Homework 2: Graph Databases

E-63 Big Data Analytics
Harvard University, Autumn 2017

Tim Hagmann

September 16, 2017

Contents

Introduction

The following assignement is concerned with the application of Neo4J, a big data capable graph database. A graph database is a database that uses graph structures for semantic queries with nodes, edges and properties to represent and store data. A key concept of the system is the graph (or edge or relationship), which directly relates data items in the store. The relationships allow data in the store to be linked together directly, and in many cases retrieved with one operation. This contrasts with relational databases that permit managing the data in its natural structure (without imposing implementation aspects like physical record chains).

Neo4j is available in a GPL3-licensed open-source "community edition", with online backup and high availability extensions licensed under the terms of the Affero General Public License. Neo4j is implemented in Java and accessible from software written in other languages such as Python or R.

Setup

Before beeing able to work with the Neo4J database, the correct work environment has to be setup. You can find the setup instructions for Neo4J in the appendix of this document. Those instructions have to be followed in order to reproduce the following results.

Initialize

This paper is beeing written and executed in a R environment. In order to set up that environment, the following R code has to be executed. This loads all the necessary packages and functions for the following steps.

```r
## Options
options(scipen = 10)                            # Disable scientific notation
update_package <- FALSE                         # Use old status of packages

## Init files (always execute, eta: 10s)
source("scripts/01_init.R")                     # Helper functions to load packages
source("scripts/02_packages.R")                 # Load all necessary packages
source("scripts/03_functions.R")                # Load project specific functions
```

Password

The first steps after having installed a default Neo4J environment is to change the default passwort. We're changing it to *neo4jneo4j*.

```
curl -H "Content-Type: application/json" -XPOST -d '{"password":"neo4jneo4j"}' -u \
neo4j:neo4j http://34.251.36.174:7474/user/neo4j/password
```

```
##   % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
##                                  Dload  Upload   Total   Spent    Left  Speed
##
```

```
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100    25    0     0  100    25      0   5953 --:--:-- --:--:-- --:--:--  6250
```

Problem 1 (20%)

The first problem is concerned with putting the following movie content (i.e., cypher statement) into the Neo4J database using curl.

```
CREATE (matrix1:Movie { title : 'The Matrix', year : '1999-03-31' }) return id(matrix1)
CREATE (matrix2:Movie { title : 'The Matrix Reloaded', year : '2003-05-07' }) return id(matrix2)
CREATE (matrix3:Movie { title : 'The Matrix Revolutions', year : '2003-10-27' }) return id(matrix3)
CREATE (keanu:Actor { name:'Keanu Reeves' }) return id(keanu)
CREATE (laurence:Actor { name:'Laurence Fishburne' })
CREATE (carrieanne:Actor { name:'Carrie-Anne Moss' })
CREATE (keanu)-[:ACTS_IN { role : 'Neo' }]->(matrix1)
CREATE (keanu)-[:ACTS_IN { role : 'Neo' }]->(matrix2)
CREATE (keanu)-[:ACTS_IN { role : 'Neo' }]->(matrix3)
CREATE (laurence)-[:ACTS_IN { role : 'Morpheus' }]->(matrix1)
CREATE (laurence)-[:ACTS_IN { role : 'Morpheus' }]->(matrix2)
CREATE (laurence)-[:ACTS_IN { role : 'Morpheus' }]->(matrix3)
CREATE (carrieanne)-[:ACTS_IN { role : 'Trinity' }]->(matrix1)
CREATE (carrieanne)-[:ACTS_IN { role : 'Trinity' }]->(matrix2)
CREATE (carrieanne)-[:ACTS_IN { role : 'Trinity' }]->(matrix3)
```

JSON

In order to import the above data into the neo4j database using curl, the data can be formated into a JSON document. This allows us to input all the data in one curl statement. The necessary JSON file can be found in the *data/cypher_node1.json* document and is also shown below:

```
{
    "statements": [
        {
            "statement": "CREATE (matrix1:Movie { title : 'The Matrix',
                          year : '1999-03-31' }), (matrix2:Movie { title :
                          'The Matrix Reloaded', year : '2003-05-07' }),
                          (matrix3:Movie { title : 'The Matrix Revolutions',
                          year : '2003-10-27' }), (keanu:Actor {
                          name:'Keanu Reeves' }), (laurence:Actor { name:
                          'Laurence Fishburne' }), (carrieanne:Actor { name:
                          'Carrie-Anne Moss' }), (keanu)-[:ACTS_IN { role :
                          'Neo' }]->(matrix1), (keanu)-[:ACTS_IN { role : 'Neo'
                          }]->(matrix2), (keanu)-[:ACTS_IN { role : 'Neo' }]->
                          (matrix3), (laurence)-[:ACTS_IN { role : 'Morpheus'
                          }]->(matrix1), (laurence)-[:ACTS_IN { role : 'Morpheus'
                          }]->(matrix2), (laurence)-[:ACTS_IN { role : 'Morpheus'
                          }]->(matrix3), (carrieanne)-[:ACTS_IN { role : 'Trinity'
                          }]->(matrix1), (carrieanne)-[:ACTS_IN { role : 'Trinity'
                          }]->(matrix2), (carrieanne)-[:ACTS_IN { role : 'Trinity'
                          }]->(matrix3)"
        }
]
}
```

CURL

Once the cypher statements are in the *cypher_node1.json* file, the following curl command can be executed:

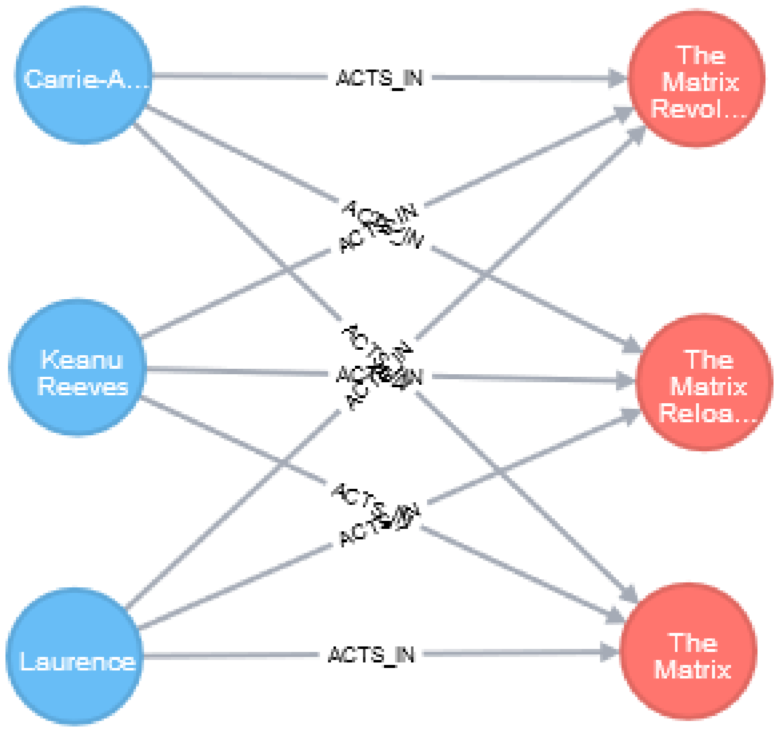
```
curl -i -H accept:application/json -H content-type:application/json -XPOST \
http://neo4j:neo4jneo4j@34.251.36.174:7474/db/data/transaction/commit --data \
@data/cypher_node1.json
```

Visualization

Having executed the above curl we can visualize the graph database in the neo4j browser with the following command:

```
MATCH (n1)-[r]->(n2) RETURN r, n1, n2
```

This leads to the following image:



The above output shows, that the 3 actors and movies have been created with the appropriate “Acted_In” relationships.

Problem 2 (15%)

Keanu Reeves acted in the movie “John Wick” which is not in the database. That movie was directed by Chad Stahelski and David Leitch. Cast of the movie included William Dafoe and Michael Nyquist.

