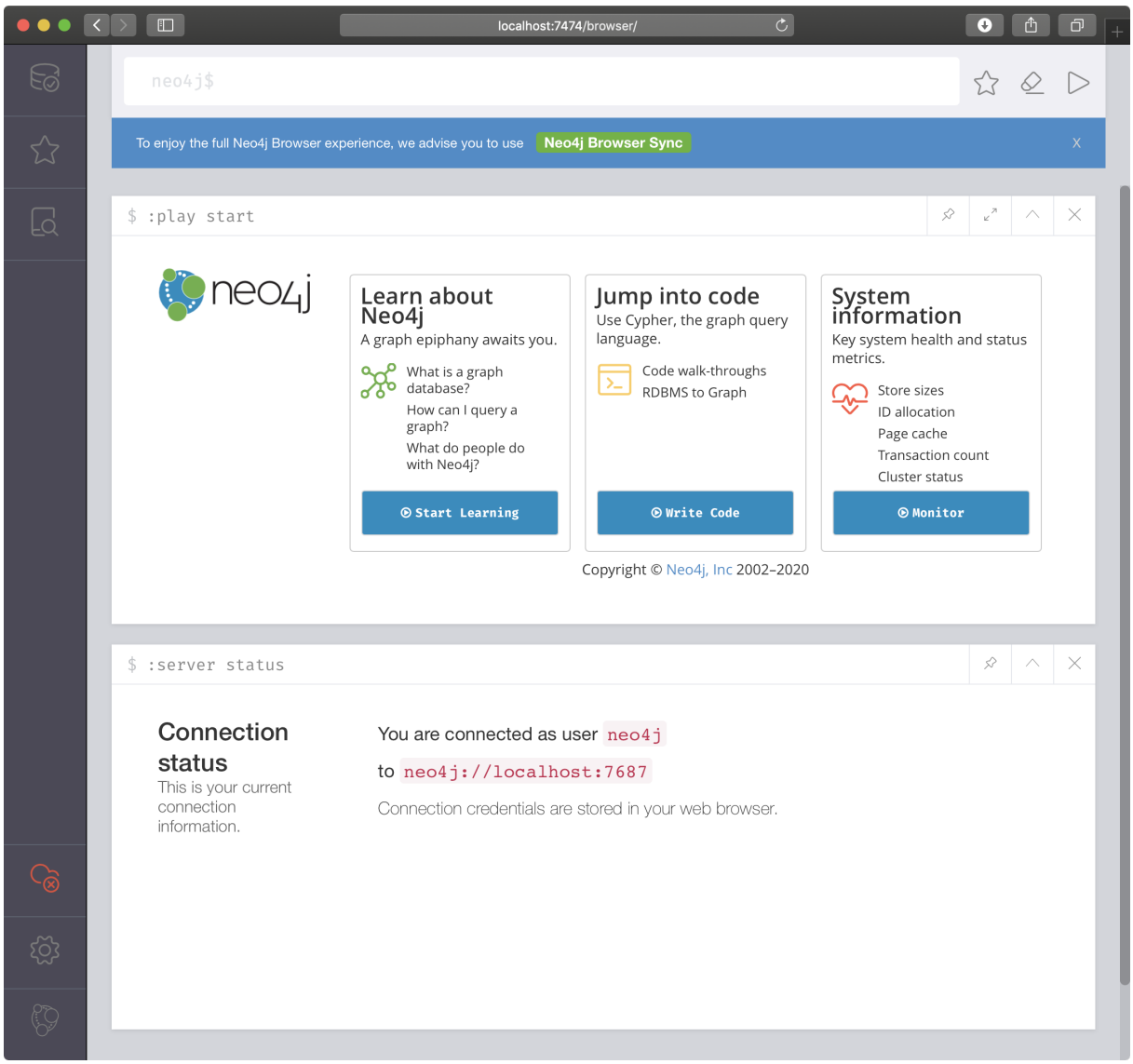

Figure 2: Neo4j Welcome Screen

In the main part of the page (the part with the light grey background), from top to bottom, we have the interactive query window, then some startup tutorials, and the final block tells us that we've successfully connected to the database backend (you can safely close this block by clicking on the cross in the top right of that block).

In the vertical bar on the left we have the following options (from top to bottom): Database Information (at present it will show that there are no nodes, no relationships, nor properties), Favourites (to assist with keeping track of queries), Documentation, the Cloud Services features (we won't need this), Browser Settings (controlling the appearance of the dashboard), and finally, About (describing the

software). For now, leave everything at the default values.

When working through the material below, you are advised to keep a record of commands that you have run so that you can easily recreate any databases or queries in future. For example, I (David) had my virtual machine automatically shutdown without warning at one point. I just needed to `vagrant up` to start the CoreOS system again, and then through `vagrant ssh` run the necessary Docker commands, however any data that I might have saved into the Neo4j would have been lost.

