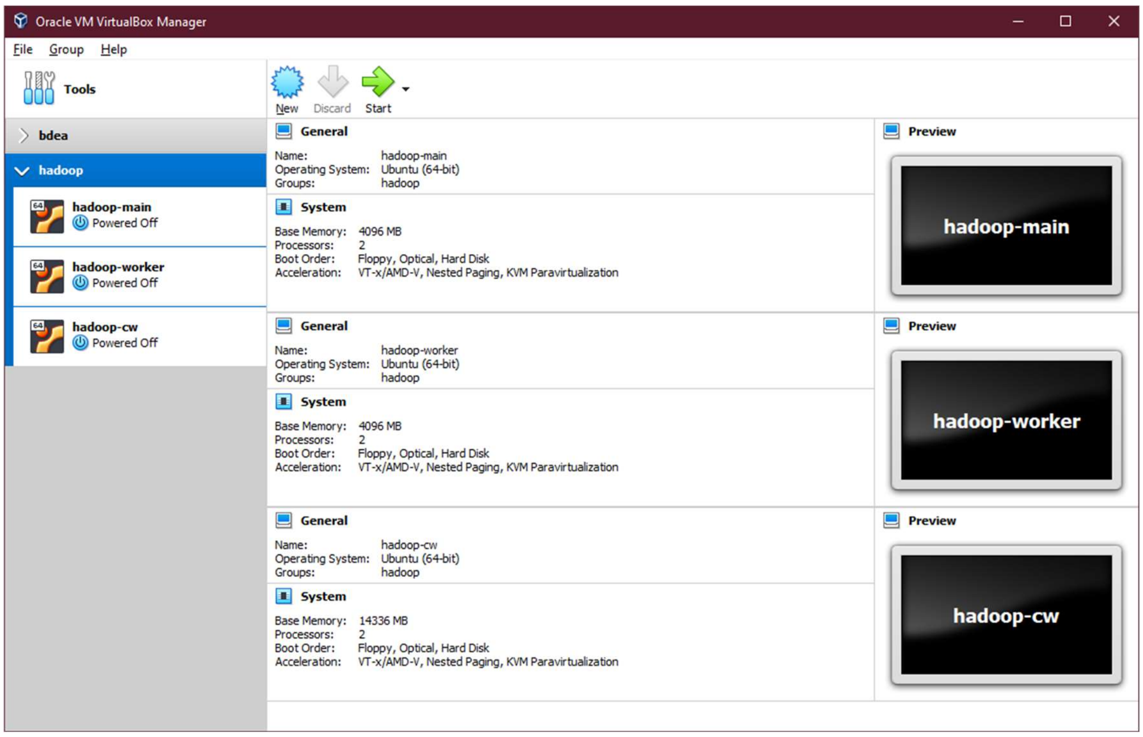
MS4S21 CW-1

Mark Baber – 17076749

Table of Contents

1 – Experiment 1

This section will detail what was carried out for experiment 1, which was to create a Hadoop cluster using Ubuntu Virtual Machines (VMs) on our local machines. This will be done with Oracles VirtualBox software. The main task was to replace the existing worker node with a new worker node with the following specifications:



General
Name: hadoop-main
Operating System: Ubuntu (64-bit)
Groups: hadoop

System
Base Memory: 4096 MB
Processors: 2
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, KVM Paravirtualization

General
Name: hadoop-worker
Operating System: Ubuntu (64-bit)
Groups: hadoop

System
Base Memory: 4096 MB
Processors: 2
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, KVM Paravirtualization

General
Name: hadoop-cw
Operating System: Ubuntu (64-bit)
Groups: hadoop

System
Base Memory: 14336 MB
Processors: 2
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, KVM Paravirtualization

(Figure 1 – Screenshot of the 3 node hadoop cluster)

Seeing as I ran into some issues with my existing cluster from the tutorials, I took a day to re-create the whole cluster again following the notes from the tutorials which I manipulated for myself. The next few steps will be the sequence of events which happened from start to finish, with a working 3 node cluster.

1.1 - Hadoop-Main

After installing Ubuntu Desktop 20.04 Long Term Support (Instruction Here), I installed the updates and openssh-server so I could ssh into the main node from my computer using Windows Terminal. This would allow me to copy and paste some of the commands I initially edited during the tutorials, to speed up my process of setting up the VMs.

1.1.1 – Set Hostname

I checked the hostname on the main node to make sure it was correctly assigned as: **hadoop-main**. This was done by editing the file with ***sudo nano /etc/hostname*** and deleting the hostname and changing it to **hadoop-main**.

1.1.2 – Set Hosts

Next was to assign the hosts correctly for the main node and the two worker nodes – I already knew I was going to set a static IP as it is good practice, so these were hard coded in a range which I knew was free on my local network. Setting the hosts file was done by typing into a terminal, **sudo nano /etc/hosts** and adding the following:

- 127.0.0.1 localhost
- #127.0.1.1 hadoop-xyz (this would be Virtual Machine hostname)
- 192.168.1.173 hadoop-main
- 192.168.1.174 hadoop-worker
- 192.168.1.175 hadoop-cw

1.1.3 – Download and Install Java

To install java on Linux, this can be done with the follow code: **sudo apt install openjdk-8-jdk** which is an open-source version of java freely distributed. Now that java is installed, the path needs to be assigned to the **~/.bashrc** file. This can be done with the following code: **sudo nano ~/.bashrc** and add the following code at the very bottom of the file:

- # JAVA
- export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre

and apply these changes with this: **source ~/.bashrc** – this is all that is currently needed for setting up java on Linux.

1.1.4 – Download and Install Hadoop

This next section will cover how to download and install Hadoop, which is required for our clusters, this code can be run from everywhere, but I usually change directory to downloads when downloading files. This can be done with:

- **cd Downloads/**
- **wget https://archive.apache.org/dist/hadoop/common/hadoop-3.2.2/hadoop-3.2.2.tar.gz**

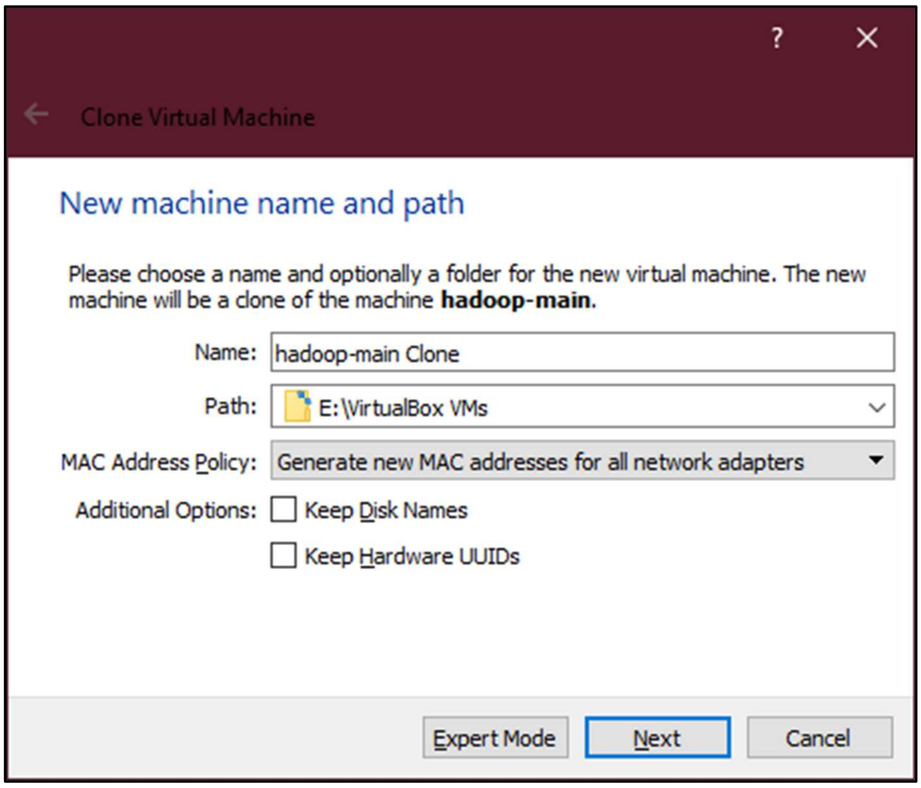
Next will be to extract the archive which can be done with **tar -xvf hadoop-3.2.2.tar.gz** and should be moved to /usr/local/hadoop, this can be done with the following code: **sudo mv hadoop-3.2.2 /usr/local/hadoop** – now that we have moved the hadoop files to our local user files, we would need to add the path to **~/.bashrc** again. This can be done with: **sudo nano ~/.bashrc** and adding the 3 lines to the bottom of the file (under the Java entry)

- # HADOOP
- export PATH=$PATH:/usr/local/hadoop/bin:/usr/local/hadoop/sbin
- export CONF=/usr/local/hadoop/etc/hadoop

and applying the changes again with **source ~/.bashrc** – this will conclude this section.