Cypher

In order to insert the above relationship into the graph database, the following cypher statements can be run:

```
CREATE (johnwick:Movie { title : 'John Wick', year : '2014-10-24' }) return id(johnwick)
CREATE (chad:Director { name:'Chad Stahelski' })
CREATE (david:Director { name:'David Leitch' })
CREATE (william:Actor { name:'William Dafoe' })
CREATE (michael:Actor { name:'Michael Nyquist' })
CREATE (william)-[:ACTS_IN]->(johnwick)
CREATE (michael)-[:ACTS_IN]->(johnwick)
CREATE (chad)-[:DIRECTS]->(johnwick)
CREATE (david)-[:DIRECTS]->(johnwick)
MATCH (n {title:'John Wick'}), (m {name:'Keanu Reeves'}) CREATE (n)-[:ACTS_IN]->(m)
```

JSON

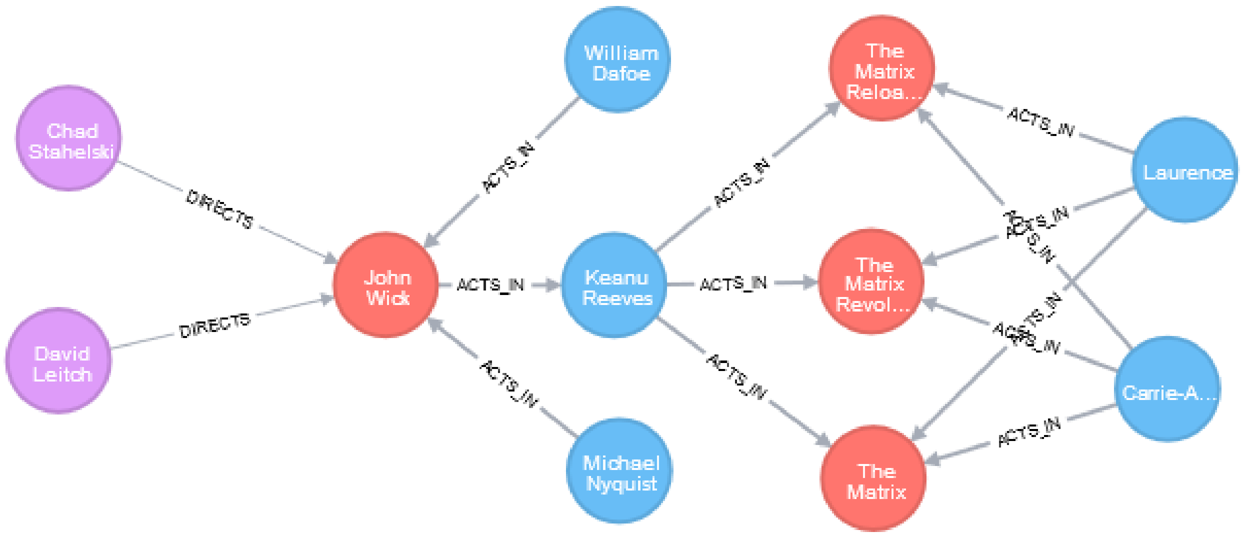
We're again translating those statements into json *(cypher_node2.json)* and are running them trough curl:

```
curl -i -H accept:application/json -H content-type:application/json -XPOST \
http://neo4j:neo4jneo4j@34.251.36.174:7474/db/data/transaction/commit --data \
@data/cypher_node2.json
```

Visualization

Demonstrate that you have successfully brought data about John Wick movie into the database. You can use Cypher Browser or any other means.

To validate the result, the following image can be brought up using the Neo4j browser.



Cleanup

Delete above movie and all the cast except Keanu Reeves.

Deleting the movie can be done with a *MATCH* and *DETACH DELETE* statement. An example of such a delete statment is show below:

```
MATCH (n { name: 'Chad Stahelski' }) DETACH DELETE n
```

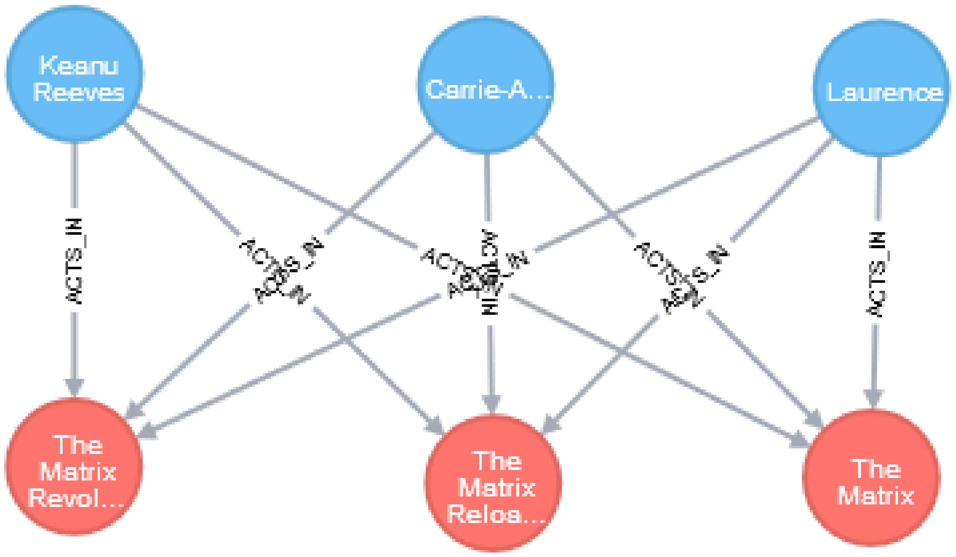
CURL

We're deleting the above movie in one go. In order to run a series of detach statments we're again running a CURL/JSON *(cypher_node3.json)* statement:

```
curl -i -H accept:application/json -H content-type:application/json -XPOST \
http://neo4j:neo4jneo4j@34.251.36.174:7474/db/data/transaction/commit --data \
@data/cypher_node3.json
```

Visualization

To validate the result, the following image can be brought up using the Neo4j browser.



Problem 3 (15%)

Add all the actors and the roles they played in this movie "John Wick" to the database using JAVA REST API or some other APIs for Neo4J in a language of your choice (not Curl). Demonstrate that you have successfully brought data about John Wick movie into the database. You can use Cypher Browser or any other means.

RNeo4j

In order to solve problem 3, the *RNeo4j* package (API) is beeing used. the package can be loaded trough the *library(RNeo4j)* command. However, we already loaded the package in the initialize portion at the begining of this document. Having loaded the *RNeo4j* package the next part is to build a connection to the database. This is beeing done with the following command:

```r
graph <- startGraph(url="http://34.251.36.174:7474/db/data/",
                    username="neo4j", password="neo4jneo4j")
```

Add data

Next we're adding the data into the database with the *Create Node* and *CreateRel* commands.

```r
# Add movie
john_wick <- createNode(graph, "Movie", title="John Wick",
                       year="2014-10-24")

# Add directors
chad_stahelski <- createNode(graph, "Director", name="Chad Stahelski")
david_leitch <- createNode(graph, "Director", name="David Leitch")

# Add actors
william_dafoe <- createNode(graph, "Actor", name="William Dafoe")
michael_nyquist <- createNode(graph, "Actor", name="Michael Nyquist")

# Add relationships
createRel(william_dafoe, "ACTS_IN", john_wick)
createRel(michael_nyquist, "ACTS_IN", john_wick)
createRel(chad_stahelski, "DIRECTS", john_wick)
createRel(david_leitch, "DIRECTS", john_wick)
```

Having created the above nodes and relationships we also have to create a relationsship between the actor *Keanu Reeves* and the movie *John Wick*. This can be done by selecting the the actor with a cypher command and then creating a relationship between the two:

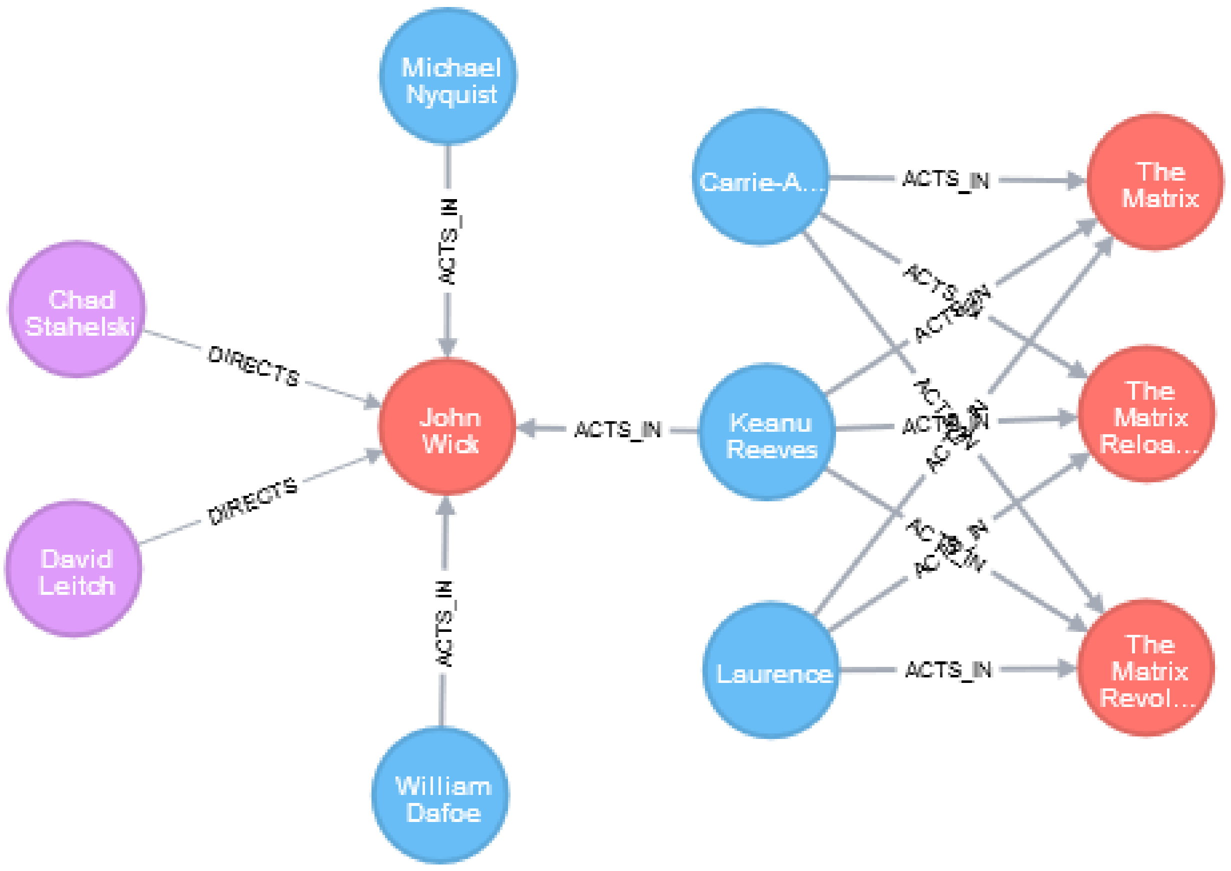
```r
# Get node
query <- "MATCH (a:Actor) WHERE a.name = 'Keanu Reeves' return a"
keanu_reeves <- getNodes(graph, query)[[1]]

# Add relationship
createRel(keanu_reeves, "ACTS_IN", john_wick)
```

```
## < Relationship >
## ACTS_IN
```

```r
# Remove objects (Cleanup)
rm(john_wick, chad_stahelski, david_leitch, william_dafoe, michael_nyquist,
   keanu_reeves)
```

Visualization

To validate the result, the following image can be brought up using the Neo4j browser.



Problem 4 (15%)

Find a list of actors playing in movies in which Keanu Reeves played. Find directors of movies in which K. Reeves played. Verify your results using Cypher queries in Cypher Browser.

Path Query

The first step to find the movies Keanu Reves played in is to run the following *MATCH* cypher query:

```
query <- "MATCH p = (:Actor {name:'Keanu Reeves'})-[:ACTS_IN]->(:Movie)<-[:ACTS_IN]-(:Actor) RETURN p"
paths <- getPaths(graph, query)
```

Search object

Following that we can loop trough the paths object in order to find his co-actors:

```
# Creaty empty array
co_actors <- NULL

# Loop trough arrays
for (i in 1:length(paths)){
  co_actors <- c(co_actors, endNode(paths[[i]]))
}
```

```
# Select unique names
co_actors <- unique(as.character(co_actors))
```

Output Table

Having found the unique actors they can be outputed in the following table:

```
pander(co_actors)
```

Michael Nyquist, *William Dafoe*, *Laurence Fishburne* and *Carrie-Anne Moss*

As can be seen above, there are four co-actors, that played in movies that Keanue Reeves also played in.

Problem 5 (20%)

Find a way to export data from Neo4j into a set of CSV files.

Export to CSV

In order to export the data into tabular CSV file an efficient way is to select all the nodes and relationsships seperatly and write them into CSV's. That means, that the following R code is creating separate CSV's for movies, actors, directors, and two types of relationships (ACTS_IN, DIRECTS).

```
# Export Nodes
write_csv(cypher(graph, "MATCH (m:Movie) return m.title as title, m.year as year"),
          "data/export_movies.csv")
write_csv(cypher(graph, "MATCH (a:Actor) return a.name as name"),
          "data/export_actors.csv")
write_csv(cypher(graph, "MATCH (a:Director) return a.name as name"),
          "data/export_directors.csv")

# Export Relationships
write_csv(cypher(graph, "MATCH (a:Actor)-[r:ACTS_IN]->(m:Movie) return a.name as name, m.title as title
          "data/export_acts_in.csv")
write_csv(cypher(graph, "MATCH (d:Director)-[r:DIRECTS]->(m:Movie) return d.name as name, m.title as ti
          "data/export_directs_in.csv")
```

Delete database

Delete your database and demonstrate that you can recreate the database by loading those CSV files.

The following R statment deletes the whole graph database on the Neo4J server:

```
clear(graph, input=FALSE)
```

Change security settings

Important Note: In order to make Neo4J able to read directly from disk, some of the security settings have to be changed. This can be done in the settings file, i.e., trough commenting out the dbms.directories.import line:

```
sudo nano /etc/neo4j/neo4j.conf
#dbms.directories.import
```

Security wise, this is not the safest practice, however, in order to ensure reproducablitly, this is a quick and easy way achieving this goal.

After the change to the security setting the Neo4J service should be restarted:

```
service neo4j restart
```