2.1 Creating the Database and Introduction to Cypher

Click on the `Write Code` button in the `Jump into code box` (the centre one). Create the Movie Graph by following the prompts. This will create a sample database for you. I've copied a (small) portion of the example commands that are creating the database for you.

```
CREATE (TheMatrix:Movie {title:'The Matrix', released:1999,
    tagline:'Welcome to the Real World'})
CREATE (Keanu:Person {name:'Keanu Reeves', born:1964})
...
CREATE (Keanu)-[:ACTED_IN {roles:['Neo']}]->(TheMatrix)
...
CREATE (TheMatrixReloaded:Movie {title:'The Matrix Reloaded',
    released:2003, tagline:'Free your mind'})
CREATE (Keanu)-[:ACTED_IN {roles:['Neo']}]->(TheMatrixReloaded)
...
```

This is the Cypher query language and may look somewhat similar to SQL. The above extract defines some properties of the nodes. Note that you don't *have to* define properties of nodes or links. In this case the visual nature of cypher becomes more apparent (however, less useful):

```
CREATE (Keanu:Person)
CREATE (TheMatrix:Movie)
CREATE (Keanu)-->(TheMatrix)
```

You may notice that the Cypher query language is almost visual in that the nodes are represented inside parentheses (thus making them look close to the nodes in the example graphs shown in COSC430 lectures) and the edges look like arrows (resembling the directed edges).

There are two styles of create statements in the first example above. The first, `CREATE (name:type {key:value, ... })`, creates nodes of the graph. In this extract we are creating two different types of node, a `Person` and a `Movie`. The person node is called 'Keanu' with the given properties. There are two `Movie` nodes being created `TheMatrix` and `TheMatrixReloaded`, again each with

their own properties.

The second style, `CREATE (from)-[:link_type {key:value ...}]->(to)`, creates a directed edge between the 'from' node to the 'to' node with the given link type and the properties. In this example, we're saying that the node named 'Keanu' is linked to the node named 'TheMatrix' by the relationship 'ACTED_IN' (and has the given property).

NOTE: The names of the nodes are *not* related to the properties of the node. For example, these statements have the same effect on the data stored in the database. The last line has no name, and therefore cannot be referenced in subsequent queries.

```
CREATE (Keanu:Person {name:'Keanu Reeves', born:1964})
CREATE (K:Person {name:'Keanu Reeves', born:1964})
CREATE (k:Person {name:'Keanu Reeves', born:1964})
CREATE (Person {name:'Keanu Reeves', born:1964})
```

Continue with the rest of the steps in the tutorial. By the end you should have completed all these steps:

- Movie Graph
- Create
- Find
- Query
- Solve
- Recommend
- Clean up

You'll notice that every query you run, a new box will appear with the results. You can close each such box by clicking on the cross in the top right of the box. To aid in visualising the results of your queries, you can make the box occupy the entire web browser window as well as being able to drag and drop the nodes, and zoom in and out—these controls are in the bottom right of the box. Other visualisations are possible—a table, plain text, and code (using a JSON representation).