1.1.5 – Cloning the VM

After getting this far and setting up a few basic things, the next step is to clone the VM. This can be done by right clicking the VM and clicking clone with these settings:



New machine name and path

Please choose a name and optionally a folder for the new virtual machine. The new machine will be a clone of the machine **hadoop-main**.

Name: hadoop-main Clone

Path: E:\VirtualBox VMs

MAC Address Policy: Generate new MAC addresses for all network adapters

Additional Options: ☐ Keep Disk Names

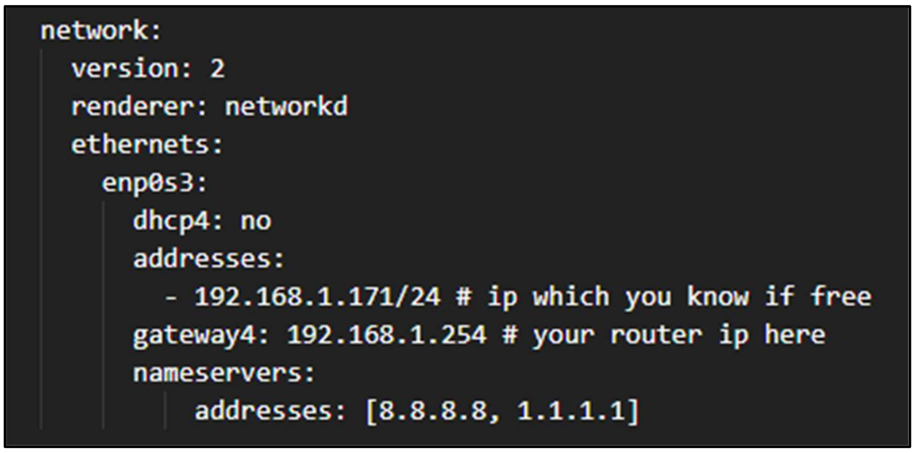
☐ Keep Hardware UUIDs

(Figure 2 – Screenshot of Cloning a VM)

And using the **Full Clone** option for the next setting – this will create (as the name suggests) a full clone of the VM but will generate a new mac address for the network adapter.

1.1.6 – Set Static IP

Now that the VM has been cloned, this is where I set a static IP for my VMs which I know are free on my local network. This can be done with the following code: **sudo nano /etc/netplan/01-network-manager-all.yaml** and pasting the following code:



```
network:
  version: 2
  renderer: networkd
  ethernets:
    enp0s3:
      dhcp4: no
      addresses:
        - 192.168.1.171/24 # ip which you know if free
      gateway4: 192.168.1.254 # your router ip here
      nameservers:
          addresses: [8.8.8.8, 1.1.1.1]
```

(Figure 3 – Screenshot of how to set static IP)

1.1.7 – Generate SSH Keys

The next step will be to generate SSH keys from the main node to the two worker nodes, this is an easy task to do, but will need to be done after completing a few dependencies:

- Changing the hostname of the 2 worker nodes
- Setting static IPs for the 2 worker nodes

To generate your ssh keys from the main node, this can be done with: **ssh-keygen -t rsa** and accepting the default (which is not best practice, but this is only local). These keys would then need to be copied to the two worker nodes. This can be done with:

- **ssh-copy-id hadoop-cw@hadoop-main** # copy the key to main
- **ssh-copy-id hadoop-cw@hadoop-worker** # copy the key to worker 1
- **ssh-copy-id hadoop-cw@worker-cw** # copy the key to worker 2

These can be tested with all the machine running and typing from the main node: **ssh hadoop-cw@hadoop-worker** – you should be asked about accepting the connection and asked for the password. If successful you should see the terminal for **hadoop-cw@hadoop-worker**.

1.1.8 – Modifying Hadoop Files

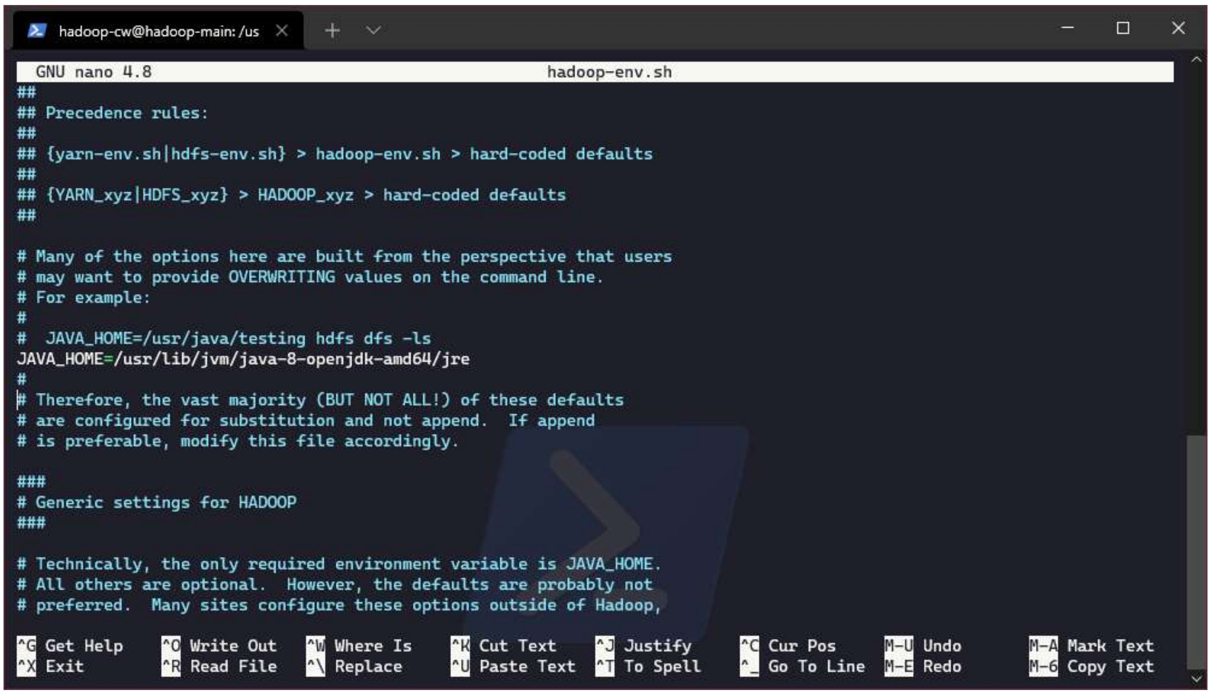
This step requires a few hadoop files to be modified, seeing as this will require going back and forth the path, we will create a command which will make changing directories much easier. This can be done by adding this code to the **~/.bashrc** file again, with: **sudo nano ~/.bashrc** and at the bottom of the file adding: **export CONF=usr/local/hadoop/etc/hadoop**, saving the changes and typing **source ~/.bashrc**. This will allow us to easily navigator to the hadoop folder with **cd $CONF/**.

1.1.8.1 – hadoop-env.sh

Now this has been added, we need to change 4 files for the main node, these files are:

- hadoop-env.sh
- core-site.xml
- hdfs-site.xml
- workers

The first file will be changed with **sudo nano $CONF/hadoop-env.sh**, for this file we need to add the Java path which we created earlier. Within this file look for the java which is commented out and add our customised one like so:



```
  GNU nano 4.8                                   hadoop-env.sh
##
## Precedence rules:
##
## {yarn-env.sh|hdfs-env.sh} > hadoop-env.sh > hard-coded defaults
##
## {YARN_xyz|HDFS_xyz} > HADOOP_xyz > hard-coded defaults
##

# Many of the options here are built from the perspective that users
# may want to provide OVERWRITING values on the command line.
# For example:
#
#  JAVA_HOME=/usr/java/testing hdfs dfs -ls
JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre
#
# Therefore, the vast majority (BUT NOT ALL!) of these defaults
# are configured for substitution and not append.  If append
# is preferable, modify this file accordingly.

###
# Generic settings for HADOOP
###

# Technically, the only required environment variable is JAVA_HOME.
# All others are optional.  However, the defaults are probably not
# preferred.  Many sites configure these options outside of Hadoop,

^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos    M-U Undo      M-A Mark Text
^X Exit       ^R Read File  ^\ Replace    ^U Paste Text ^T To Spell   ^_ Go To Line M-E Redo      M-6 Copy Text
```

(Figure 4 – hadoop-env.sh modified)

1.1.8.2 – core-site.xml

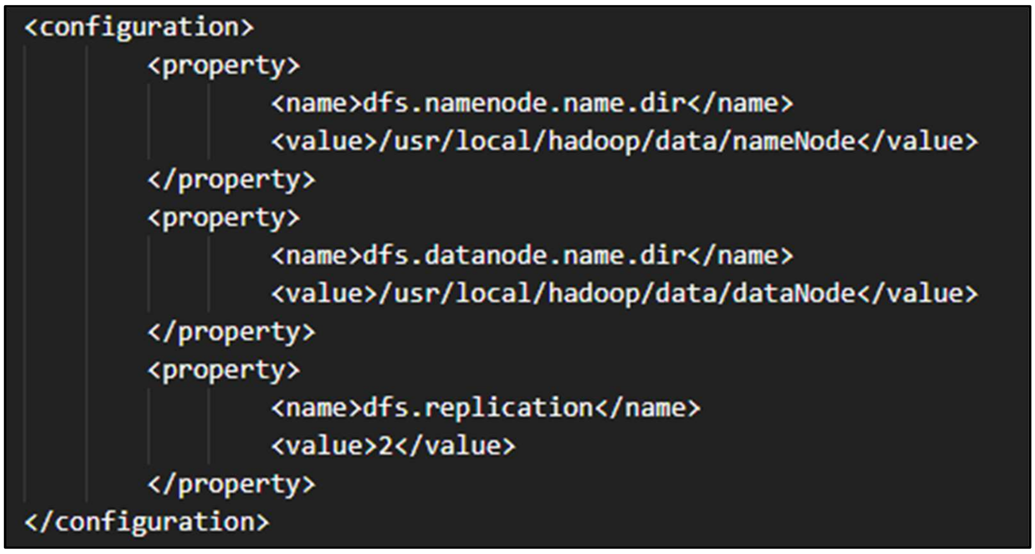
Next we will need to change core-site.xml, this can be done with **sudo nano $CONF/core-site.xml**, within this file we want to add:

```
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://hadoop-main:9000</value>
    </property>
</configuration>
```

This is all for this file.