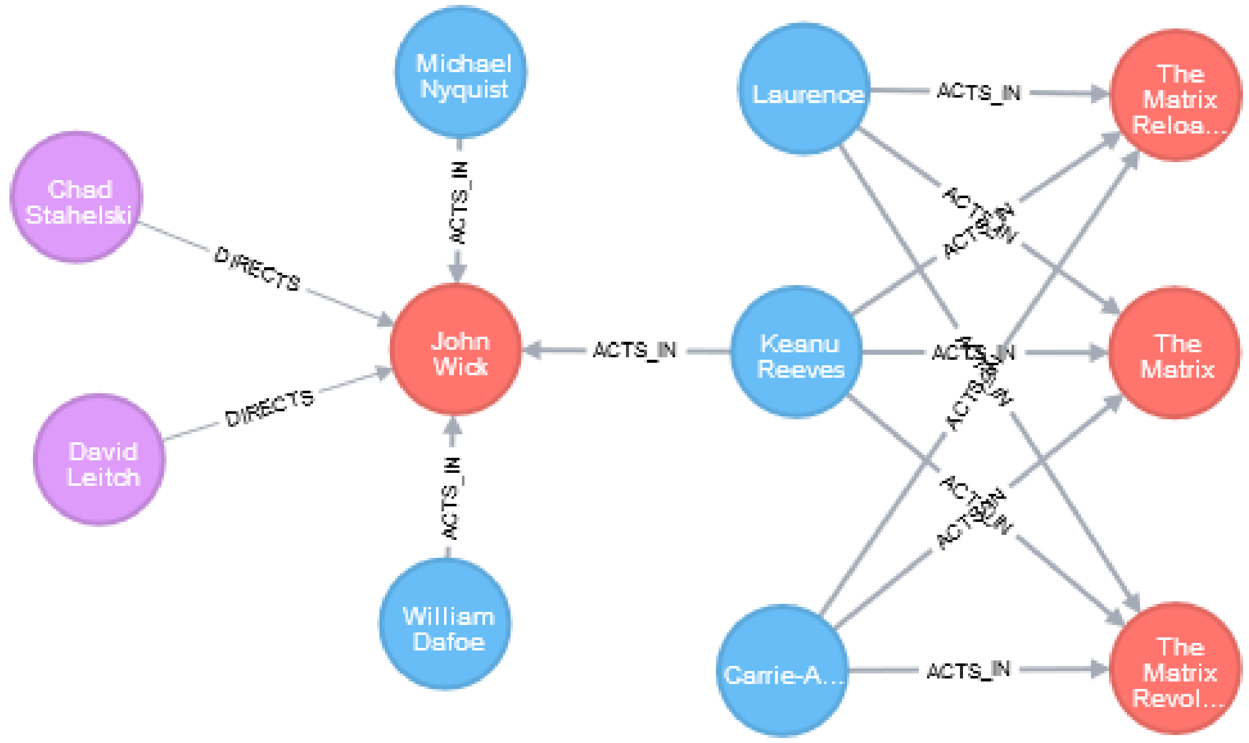
Import CSV

Having changed the the security settings the following cypher query are loading the nodes and relationsships into the database:

```
# Load nodes
query <- "load csv with headers from
          'file:///home/tim/e63-coursework/hw2/data/export_actors.csv' AS
          line CREATE (a:Actor {id:line.id,name:line.name})"
cypher(graph, query)

query <- "load csv with headers from
          'file:///home/tim/e63-coursework/hw2/data/export_directors.csv' AS
          line CREATE (d:Director { id:line.id,name:line.name})"
cypher(graph, query)

query <- "load csv with headers from
          'file:///home/tim/e63-coursework/hw2/data/export_movies.csv' AS
          line CREATE (m:Movie { id:line.id, title:line.title, year:line.year})"
cypher(graph, query)

# Load relationsshhips
query <- "load csv with headers from
          'file:///home/tim/e63-coursework/hw2/data/export_acts_in.csv' AS
          line MATCH (m:Movie { title:line.title }) MATCH (a:Actor { name:line.name })
          CREATE (a)-[:ACTS_IN { roles: [line.role]}]->(m)"
cypher(graph, query)

query <- "load csv with headers from
          'file:///home/tim/e63-coursework/hw2/data/export_directs_in.csv' AS
          line MATCH (m:Movie { title:line.title }) MATCH (d:Director { name:line.name })
          CREATE (d)-[:DIRECTS]->(m)"
cypher(graph, query)
```

Visualization

To validate the result, the following image can be brought up using the Neo4j browser.



Problem 6 (15%)

Find a way to use the Arrow Tool (http://www.apcjones.com/arrows/) to paint a relationship between a dog and his owner who live in New York and walk through the Central Park on Sunday afternoon. Add Labels and necessary properties to all nodes and relationships.

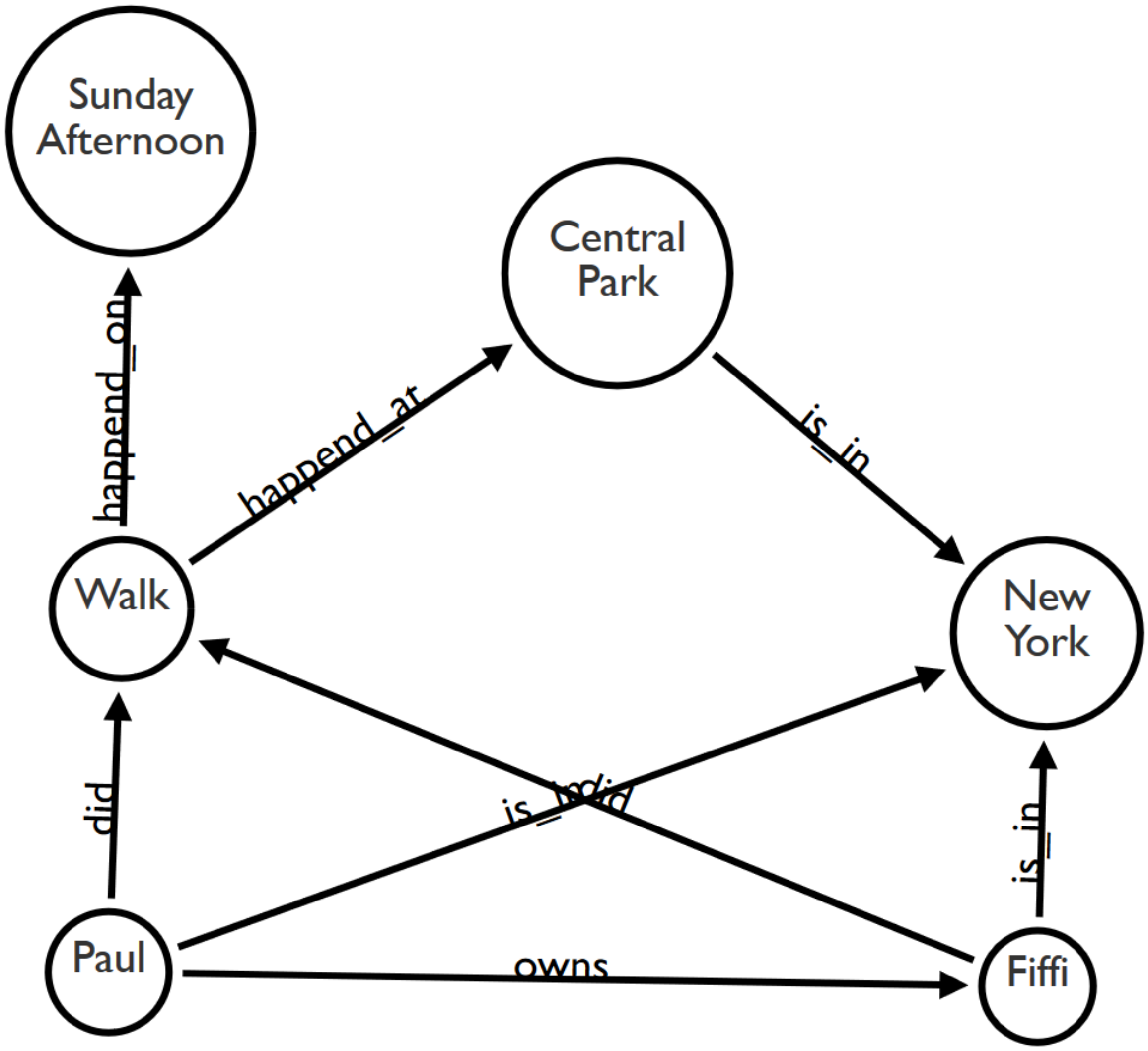
Clear database

Before starting with the exercise we're clearing the database:

```
clear(graph, input=FALSE)
```

Arrow tool

The arrow tool by Alistair Jones is a JavaScript library for drawing diagrams of small graphs, using D3. A simply visualization of the above relationsship looks in the arrow tool like the image below:



The above graph/relationsships could be modeled in a multitude of ways. It could for example also be made more granular, e.g., the time node *Sunday Afternoon* could be modeled by *Time:Afternoon* beeing a part of *Time:Sunday*. However, in my eyes, doing this would make the above representation less clear.

Cypher statment

Export your graph in Cypher format and then adjust (if necessary) generated Cypher so that you can create that graph in Neo4J database. Verify that your graph is indeed created using Cypher Browser.