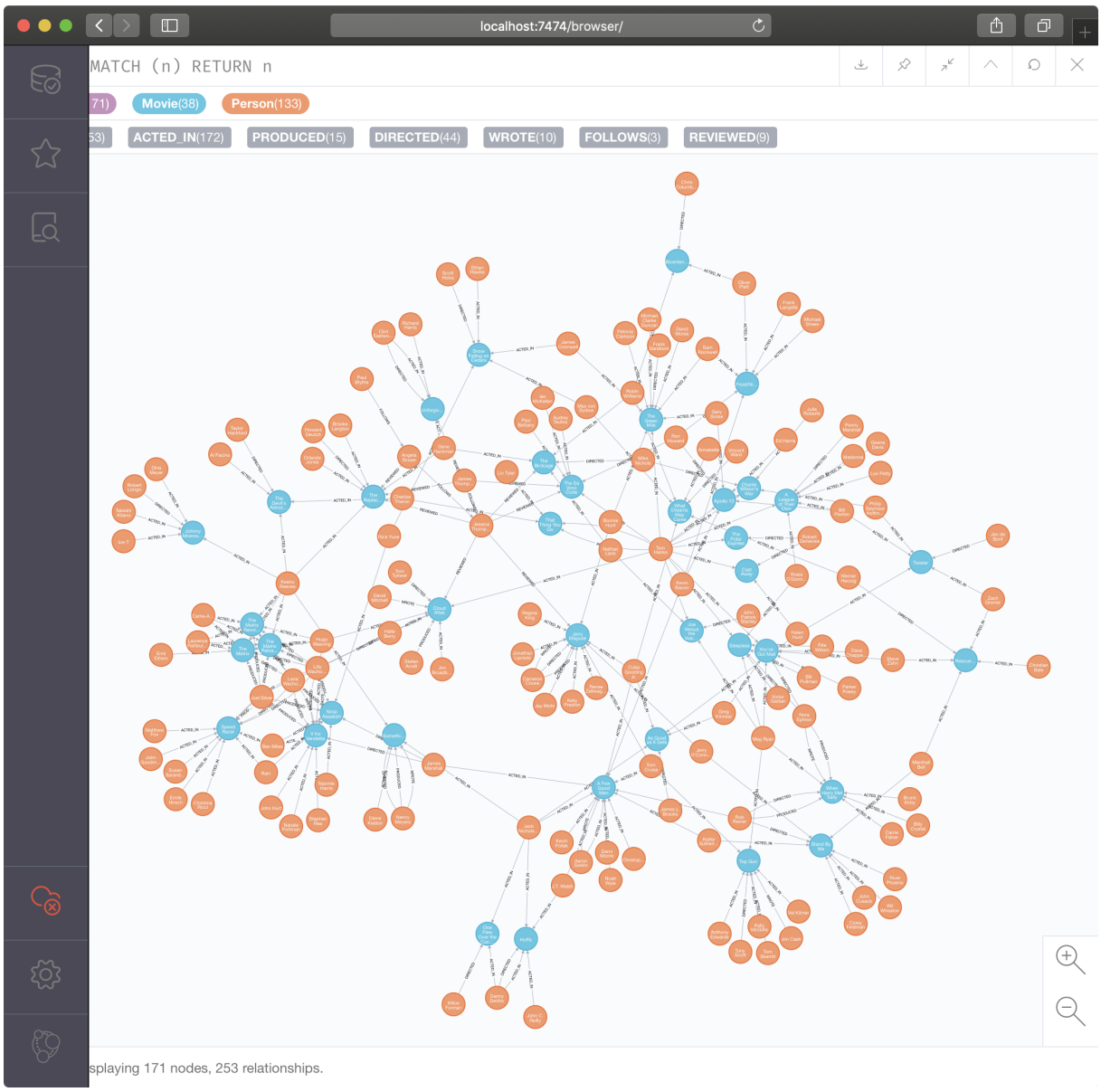

Figure 3: Neo4j Movie Database

2.2 Finding and Querying

The next step is to query the data. Follow the examples given and write down the queries you used for the examples below:

1. Find the actor named 'Keanu Reeves' (hint: look up the properties of the node).
2. Find the movie with the title 'The Matrix'.
3. Find movies released in 1999.

4. List all the Keanu Reeves movies.
5. Who directed 'The Matrix'?
6. What other movies did they direct?
7. Who else has Keanu starred with?
8. The movies three 'hops' away from Keanu Reeves.
9. Extension Queries
    1. Find all actors that directed a movie they also acted in (showing both the movie and the actor)
    2. Find all reviewer pairs, one following the other and both reviewing the same movie, and return entire sub-graphs
    3. Find all actors that acted in a movie together after 2010 and return the actor names and movie node
    4. By extending the previous query, find all movies that the cast of the movies found before also acted in
10. Counting
    1. Count the number of paths of at most length 4 starting from Clint Eastwood ignoring edge direction
    2. Count the number of paths of at most length 10 starting from Clint Eastwood ignoring edge direction
    3. Count the number of paths of at most length 11 starting from Clint Eastwood ignoring edge direction
    4. Count the number of nodes reachable in at most 4 hops from Clint Eastwood ignoring edge direction
    5. Count the number of nodes reachable in at most 10 hops from Clint Eastwood ignoring edge direction
    6. Count the number of nodes reachable in at most 11 hops from Clint Eastwood ignoring edge direction

2.3 Modelling

Your task for this section is to implement the social network example presented during the lecture. You should also attempt to develop the queries also presented in the lecture.

3 Cleaning up the software downloaded during this lab

The simplest way to remove all of your containers is to destroy the CoreOS virtual machine that is hosting them. From the directory that you ran `vagrant ssh` from, you can run the command that

follows. It will ask for your confirmation, as it irrevocably throws away your virtual machine, and in this case all of the Neo4j containers and images, the caches from Docker, etc.

```
vagrant destroy
```

If you also want to remove all of the caches that Vagrant creates, you can examine the VAGRANT_HOME environmental variable (`echo $VAGRANT_HOME`) to discover the folder that you should delete. This folder will contain the cached image for CoreOS as well as the virtual machine hard-disk file onto which you downloaded Neo4j, using Docker.

4 Useful websites and reference documentation

- Neo4j
  - Neo4j Website https://neo4j.com/
  - Cypher Reference Documentation https://neo4j.com/docs/developer-manual/3.2/cypher/
  - Cypher Reference Card https://neo4j.com/docs/cypher-refcard/3.2/
- A Neo4j tutorial https://neo4j.com/developer/get-started/
- Documentation that you probably do not need, but which may be of interest:
  - Docker documentation: https://docs.docker.com
  - CoreOS http://coreos.com
  - Vagrant https://www.vagrantup.com

If you are interested in exploring other database systems, have a look at the Docker Hub https://hub.docker.com and you'll find there are a lot of pre-made images covering a wide variety of database systems (you don't need an account to download these containers). In many cases the Docker Hub pages include instructions on how to get started using each such database.