1.1.8.3 – hdfs-site.xml

Next we will change hdfs-site.xml, this can be done with **sudo nano $CONF/hdfs-site.xml**, within this file we want to add:



```
<configuration>
        <property>
                <name>dfs.namenode.name.dir</name>
                <value>/usr/local/hadoop/data/nameNode</value>
        </property>
        <property>
                <name>dfs.datanode.name.dir</name>
                <value>/usr/local/hadoop/data/dataNode</value>
        </property>
        <property>
                <name>dfs.replication</name>
                <value>2</value>
        </property>
</configuration>
```

(Figure 5 – hdfs-site.xml)

This is all we need for this file.

1.1.8.4 – workers

The workers file is just there to list our worker nodes, for this we can access it with: **sudo nano workers**, within this file we want to add the two worker node hostnames, mine is as follows:

- hadoop-worker
- hadoop-cw

1.1.9 – Copy files to worker nodes

Now all these files have been modified, this is where we want to securely copy these files to our workers. This can be done with 2 lines of code:

- scp $CONF/* hadoop-cw@bdea-worker:/usr/local/hadoop/etc/hadoop/
- scp $CONF/* hadoop-cw@worker-cw:/usr/local/hadoop/etc/hadoop/

You should be presented with a success notification, and this concludes this section.

1.1.10 – Adding HADOOP to bashrc

To make sure we can call hadoop from everywhere on the machine, we want to add a few paths to bashrc. This can be done with **sudo nano ~/.bashrc** and within this file, again at the bottom add:

- export HADOOP_HOME=/usr/local/hadoop
- export HADOOP_COMMON_HOME=$HADOOP_HOME
- export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
- export HADOOP_HDFS_HOME=$HADOOP_HOME
- export HADOOP_MAPPED_HOME=$HADOOP_HOME

saving the changes and applying the changes with **source ~/.bashrc**. Now before moving on to running a map reduce task, there are a few more changes to add to the worker nodes.

1.1 – Hadoop-Worker

To begin with our first worker, lets assigned a new hostname as it will be the same as the main node when we cloned the VM. Seeing as these instructions have already been described, I will just link to that section from here.

The next step is to make sure we have a static IP which we can use when connecting via ssh – again these steps have been mentioned so will be linked here.

Repeat these steps for the other worker and all should be good.

1.1.1 – Modify files

This section will show the files which need to be modified to get the workers up and running, the only file which needs to be modified is yarn-site.xml. Just make sure this file has this in the file:

- **<configuration>**
  - **<property>**
    - **<name>yarn.resourcemanager.hostname</name>**
    - **<value>hadoop-main</value>**
  - **</property>**
- **</configuration>**

Now this file has been modified, we want to copy this file to the other worker node, both are all set up. This can be done with:

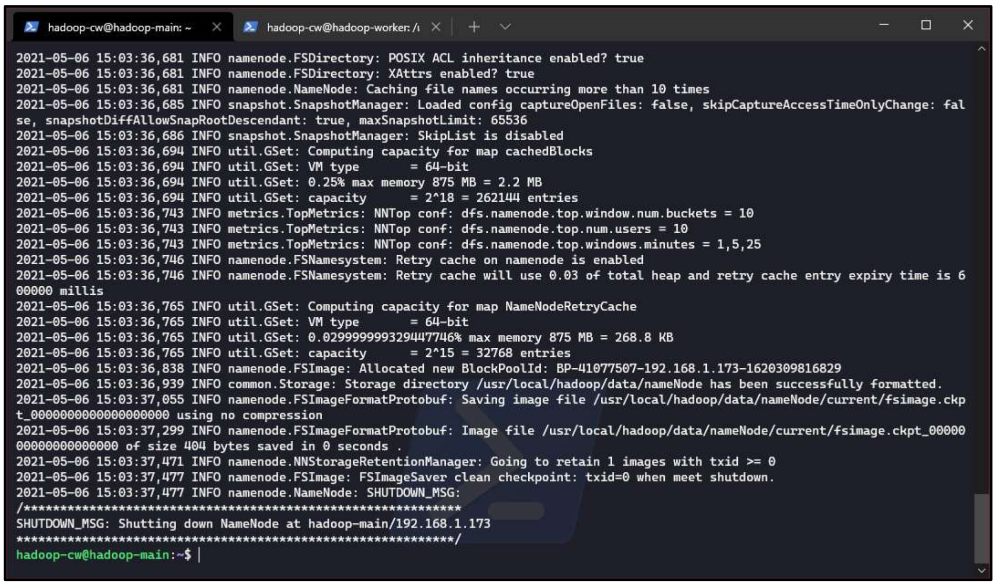
- **scp yarn-site.xml hadoop-cw@worker-cw:/usr/local/hadoop/etc/hadoop/**

After this, we are finished for the two workers and can begin to format and run a hdfs map reduce job.

1.2 – Map Reduce

To begin with getting our cluster ready to carry out a map reduce job, there is still a few things left to do. The first is so format the main node which sets up the node and looks to see if the worker nodes are ready. This can be done with the following code, which can be executed from anywhere on the main node:

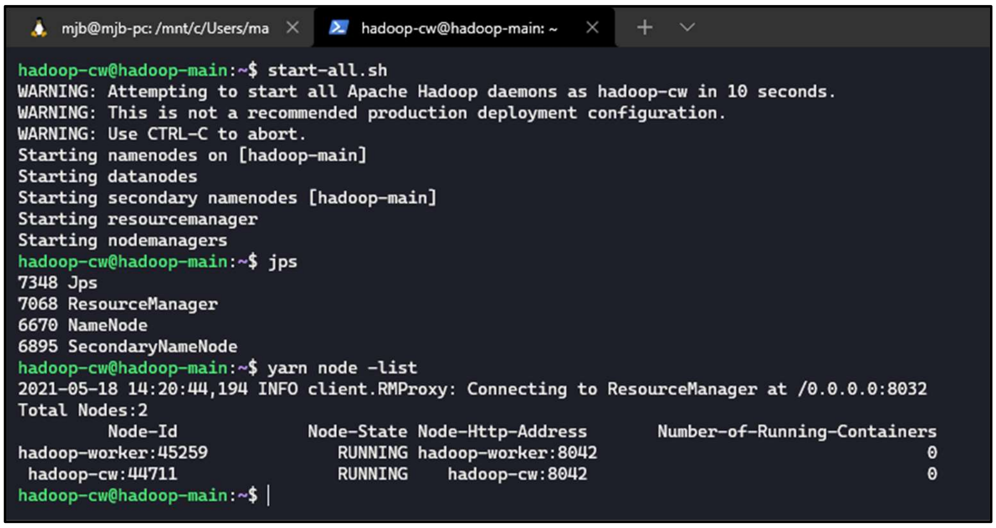
- **hadoop namenode -format**



```
2021-05-06 15:03:36,681 INFO namenode.FSDirectory: POSIX ACL inheritance enabled? true
2021-05-06 15:03:36,681 INFO namenode.FSDirectory: XAttrs enabled? true
2021-05-06 15:03:36,681 INFO namenode.NameNode: Caching file names occurring more than 10 times
2021-05-06 15:03:36,685 INFO snapshot.SnapshotManager: Loaded config captureOpenFiles: false, skipCaptureAccessTimeOnlyChange: fal
se, snapshotDiffAllowSnapRootDescendant: true, maxSnapshotLimit: 65536
2021-05-06 15:03:36,686 INFO snapshot.SnapshotManager: SkipList is disabled
2021-05-06 15:03:36,694 INFO util.GSet: Computing capacity for map cachedBlocks
2021-05-06 15:03:36,694 INFO util.GSet: VM type       = 64-bit
2021-05-06 15:03:36,694 INFO util.GSet: 0.25% max memory 875 MB = 2.2 MB
2021-05-06 15:03:36,694 INFO util.GSet: capacity      = 2^18 = 262144 entries
2021-05-06 15:03:36,743 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.window.num.buckets = 10
2021-05-06 15:03:36,743 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.num.users = 10
2021-05-06 15:03:36,743 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.windows.minutes = 1,5,25
2021-05-06 15:03:36,746 INFO namenode.FSNamesystem: Retry cache on namenode is enabled
2021-05-06 15:03:36,746 INFO namenode.FSNamesystem: Retry cache will use 0.03 of total heap and retry cache entry expiry time is 6
00000 millis
2021-05-06 15:03:36,765 INFO util.GSet: Computing capacity for map NameNodeRetryCache
2021-05-06 15:03:36,765 INFO util.GSet: VM type       = 64-bit
2021-05-06 15:03:36,765 INFO util.GSet: 0.029999999329447746% max memory 875 MB = 268.8 KB
2021-05-06 15:03:36,765 INFO util.GSet: capacity      = 2^15 = 32768 entries
2021-05-06 15:03:36,838 INFO namenode.FSImage: Allocated new BlockPoolId: BP-41077507-192.168.1.173-1620309816829
2021-05-06 15:03:36,939 INFO common.Storage: Storage directory /usr/local/hadoop/data/nameNode has been successfully formatted.
2021-05-06 15:03:37,055 INFO namenode.FSImageFormatProtobuf: Saving image file /usr/local/hadoop/data/nameNode/current/fsimage.ckp
t_0000000000000000000 using no compression
2021-05-06 15:03:37,299 INFO namenode.FSImageFormatProtobuf: Image file /usr/local/hadoop/data/nameNode/current/fsimage.ckpt_00000
00000000000000 of size 404 bytes saved in 0 seconds .
2021-05-06 15:03:37,471 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0
2021-05-06 15:03:37,477 INFO namenode.FSImage: FSImageSaver clean checkpoint: txid=0 when meet shutdown.
2021-05-06 15:03:37,477 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at hadoop-main/192.168.1.173
************************************************************/
hadoop-cw@hadoop-main:~$ |
```