The above graph can be exported (with some adjustments) into the following cypher statement:

```
CREATE (paul:Human { label:'Paul' })
CREATE (fiffi:Dog { label:'Fiffi' })
CREATE (park:Place { label:'Central Park' })
CREATE (newyork:Place { label:'New York' })
CREATE (walk:Event { label:'Walk' })
CREATE (sunafternoon:Time { label:'Sunday Afternoon' })
CREATE (paul)-[:owns]->(fiffi)
CREATE (paul)-[:is_in]->(newyork)
CREATE (fiffi)-[:is_in]->(newyork)
CREATE (park)-[:is_in]->(newyork)
CREATE (paul)-[:did]->(walk)
CREATE (fiffi)-[:did]->(walk)
CREATE (walk)-[:happend_on]->(sunafternoon)
CREATE (walk)-[:happend_at]->(park)
```

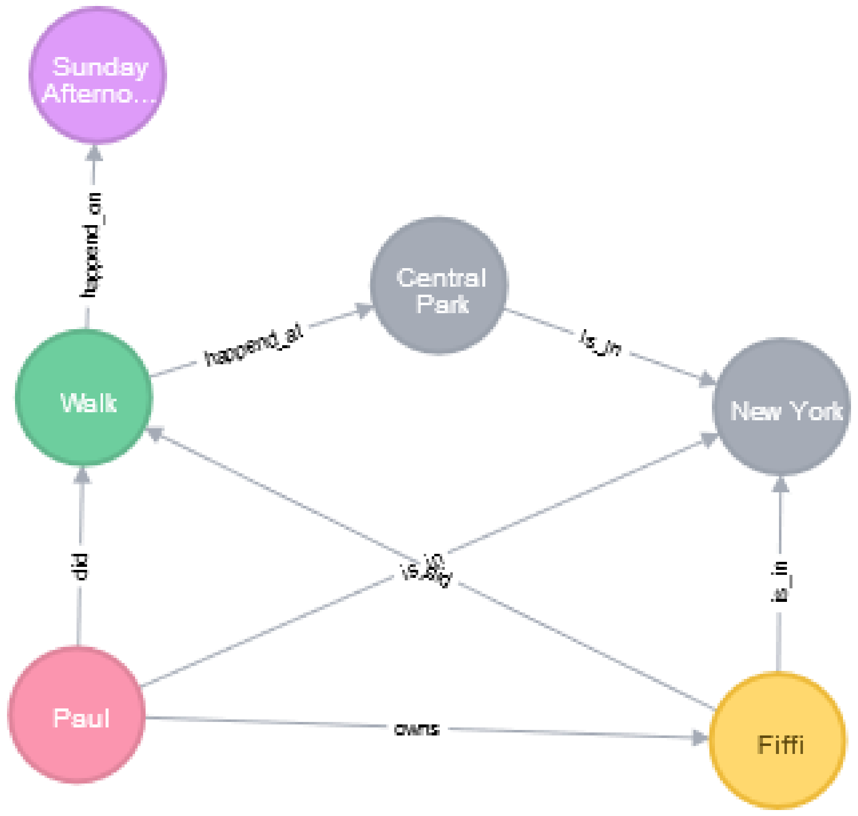
CURL

As seen before, the above cypher statment can be translated in a JSON file in order to execute it trough CURL.

```
curl -i -H accept:application/json -H content-type:application/json -XPOST \
http://neo4j:neo4jneo4j@34.251.36.174:7474/db/data/transaction/commit --data \
@data/cypher_node4.json
```

Visualization

To validate the result, the following image can be brought up using the Neo4j browser.



Appendix: Neo4J Setup

In order to reproduce the above results, a Neo4J database has to be setup. Because different OS enviroments can lead to different results, a cloud solution with AWS was chosen. The following instructions will focus on a single, well-defined goal: setting up a Neo4J database in the cloud. The only prerequisite is an AWS account.

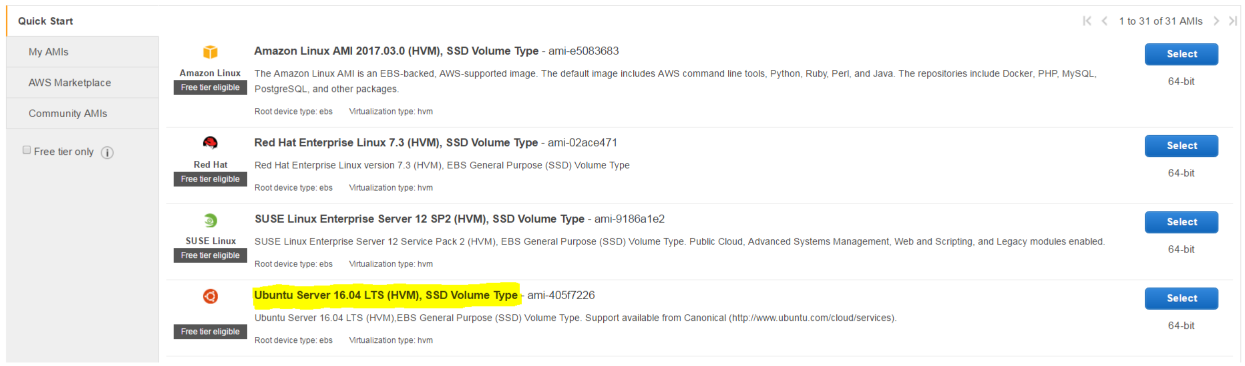
Deploy EC2

Log in to the AWS console (https://aws.amazon.com) and click on the EC2 icon under compute.

Create instance

Search for "EC2" in the search bar. Click on "EC2" to start the EC2 wizard.

Step 1: Choose an Amazon Machine Image (AMI)



Amazon Linux AMI 2017.03.0 (HVM), SSD Volume Type - ami-e5083683
The Amazon Linux AMI is an EBS-backed, AWS-supported image. The default image includes AWS command line tools, Python, Ruby, Perl, and Java. The repositories include Docker, PHP, MySQL, PostgreSQL, and other packages.
Root device type: ebs Virtualization type: hvm

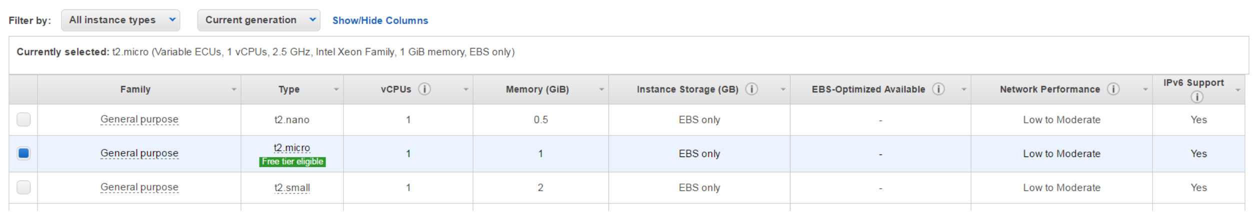
Red Hat Enterprise Linux 7.3 (HVM), SSD Volume Type - ami-02ace471
Red Hat Enterprise Linux version 7.3 (HVM), EBS General Purpose (SSD) Volume Type
Root device type: ebs Virtualization type: hvm

SUSE Linux Enterprise Server 12 SP2 (HVM), SSD Volume Type - ami-9186a1e2
SUSE Linux Enterprise Server 12 Service Pack 2 (HVM), EBS General Purpose (SSD) Volume Type. Public Cloud, Advanced Systems Management, Web and Scripting, and Legacy modules enabled.
Root device type: ebs Virtualization type: hvm

Ubuntu Server 16.04 LTS (HVM), SSD Volume Type - ami-405f7226
Ubuntu Server 16.04 LTS (HVM),EBS General Purpose (SSD) Volume Type. Support available from Canonical (http://www.ubuntu.com/cloud/services).
Root device type: ebs Virtualization type: hvm

- Select the Ubuntu Server 16.04 image

Step 2: Choose an Instance Type

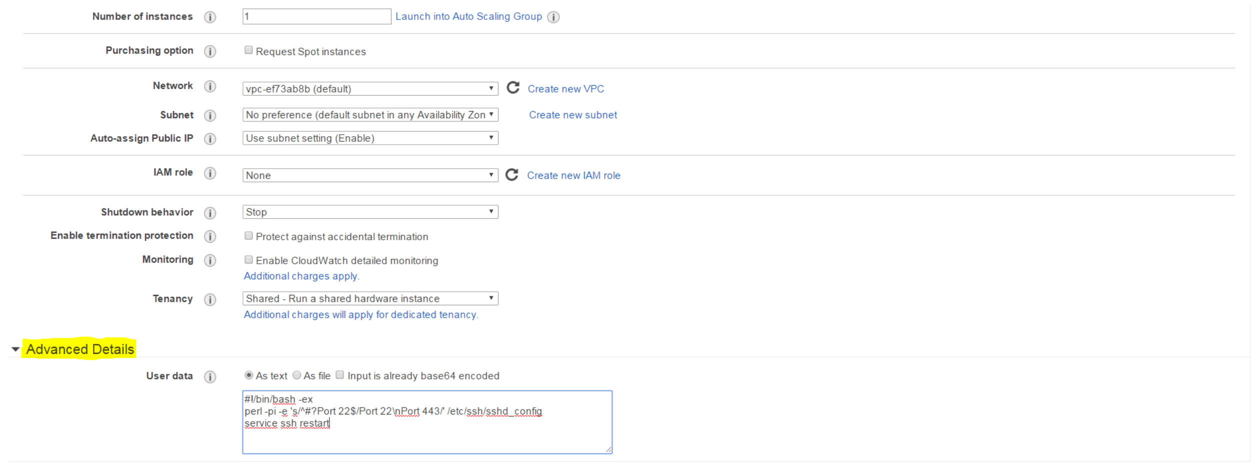


Currently selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
☐	General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
■	General purpose	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate	Yes
☐	General purpose	t2.small	1	2	EBS only	-	Low to Moderate	Yes

- Select the free tier t2.micro instance. This can also be changed later on when more capacity is needed.

Step 3: Configure Instance Details



Number of instances	1 — Launch into Auto Scaling Group
Purchasing option	☐ Request Spot instances
Network	vpc-ef73ab8b (default) — Create new VPC
Subnet	No preference (default subnet in any Availability Zon — Create new subnet
Auto-assign Public IP	Use subnet setting (Enable)
IAM role	None — Create new IAM role
Shutdown behavior	Stop
Enable termination protection	☐ Protect against accidental termination
Monitoring	☐ Enable CloudWatch detailed monitoring. Additional charges apply.
Tenancy	Shared - Run a shared hardware instance. Additional charges will apply for dedicated tenancy.

Advanced Details

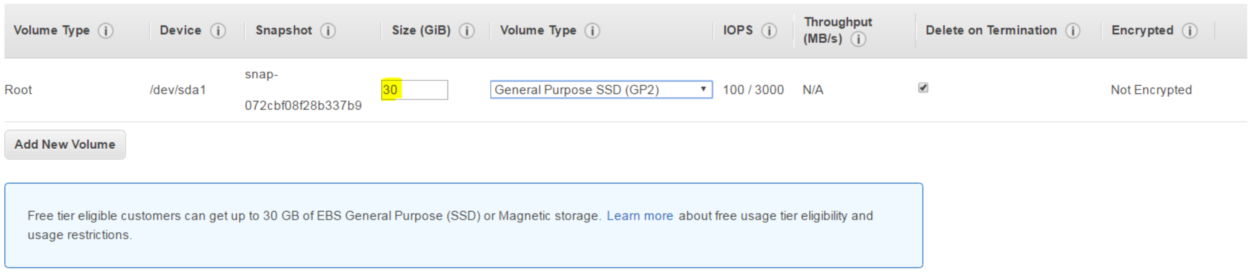
User data: ● As text ○ As file ☐ Input is already base64 encoded

```
#!/bin/bash -ex
perl -pi -e 's/^#?Port 22$/Port 22\nPort 443/' /etc/ssh/sshd_config
service ssh restart
```

- When using other instances such as p2.xlarge, click on "Request Spot Instances". Spot instances are significantly cheaper than normal instances. It is a way for Amazon to sell excess capacity at reduced prices.
- (Optional) Click on "Enable CloudWatch Detailed Monitoring". This will enable additional services like automatically shutting down an idle instance. You will be warned that additional charges may be incurred, which will go against your allotment. Consider it like buying insurance.
- (Optional) Under Advanced Details a custom startup script can be run. This can be useful, when you're behind a company firewall and the ssh port (22) is blocked. In order to circumvent this, the following bash/perl script can be run. It runs the ssh on port 443:

```
#!/bin/bash -ex
perl -pi -e 's/^#?Port 22$/Port 22\nPort 443/' /etc/ssh/sshd_config
  service ssh restart
```

Step 4: Add Storage



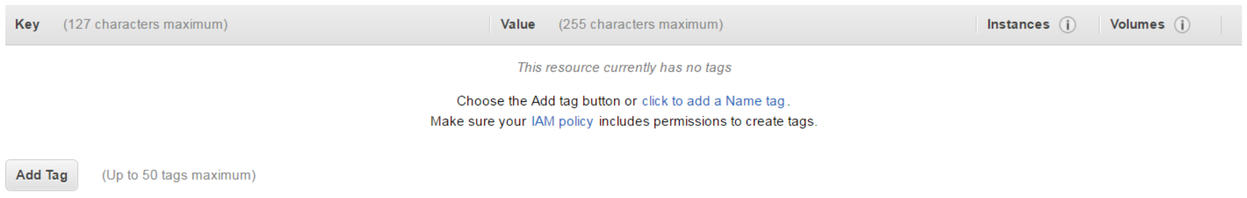
Volume Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Throughput (MB/s)	Delete on Termination	Encrypted
Root	/dev/sda1	snap-072cbf08f28b337b9	30	General Purpose SSD (GP2)	100 / 3000	N/A	✓	Not Encrypted

Add New Volume

Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage. Learn more about free usage tier eligibility and usage restrictions.

- The default SSD storage of 8 GB is sometimes not enough when installing all the necessary software. That is why 30GB is chosen.

Step 5: Add Tags



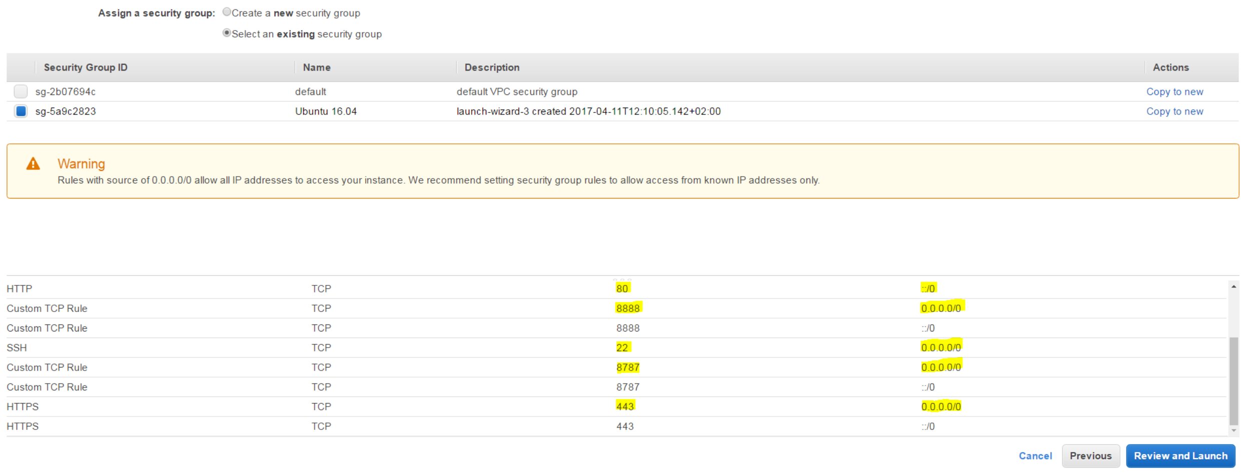
Key (127 characters maximum)	Value (255 characters maximum)	Instances	Volumes

This resource currently has no tags

Choose the Add tag button or click to add a Name tag.
Make sure your IAM policy includes permissions to create tags.