(Figure 6 – Hadoop Main Format Response)

After running this command, we can start the distributed file system (dfs), this can be done with **start-dfs.sh** or **start-all.sh**. This will have an output as seen below:



```
hadoop-cw@hadoop-main:~$ start-all.sh
WARNING: Attempting to start all Apache Hadoop daemons as hadoop-cw in 10 seconds.
WARNING: This is not a recommended production deployment configuration.
WARNING: Use CTRL-C to abort.
Starting namenodes on [hadoop-main]
Starting datanodes
Starting secondary namenodes [hadoop-main]
Starting resourcemanager
Starting nodemanagers
hadoop-cw@hadoop-main:~$ jps
7348 Jps
7068 ResourceManager
6670 NameNode
6895 SecondaryNameNode
hadoop-cw@hadoop-main:~$ yarn node -list
2021-05-18 14:20:44,194 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
Total Nodes:2
         Node-Id             Node-State Node-Http-Address       Number-of-Running-Containers
hadoop-worker:45259             RUNNING hadoop-worker:8042                                 0
 hadoop-cw:44711                RUNNING     hadoop-cw:8042                                 0
hadoop-cw@hadoop-main:~$ |
```

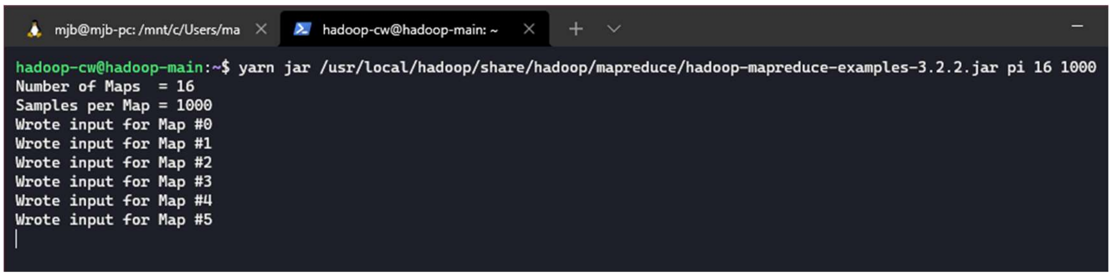
(Figure 7 – start-all.sh)

Now the cluster has been formatted and checked with **yarn node -list** (as seen above), this means our cluster is ready to run a map reduce job.

To run a map-reduce job, for this example lets run one of the built-in examples which is to estimate the value of pi. This can be done with the following code:

- **yarn jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.2.2.jar pi 16 50000**

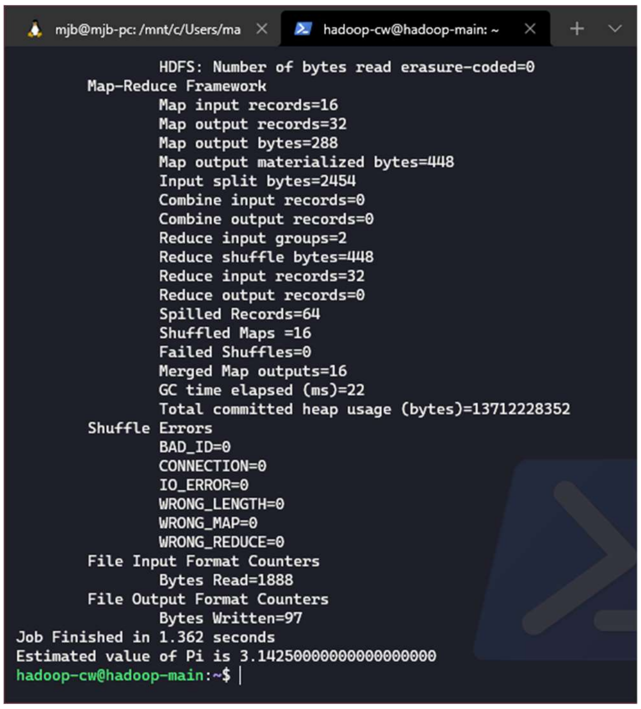
Where we are running a yarn jar file to estimate pi, with 16 map tasks and 50000 examples. Hopefully, you get no errors, and the output will look like the figure below.



```
hadoop-cw@hadoop-main:~$ yarn jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.2.2.jar pi 16 1000
Number of Maps  = 16
Samples per Map = 1000
Wrote input for Map #0
Wrote input for Map #1
Wrote input for Map #2
Wrote input for Map #3
Wrote input for Map #4
Wrote input for Map #5
```

(Figure 8 – Map Reduce Job Running)

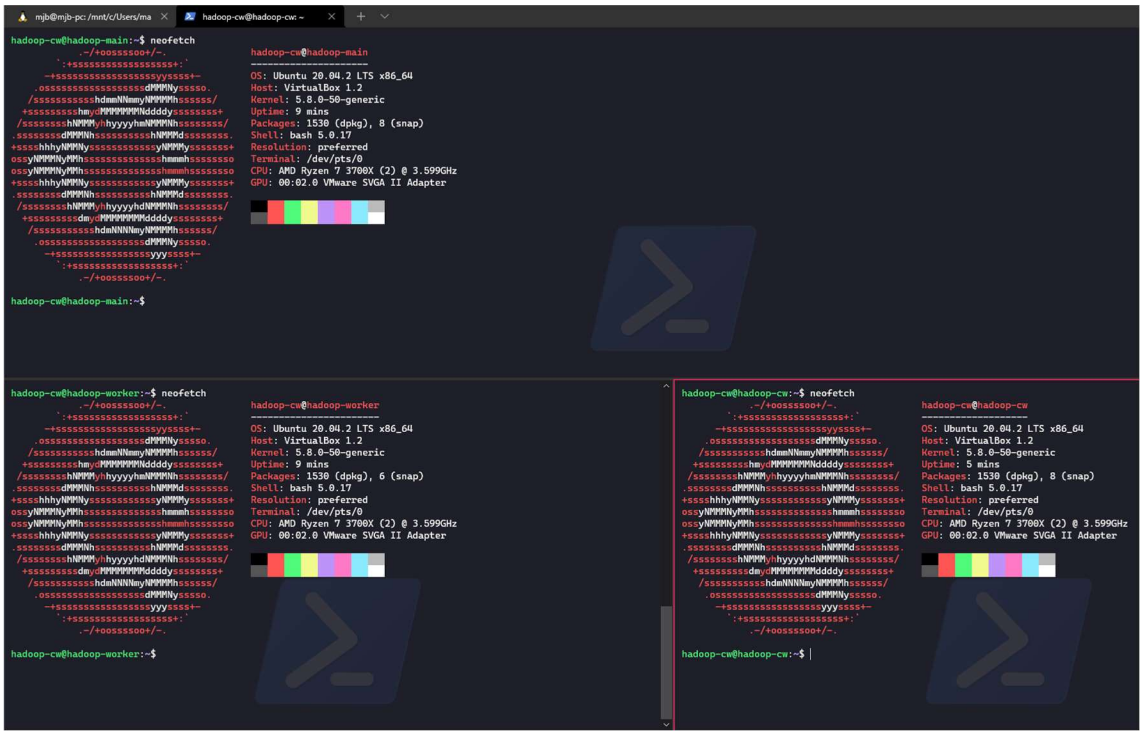
With the end looking like this:



```
                HDFS: Number of bytes read erasure-coded=0
        Map-Reduce Framework
                Map input records=16
                Map output records=32
                Map output bytes=288
                Map output materialized bytes=448
                Input split bytes=2454
                Combine input records=0
                Combine output records=0
                Reduce input groups=2
                Reduce shuffle bytes=448
                Reduce input records=32
                Reduce output records=0
                Spilled Records=64
                Shuffled Maps =16
                Failed Shuffles=0
                Merged Map outputs=16
                GC time elapsed (ms)=22
                Total committed heap usage (bytes)=13712228352
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=1888
        File Output Format Counters
                Bytes Written=97
Job Finished in 1.362 seconds
Estimated value of Pi is 3.14250000000000000000
hadoop-cw@hadoop-main:~$
```