- Nothing is changed/done here.

Step 6: Configure Security Group



Assign a security group: Create a **new** security group [✓ Select an **existing** security group]

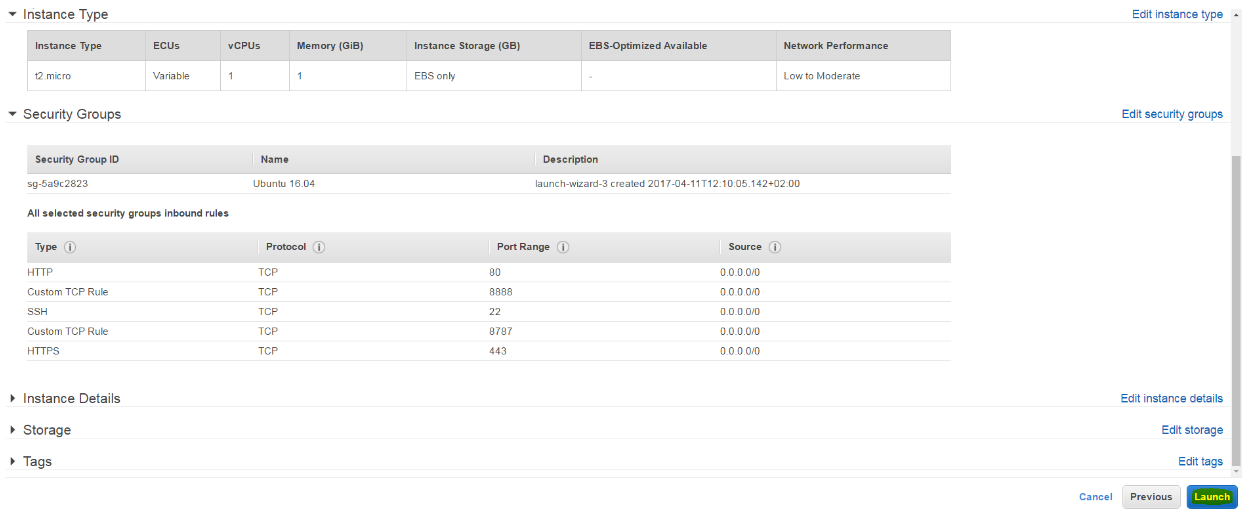
	Security Group ID	Name	Description	Actions
	sg-2b07694c	default	default VPC security group	Copy to new
✓	sg-5a9c2823	Ubuntu 16.04	launch-wizard-3 created 2017-04-11T12:10:05.142+02:00	Copy to new

Warning
Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

HTTP	TCP	80	::/0
Custom TCP Rule	TCP	8888	0.0.0.0/0
Custom TCP Rule	TCP	8888	::/0
SSH	TCP	22	0.0.0.0/0
Custom TCP Rule	TCP	8787	0.0.0.0/0
Custom TCP Rule	TCP	8787	::/0
HTTPS	TCP	443	0.0.0.0/0
HTTPS	TCP	443	::/0

- This is an important step. If the necessary ports are not opened, it isn't possible to connect to neo4j Server.
- The following ports are opened: 80 (HTTP), 22 (SSH), HTTPS (443), RStudio Port (8787), Jupyter Port (8888), Neo4J http (7474), Neo4J https (7473), Neo4J Bolt (7687).
- It is allowed to connect to them from any IP (0.0.0.0). It would also be possible to restrict the IP Range.

Step 7: Review Instance Launch



Instance Type

Instance Type	ECUs	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
t2.micro	Variable	1	1	EBS only	-	Low to Moderate

Security Groups

Security Group ID	Name	Description
sg-5a9c2823	Ubuntu 16.04	launch-wizard-3 created 2017-04-11T12:10:05.142+02:00

All selected security groups inbound rules

Type	Protocol	Port Range	Source
HTTP	TCP	80	0.0.0.0/0
Custom TCP Rule	TCP	8888	0.0.0.0/0
SSH	TCP	22	0.0.0.0/0
Custom TCP Rule	TCP	8787	0.0.0.0/0
HTTPS	TCP	443	0.0.0.0/0

- When you attempt to launch the instance it will ask if you have a keypair.

- If you have not used SSH before and do not have a keypair, select "Create a new keypair"
- Download the .pem file and store it in a safe place
- You will need to import the .pem file to connect your instance over ssh. See below.

Login EC2

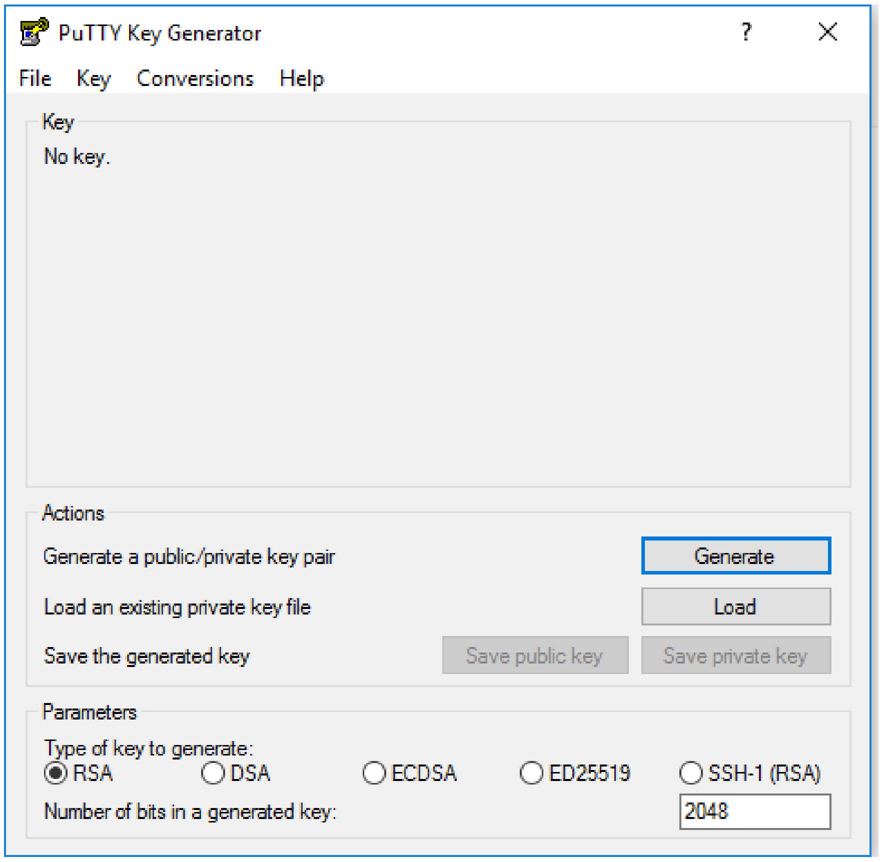
Connect to the instance over ssh which establishes a terminal session to your newly created instance.

For Windows 7 Users

Windows users may not have an ssh client installed. If you need ssh for Windows, download PuTTY (http://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html).

You will need to convert the .pem key from AWS into a .ppk key in PuTTY Key Generator - Select Import Key under Conversions - Save as Private Key with RSA selected

The below screenshot shows the PuTTy key generator.



No key.