(Figure 9 – Map Reduce Job Finished)

This concludes this section of the report which looks to set up a 3-node cluster and run a map reduce task. With the 3 nodes and their system resources found below:



```
hadoop-cw@hadoop-main:~$ neofetch
            .-/+oossssoo+/-.               hadoop-cw@hadoop-main
        `:+ssssssssssssssssss+:`           ---------------------
      -+ssssssssssssssssssyyssss+-         OS: Ubuntu 20.04.2 LTS x86_64
    .ossssssssssssssssssdMMMNysssso.       Host: VirtualBox 1.2
   /ssssssssssshdmmNNmmyNMMMMhssssss/      Kernel: 5.8.0-50-generic
  +ssssssssshmydMMMMMMMNddddyssssssss+     Uptime: 9 mins
 /sssssssshNMMMyhhyyyyhmNMMMNhssssssss/    Packages: 1530 (dpkg), 8 (snap)
.ssssssssdMMMNhsssssssssshNMMMdssssssss.   Shell: bash 5.0.17
+sssshhhyNMMNyssssssssssssyNMMMysssssss+   Resolution: preferred
ossyNMMMNyMMhsssssssssssssshmmmhssssssso   Terminal: /dev/pts/0
ossyNMMMNyMMhsssssssssssssshmmmhssssssso   CPU: AMD Ryzen 7 3700X (2) @ 3.599GHz
+sssshhhyNMMNyssssssssssssyNMMMysssssss+   GPU: 00:02.0 VMware SVGA II Adapter
.ssssssssdMMMNhsssssssssshNMMMdssssssss.
 /sssssssshNMMMyhhyyyyhdNMMMNhssssssss/
  +sssssssssdmydMMMMMMMMddddyssssssss+
   /ssssssssssshdmNNNNmyNMMMMhssssss/
    .ossssssssssssssssssdMMMNysssso.
      -+sssssssssssssssssyyyssss+-
        `:+ssssssssssssssssss+:`
            .-/+oossssoo+/-.

hadoop-cw@hadoop-main:~$
```

```
hadoop-cw@hadoop-worker:~$ neofetch
            .-/+oossssoo+/-.               hadoop-cw@hadoop-worker
        `:+ssssssssssssssssss+:`           -----------------------
      -+ssssssssssssssssssyyssss+-         OS: Ubuntu 20.04.2 LTS x86_64
    .ossssssssssssssssssdMMMNysssso.       Host: VirtualBox 1.2
   /ssssssssssshdmmNNmmyNMMMMhssssss/      Kernel: 5.8.0-50-generic
  +ssssssssshmydMMMMMMMNddddyssssssss+     Uptime: 9 mins
 /sssssssshNMMMyhhyyyyhmNMMMNhssssssss/    Packages: 1530 (dpkg), 6 (snap)
.ssssssssdMMMNhsssssssssshNMMMdssssssss.   Shell: bash 5.0.17
+sssshhhyNMMNyssssssssssssyNMMMysssssss+   Resolution: preferred
ossyNMMMNyMMhsssssssssssssshmmmhssssssso   Terminal: /dev/pts/0
ossyNMMMNyMMhsssssssssssssshmmmhssssssso   CPU: AMD Ryzen 7 3700X (2) @ 3.599GHz
+sssshhhyNMMNyssssssssssssyNMMMysssssss+   GPU: 00:02.0 VMware SVGA II Adapter
.ssssssssdMMMNhsssssssssshNMMMdssssssss.
 /sssssssshNMMMyhhyyyyhdNMMMNhssssssss/
  +sssssssssdmydMMMMMMMMddddyssssssss+
   /ssssssssssshdmNNNNmyNMMMMhssssss/
    .ossssssssssssssssssdMMMNysssso.
      -+sssssssssssssssssyyyssss+-
        `:+ssssssssssssssssss+:`
            .-/+oossssoo+/-.

hadoop-cw@hadoop-worker:~$
```

```
hadoop-cw@hadoop-cw:~$ neofetch
            .-/+oossssoo+/-.               hadoop-cw@hadoop-cw
        `:+ssssssssssssssssss+:`           -------------------
      -+ssssssssssssssssssyyssss+-         OS: Ubuntu 20.04.2 LTS x86_64
    .ossssssssssssssssssdMMMNysssso.       Host: VirtualBox 1.2
   /ssssssssssshdmmNNmmyNMMMMhssssss/      Kernel: 5.8.0-50-generic
  +ssssssssshmydMMMMMMMNddddyssssssss+     Uptime: 5 mins
 /sssssssshNMMMyhhyyyyhmNMMMNhssssssss/    Packages: 1530 (dpkg), 8 (snap)
.ssssssssdMMMNhsssssssssshNMMMdssssssss.   Shell: bash 5.0.17
+sssshhhyNMMNyssssssssssssyNMMMysssssss+   Resolution: preferred
ossyNMMMNyMMhsssssssssssssshmmmhssssssso   Terminal: /dev/pts/0
ossyNMMMNyMMhsssssssssssssshmmmhssssssso   CPU: AMD Ryzen 7 3700X (2) @ 3.599GHz
+sssshhhyNMMNyssssssssssssyNMMMysssssss+   GPU: 00:02.0 VMware SVGA II Adapter
.ssssssssdMMMNhsssssssssshNMMMdssssssss.
 /sssssssshNMMMyhhyyyyhdNMMMNhssssssss/
  +sssssssssdmydMMMMMMMMddddyssssssss+
   /ssssssssssshdmNNNNmyNMMMMhssssss/
    .ossssssssssssssssssdMMMNysssso.
      -+sssssssssssssssssyyyssss+-
        `:+ssssssssssssssssss+:`
            .-/+oossssoo+/-.

hadoop-cw@hadoop-cw:~$
```

(Figure 10 – 3 Node Cluster System Resources)

2 – Experiment 2

This section of the report will go over how to stream tweets from the website Twitter using their API with a third-party API called Tweepy. This will be done using a Jupyter-Notebook and python, which is being run from the package Anaconda.

2.1 - Twitter Streamer

To start with this experiment, the first thing to do was to import two modules which are required for this task. This can be done with this:

```
# install modules
import tweepy, json
```

Next is to set up the security keys from the twitter developer account:

```
consumer_key = 'y1Hz3CADNPj7fWOtkPfReA5Vbyzz'
consumer_secret = 'sz3SszPNyJFA0URxqq8R76poS2q37pUbVwbj7Z0ARm68B5hQSRSzz'
Bearer_Token = 'AAzAAAAAAAAAAAAAAAAAAANkfKQEAAAAApqxe6v7ejWpxazCs0S2YSJuiIJE
%3DoDJ41CBVFbZQ0cZ4KftvIAMFLCWYn0Ih3SRzvF8reI74UdqNDxzz'
access_token = '7z55358007-sDkidtK10EmFad6XaOFre7scKITWvoBMLkxkQ6KUzz'
access_token_secret = 'SzsYgAuAAFOl8m1zWWFSYJRdpL1L0u8KTZRDB2ZhbSsZZXzz'
```

(This code has been modified so it will not work.)

After setting these variables, we would need to set the authentication:

```
# set authentication
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth)
```

This is using tweepy to handle the authentication.

The next step is to create an instance of a file, which we will use to store the streamed tweets to.

```
# open a file to append the streaming data
streamedTweets = open("streamed-tweets.csv", "w")
unfilteredTweets = open("unfiltered.csv", "w")
```

For this, I have set up two files which are as follows.

- streamed-tweets – this will only save certain properties from the twitter streamer.
- unfiltered – this will save everything that comes from the twitter streamer.