Generate a public/private key pair

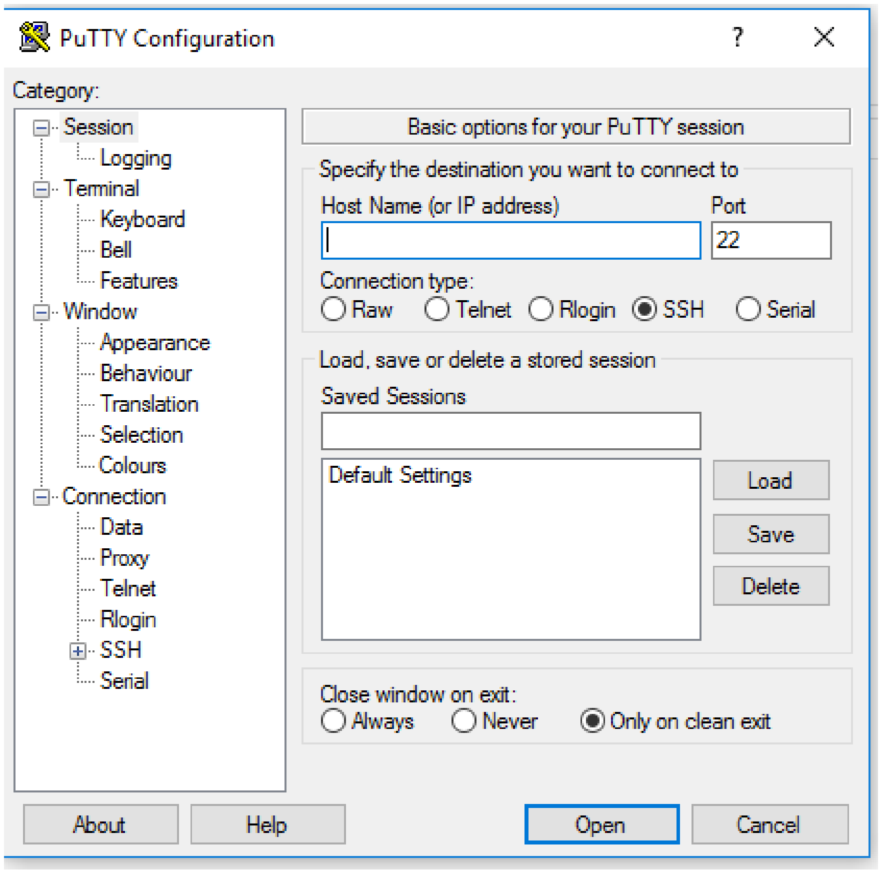
Load an existing private key file

Save the generated key

Type of key to generate:
[✓ RSA] DSA ECDSA ED25519 SSH-1 (RSA)

Number of bits in a generated key: 2048

Launch PuTTY and you will see the below screen.



Specify the destination you want to connect to

Host Name (or IP address):

Port: 22

Connection type: Raw Telnet Rlogin [✓ SSH] Serial

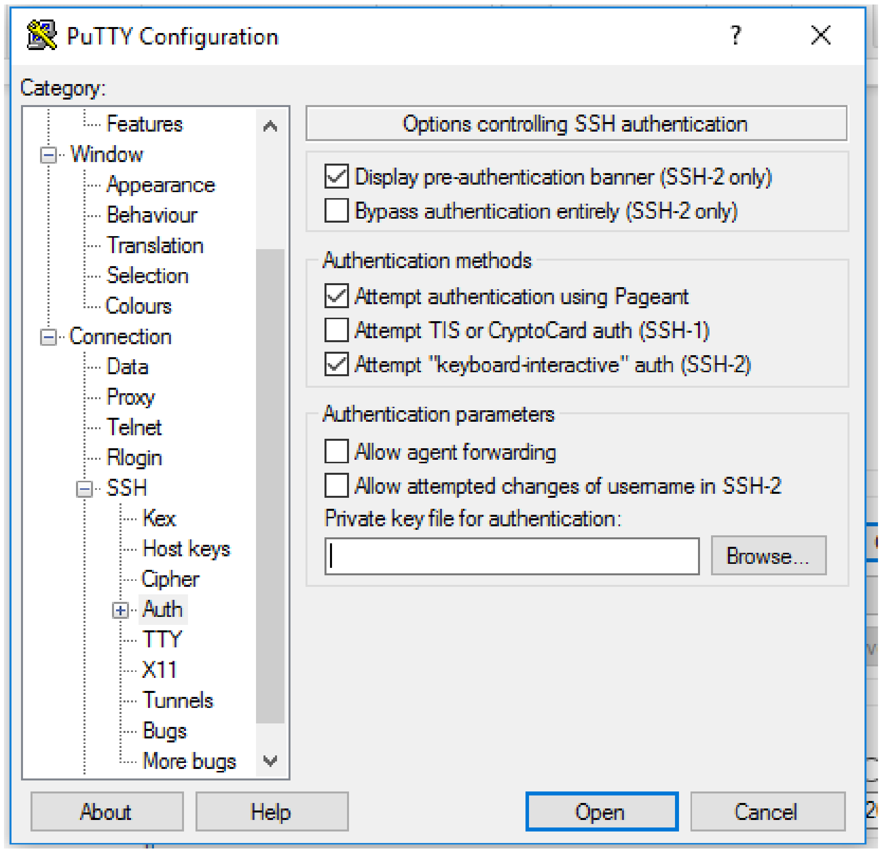
Load, save or delete a stored session

Saved Sessions

Default Settings

Close window on exit: Always Never [✓ Only on clean exit]

Navigate to SSH -> Auth, browse to add the .ppk file and Open.



Options controlling SSH authentication

- [✓] Display pre-authentication banner (SSH-2 only)
- [ ] Bypass authentication entirely (SSH-2 only)

Authentication methods

- [✓] Attempt authentication using Pageant
- [ ] Attempt TIS or CryptoCard auth (SSH-1)
- [✓] Attempt "keyboard-interactive" auth (SSH-2)

Authentication parameters

- [ ] Allow agent forwarding
- [ ] Allow attempted changes of username in SSH-2

Private key file for authentication:

Login as user ubuntu.

For MacOS and Linux or Windows 10 users

Open a terminal window.

Note: On a Mac or on Linux you may need to change the permissions of your ssh keyfile if you get the following error when attempting to ssh to your instance:

Permission denied (publickey)

Change the permission of your ssh keyfile as follows if you get the above error message.

```
chmod 600 <ssh_key>
```

Use the public IP of the running instance and username “ubuntu”. For example:

```
ssh -i <your_ssh_keyfile> ubuntu@<your_AWS_public_DNS_name_or_IP_address>
```

Add new user

```
sudo adduser <username>
sudo adduser <username> sudo
```

Copy ssh key

Copy the ssh key to the new user

```
sudo cp ~/.ssh/authorized_keys /home/<username>/.ssh/authorized_keys
su <username>
cd ~
chmod 700 .ssh
chmod 600 .ssh/authorized_keys
```

Enable Swapping

It might be necessary to enable swapping. This is especially the case with the smaller instances. You can make the size of 1024 also bigger.

```
sudo /bin/dd if=/dev/zero of=/var/swap.1 bs=1M count=1024
sudo /sbin/mkswap /var/swap.1
sudo chmod 600 /var/swap.1
sudo /sbin/swapon /var/swap.1
```

In order to activate swapping at startup append the following line to the */etc/fsab* file.

```
sudo nano /etc/fstab
swap        /var/swap.1 swap    defaults        0   0
```

Install Neo4J

In order to facilitate the neo4j installation a small bash script can be downloaded and executed from my github account.

```
wget https://raw.githubusercontent.com/greenore/linux-setup/master/setup_neo4j.sh
chmod +x setup_neo4j.sh
sudo ./setup_neo4j.sh
```

Service commands (optional)

The server should have started automatically and should also be restarted at boot. If necessary the server can be stopped with

```
service neo4j stop
```

and restarted with

```
service neo4j start
```

Edit server IP

```
sudo nano /etc/neo4j/neo4j.conf
```

Remove the # from the line below in the file

```
#org.neo4j.server.webserver.address=0.0.0.0
```

Restart the service

```
service neo4j restart
```

Accessing Neo4j

```
http://<your_AWS_public_DNS_name_or_IP_address>:7474/browser/
```