This next function is a big function which is used to connected to the streamer, check for errors and picks what data to keep from the streamer.

```
class myStreamListener(tweepy.StreamListener):
        def on_connect(self):
            # Called initially to connect to the Streaming API
            print("You are now connected to the streaming API.")

        def __init__(self, api=None):
            super(myStreamListener, self).__init__()

        def on_error(self, status_code):
            if status == 420:
            # Returning False on_data method in case rate limit occurs.
                return False
                print(status)
            # On error - if an error occurs, display the error / status code
            print('An Error has occured: ' + repr(status_code))
            return False

        def on_data(self, data):
            # Get all data from twitter as datajson
            datajson = json.loads(data)
            # dump all the data to see what we can use
            json.dump(datajson, unfilteredTweets, indent = 6)

            # select what we want to keep
            createdAt = datajson['created_at']
            userName = datajson['user']['name']
            tweetText = datajson['text']
            userLoc = datajson['user']['location']
            hashTags = datajson['entities']['hashtags']
            userMentions = datajson['entities']['user_mentions']

            # get the data as a dictionary -
            Dictionary = {1:createdAt, 2:userName, 3:tweetText,
                          4:userLoc, 5:hashTags, 6:userMentions}
            # convert dict to json string
            json_string = json.dumps(Dictionary)
            # write this converted string to csv & JSON
            json.dump(json_string, streamedTweets)
```

The ***on_data*** function is the part where we specify the data and what to save, for this I tried to take all the data as ***unfilteredTweets*** along with some filtered tweets. This was to check if working with one over the other would have been easier.

The next section is a big function which takes a few parameters which can be used against the rest of the streamer to filter down the stream. This can be done with things like locations, languages, hashtags and more. This can be done with the following code:

```
def goGetTweets():
    try:
        # locations # uk
        loCal = [-3.576205, 51.466683, -3.143618, 51.825436]
        # languages
        lanG = ['en']
        # hashtags
        hashT = ['#COVID19', '#CoronaVirus', '#UnitedKingdom', '#UK', '#India']
        # set wait on rate limit
        listener = myStreamListener(api = tweepy.API(wait_on_rate_limit = True))
        # init the streamer
        streamer = tweepy.Stream(auth=auth, listener = listener)
        # print tracking
        print("Tracking location " + str('United Kingdom'))
        # DOES THIS START OR STOP THE LOOP
        streamer.filter(locations=loCal, languages = lanG, track = hashT)
    except KeyboardInterrupt:
        print("manually stopped at:", datetime.now())
    finally:
        print('Done.')
        finalTime = datetime.now()
        print(finalTime)
        print("time ran out.")
```

The main parts of this code which is important is the locations which was set with BBoxFinder, the languages which was easy enough to set and the hashtags which are the topics we are interested in tracking.

Next is setting the listening with the Tweepy API with wait on rate limit as this will make sure we do not go over our limit, setting the streamer with the authorisation and printing a string to let us know the app is tracking when it runs.

Next is the filter being applied to streamer before setting 2 excepts which will wait for a keyboard interrupt printing the time it stopped, and finally printing out done if the code runs to completion.

Before calling the function to start the stream, I tried to set up a condition which would run for x amount of time before finishing. I could not get this to work, but left the code in, here is the following code:

```
####################################
## ALL OF THIS LOGIC DOESN'T WORK :(
####################################

# add some logic to see if it loops
from datetime import datetime, timedelta
# set start
startTimer = datetime.now()
# set end
endTimer = startTimer + timedelta(seconds=5)

print('START @',startTimer)
print('Should end at', endTimer)

# start loop
while datetime.now() < endTimer:
    goGetTweets()
else:
    finalTime = datetime.now()
    print(finalTime)
    print("time ran out.")
    # close both csv files
    streamedTweets.close()
    unfilteredTweets.close()
    # disconnect from streamer
    #streamer.disconnect() # this kept throwing an error
    print("Disconnecting from Streamer")
```

When running this code, I expected it to run for 5 seconds from when the command was run. However, this was not the case so instead I left it run for several hours before being hit with a connection issue. Although this code did run long enough to get two different types of csv's saved – one which is unfiltered and in a JSON format, and the other which was a dictionary which contained only the entries which I wanted.

However, the second dataset which was a dictionary, seemed to all be within a single cell within Excel which I thought would have taken way too long to clean on submission day. Because of this I looked to the full JSON formatted file, which will be used in the next step.

2.2 – Data Pre-Processing

To begin with the data pre-processing, I started by creating a new notebook so I could keep things simple and clean. I then started by importing json, csv, pandas and importing the full ***unfiltered*** dataset which was dumped out from the last section.

```
import json, csv
import pandas as pd
```

```
myFile = open('unfiltered.csv','r') # read the data
data = json.load(myFile)
```

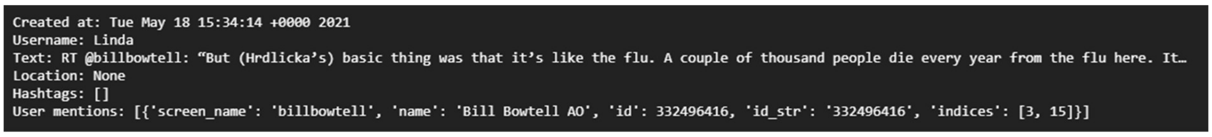
```
print(data[1]) # print all the data for the first entry
print(data[1]['created_at']) # print the create at for the first entry
```

There were some issues before getting the data to import correctly, when I was dumping out the data from section 2.1 the csv file which saved as a json format (indented) did not have commas between the curly brackets (**This = }{**) which needed to be converted to (**This = },{**). This was difficult as the data was too big to replace all with Microsoft Excel (Obviously), it was also too big to clean with Microsoft Visual Studio Code (this was a surprise), but I was able to replace all with Notepad++ which really impressed me.

After sorting out the comma between the curly brackets, the next step was to try and find what data I wanted to keep from the dataset. As this data used a JSON format, a few things were nested and proved difficult to find.

```
print('Created at:', data[1]['created_at'])
print('Username:', data[1]['user']['name'])
print('Text:', data[1]['text'])
print('Location:', data[1]['user']['location'])
print('Hashtags:', data[1]['entities']['hashtags'])
print('User mentions:', data[1]['entities']['user_mentions'])
```

This code snippet would find all the fields that I wanted to keep by looking at the first entry, which made it a lot easier when trying to get the actual fields.



```
Created at: Tue May 18 15:34:14 +0000 2021
Username: Linda
Text: RT @billbowtell: “But (Hrdlicka’s) basic thing was that it’s like the flu. A couple of thousand people die every year from the flu here. It…
Location: None
Hashtags: []
User mentions: [{'screen_name': 'billbowtell', 'name': 'Bill Bowtell AO', 'id': 332496416, 'id_str': '332496416', 'indices': [3, 15]}]
```

(Figure 11 – Output of fields from Tweets Streamed)

Next was to get this data into a pandas dataframe (df), this was done with the following code:

```
# create an empty dataframe
df = pd.DataFrame(data)
# check type
print('df type:', type(df))
# print the info
df.info()
```

After getting the data into the df, and checking the type and info for clarity, the next step was to build out the dfs structure. This included specifying the columns and then importing the data into the correct columns.

```
myDF = pd.DataFrame(columns = ['created_at',
                                'user_name',
                                'screen_name',
                                'text',
                                'location',
                                'hashtags',
                                'mentions'])
# put created_at into the df
myDF['created_at'] = df['created_at']
myDF['text'] = df['text']
print(myDF)
```

Seeing as the created_at and text was easily put into the df, these two were done together. The other columns proved to be more difficult as it was nested, the first thought was to s/lapply it by I could not figure that out in Python. So instead, I just created a loop and appended each entry into a list, before merging the list into the dfs columns.

```
# now we want to find screen names and user names
print(df['user'][0]['screen_name'])
print(df['user'][0]['name'])
# create a list of screenNames
screenNames = []
# loop through the screen_names and append to list ScreenNames
i = 0
for screen_names in df['user']:
    screenNames.append(df['user'][i]['screen_name'])
    i+=1
# add to myDF
myDF['screen_name'] = screenNames
```

After finding out how to select a user screen name and a username, this is where I implemented the loop into a list and merged the list and df. Now this section was done, the next step was to do the same thing for the other columns (the code for this will be in the appendix to save some space).

This next section will cover the pre-processing as a few of the entities within the df were nested and difficult to deal with. This step involved tokenization and removing stop words, whilst there is a good package in R, trying to convert what we have used from R to Python was quite difficult with the different package.

The package used for Python was NLTK and this was easily imported but also required the user to download additional data for the package to work. This was done with:

```
# Removing Stop Words with NLTK
import nltk
nltk.download()
# for this you can download corpus
# or all
print(stopwords.words('english'))
```

Now that I have successfully downloaded stopwords and tested this by printing the english stopwords, the next step was to create a custom stopwords dictionary and add it to the local dictionary used. After this the next step was to create a tokenized version of the words, this was done with the following:

```
# create a list
removedSW = []

# set english stopwords
en_stops = nltk.corpus.stopwords.words('english')
# add custom stop words
new_stopwords = ['“','’','https','.', ';', ':', ')', '(', '@', '...', '//t.c
o/', ',', '.']
# add stopwords to en_stops
en_stops.extend(new_stopwords)
print(len(en_stops)) # 179 stopwords


# first to tokenize the words from the tweets
twToken = nltk.word_tokenize(str(myDF['text']))

# only keep words not in the stopwords set above
for word in twToken:
    if word not in en_stops:
        removedSW.append(word)

# check the first 10 words
print(len(removedSW))
print(removedSW[1:10])
```

After removing the stopwords, I then printed the first 10 entries within the removedSW list. The next 2 steps were to sort the list and have a look at the top 10 entries again before saving the df to a csv again.

```
# here we have removed all the stop words
# now let us look the top 10
removedSW.sort(reverse=True)
print(removedSW[1:10])
# now save to csv
myDF.to_csv('marks-tweets.csv', index=False)
```

This will finally conclude this section, next will be creating a database on DynamoDB and running some HiveQL queries against the data.

2.3 – DynamoDB

For this section we will create a database on DynamoDB which will have the layout for the data we want to keep and query. This was done using a Jupyter Notebook again and using the package boto3. This will allow us to write out our code in a notebook and have it sync up to our AWS account.

I started by importing 2 modules and then creating an instance of my AWS account, this was done with the code below:

```
# import modules
import boto3
import csv
# create client instance
client = boto3.client('dynamodb', region_name='us-east-1')
```

Next was to create the skeleton of our database, seeing as this would be a NoSQL database not every single 'column' has to be designed at this point. Although it is still important to have a unique primary key, which is made up of a key value pair in DynamoDB. To create a table this code was run:

```
def createTable():
    try:
        resp = client.create_table(
            TableName="myTweets",
            # Declare your Primary Key in the KeySchema argument
            KeySchema=[
                {
                    "AttributeName": "created_at",
                    "KeyType": "HASH"
                },
                {
                    "AttributeName": "user_name",
                    "KeyType": "RANGE"
                }
            ],
            # Any attributes used in KeySchema or Indexes must be declared i
n AttributeDefinitions
            AttributeDefinitions=[
                {
                    "AttributeName": "created_at",
                    "AttributeType": "S"
                },
                {
                    "AttributeName": "user_name",
                    "AttributeType": "S"
                }
            ],
            # ProvisionedThroughput controls the amount of data you can read
 or write to DynamoDB per second.
            # You can control read and write capacity independently.
            ProvisionedThroughput={
                "ReadCapacityUnits": 1,
                "WriteCapacityUnits": 1
            }
        )
        print("Table created successfully!")
    except Exception as e:
        print("Error creating table:")
        print(e)
# now function is created run it with:
createTable()
```

This creates a function, which in turn created the table on our DynamoDB account, if there are any errors this is printed out. This piece of code was adapted from AWS documentation. Next is setting up our connection and selecting the DynamoDB we just created, before importing the data.

```
# Now that the table has been created, here we want to import our data.
# create an instance of dynamodb via boto3
dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
# select our new db table
myTable = dynamodb.Table('myTweets')
```

Now to write a script which would take our csv and import it into this newly created table:

```
# create a function to read each row
def batch_write(table_name, rows):
    table = dynamodb.Table(table_name)

    with table.batch_writer() as batch:
        for row in rows:
            batch.put_item(Item=row)
        return True

def read_csv(csv_file, list):
    # get all rows
    rows = csv.DictReader(open(csv_file, errors="ignore"))

    for row in rows:
        list.append(row)

if __name__ == '__main__':
    table_name = 'myTweets'
    file_name = 'marks-tweets.csv'
    items = []

    read_csv(file_name, items)

    status = batch_write(table_name, items)

    if(status):
        print('csv inserted')
    else:
        print('Error inserting csv')
# source : https://www.youtube.com/watch?v=MOaXGYgqipQ
```

To check this data was uploaded correctly, I selected someone from the table.

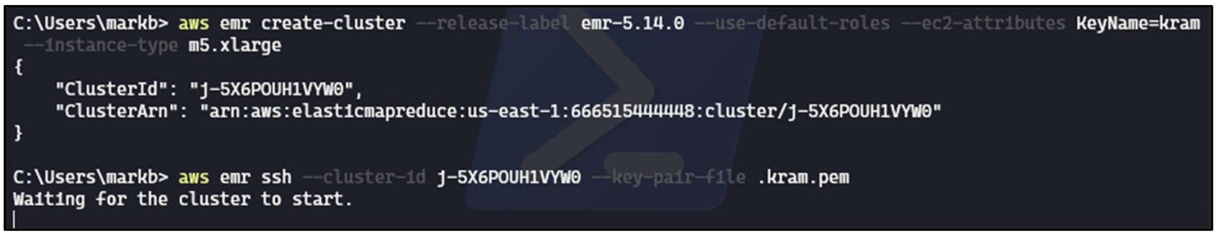
```
# get item
resp = table.get_item(Key={"created_at": "Tue May 18 15:35:01 +0000 2021",
                           "user_name": "Stefan Hawley"})

print(resp['Item'])
```

Whilst I thought it was a good idea to have the created_at and user_name as the key value pair, another 2 columns could have been used for an easy time to use.

This concludes this section of the report, with the next step being HiveQL.

3 – Experiment 3

Before being able to run some HiveQL queries against our data, the first thing to do is to create a EMR Cluster on AWS. This can be done with the following code:



```
C:\Users\markb> aws emr create-cluster --release-label emr-5.14.0 --use-default-roles --ec2-attributes KeyName=kram
 --instance-type m5.xlarge
{
    "ClusterId": "j-5X6POUH1VYW0",
    "ClusterArn": "arn:aws:elasticmapreduce:us-east-1:666515444448:cluster/j-5X6POUH1VYW0"
}

C:\Users\markb> aws emr ssh --cluster-id j-5X6POUH1VYW0 --key-pair-file .kram.pem
Waiting for the cluster to start.
|
```

(Figure 12 – Creating a EMR cluster via CLI)

The figure above shows the command I used to create a cluster, there is also an updated version within the appendix.

This section will go over the HiveQL queries specified in the brief – whilst I was not able to get this far I have included the SQL which I think would have worked if I was able to upload my data to EMR.

3.1 - The country which has authored the most tweets

select * from mytweets

order by sum(tweets)

group by location;

3.2 - The most frequent hashtag/word mentioned in experiment 2 found in tweets from each country

select * from mytweets

where sum(hashtags)

order by hashtags

group by location

3.3 - The most frequent hashtag/word mentioned in experiment found in all tweets

select * from mytweets

where sum(hashtags)

order by hashtags

3.4 - Total number of user mentions in tweets from each country

select * from mytweets

where sum(mentions)

order by mentions

group by location

3.5 - Total number of user mentions in all tweets

select * from mytweets

where sum(mentions)

order by mentions

group by location

4 - Conclusion

This will conclude this report, which was been a very good learning curve into the technology AWS. Whilst I had a lot of issues during this coursework (Issues with VM's, streaming tweets, logging into aws, creating clusters), I do think I have learnt a lot and have made notes of how to replicate this.

If there was not such a short time limit, I think I would have been able to finish this fully but didn't want to get an extension – my favourite part was probably the virtual machines but that is only because I really enjoy working on servers from past experience and have been a great fan of Linux for several years.

Since doing this coursework, I have also had an interview for a data science grad role, who was very interested in the things I have learnt on AWS and asked about if I was able to do the same thing on Google Cloud.

5 – References

Anaconda (2021) *Data science technology for human sensemaking*. Available at: https://www.anaconda.com/ (Accessed 23/04/2021)

Bboxfinder (2021) Available at: http://bboxfinder.com/ (Accessed 27/04/2021)

Boto3 (2021) *Boto3 documentation* Available at: https://boto3.amazonaws.com/v1/documentation/api/latest/index.html (Accessed 27/04/2021)

Jupyter (2021) *Jupyter.* Available at: https://jupyter.org/ (Accessed 23/04/2021)

Microsoft Excel (2021) Available at: https://www.microsoft.com/en-gb/microsoft-365/excel (Accessed 23/04/2021)

NLTK (2021) *Natural Language Toolkit* Available at: https://www.nltk.org/ (Accessed 21/05/2021)

Notepad++ (2021) *What is Notepad++* Available at: https://notepad-plus-plus.org/ (Accessed 23/04/2021)

Tweepy (2021) *Tweepy Documentation*. Available at: https://docs.tweepy.org/en/latest/index.html (Accessed 23/04/2021)

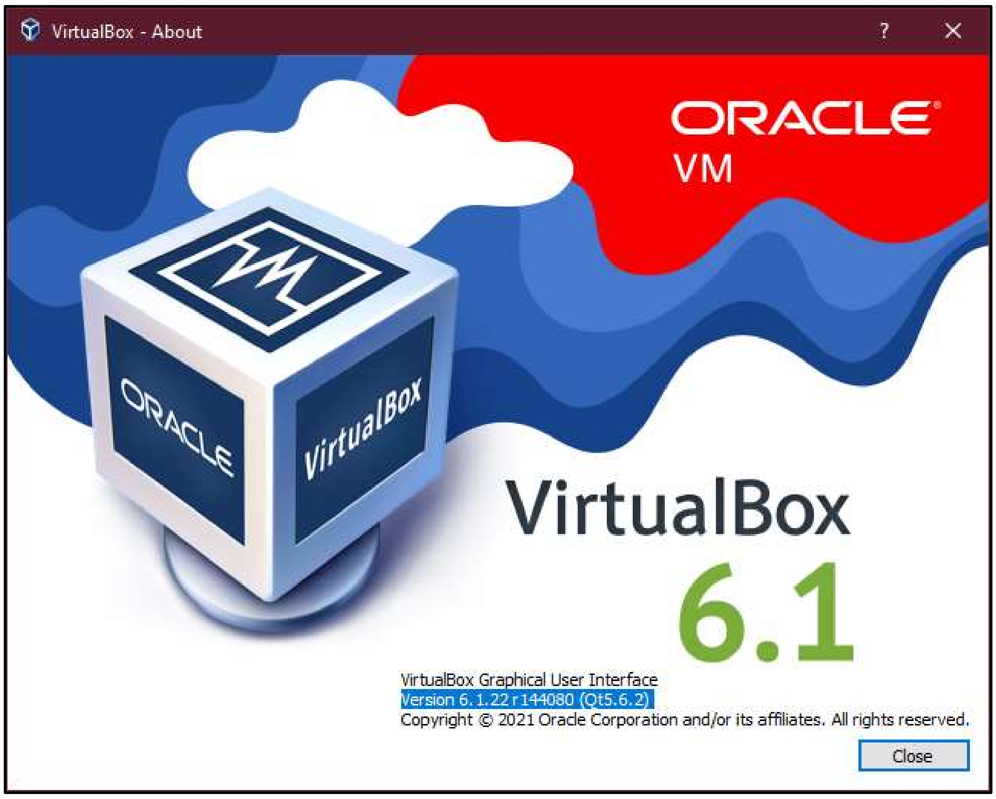
Twitter (2021) *Happening now* – Join Twitter today. Available at: https://twitter.com/ (Accessed 23/04/2021)

Twitter Developer (2021) *Developer Portal*. Available at: https://developer.twitter.com/en/portal/dashboard (Accessed 23/04/2021)

Visual Studio Code (2021) *Redefined.* Available at: https://code.visualstudio.com/ (Accessed 23/04/2021)

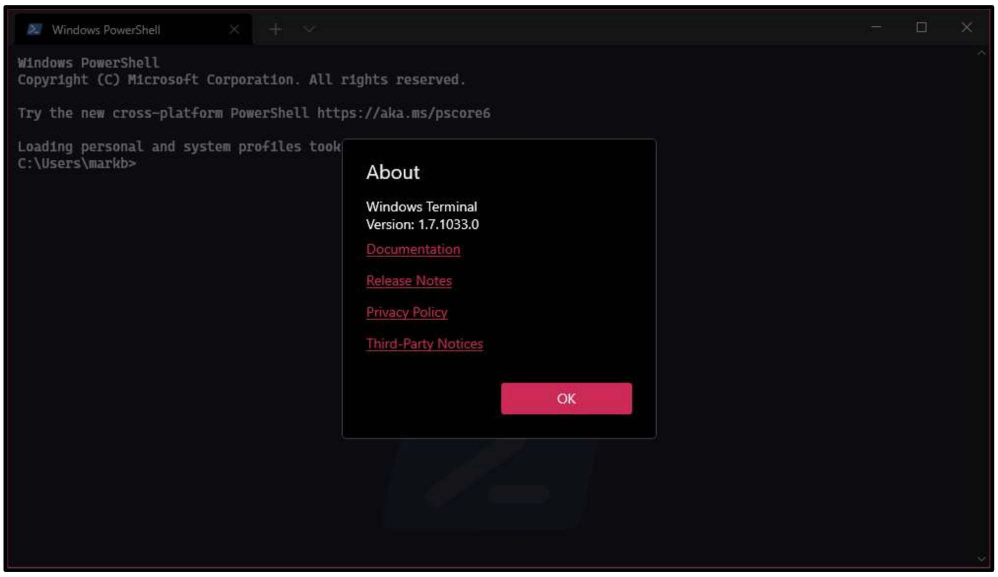
6 - Software

6.1 - VirtualBox – Version 6.1.22 r144080 (Qt5.6.2)



VirtualBox Graphical User Interface
Version 6.1.22 r144080 (Qt5.6.2)
Copyright © 2021 Oracle Corporation and/or its affiliates. All rights reserved.

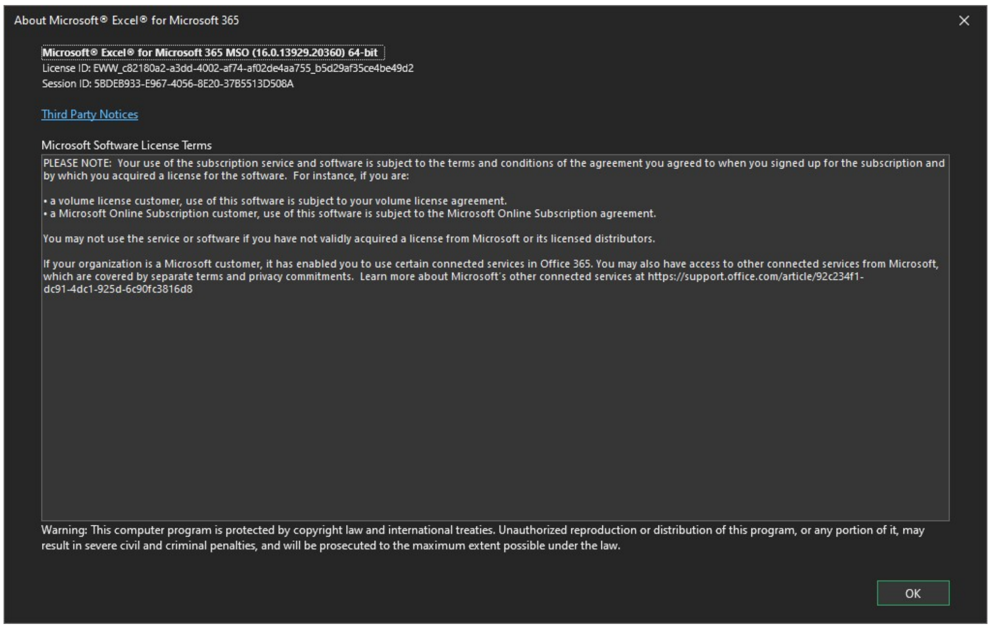
6.2 - Windows Terminal - Version: 1.7.1033.0



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

Loading personal and system profiles took
C:\Users\markb>
```

Windows Terminal
Version: 1.7.1033.0

6.3 – Microsoft Excel – Version: 16.0.13929.20360



About Microsoft® Excel® for Microsoft 365

Microsoft® Excel® for Microsoft 365 MSO (16.0.13929.20360) 64-bit
License ID: EWW_c82180a2-a3dd-4002-af74-af02de4aa755_b5d29af35ce4be49d2
Session ID: 5BDEB933-E967-4056-8E20-37B5513D508A

Third Party Notices

Microsoft Software License Terms

PLEASE NOTE: Your use of the subscription service and software is subject to the terms and conditions of the agreement you agreed to when you signed up for the subscription and by which you acquired a license for the software. For instance, if you are:

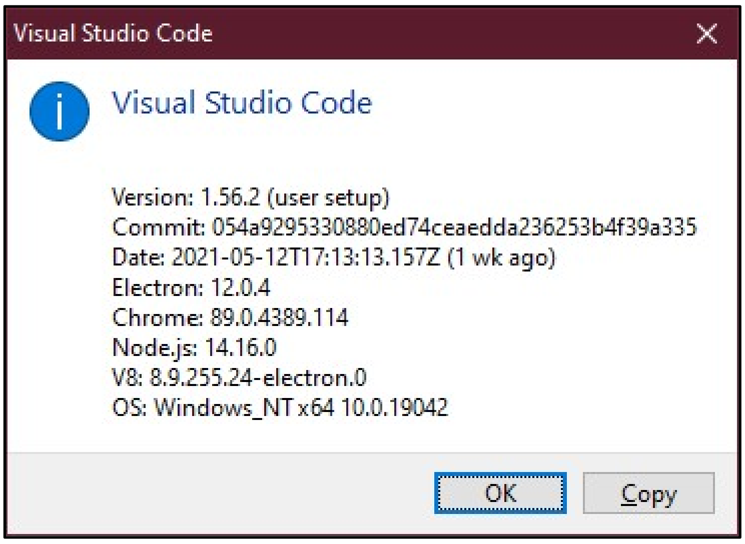
• a volume license customer, use of this software is subject to your volume license agreement.
• a Microsoft Online Subscription customer, use of this software is subject to the Microsoft Online Subscription agreement.

You may not use the service or software if you have not validly acquired a license from Microsoft or its licensed distributors.

If your organization is a Microsoft customer, it has enabled you to use certain connected services in Office 365. You may also have access to other connected services from Microsoft, which are covered by separate terms and privacy commitments. Learn more about Microsoft's other connected services at https://support.office.com/article/92c234f1-dc91-4dc1-925d-6c90fc3816d8

Warning: This computer program is protected by copyright law and international treaties. Unauthorized reproduction or distribution of this program, or any portion of it, may result in severe civil and criminal penalties, and will be prosecuted to the maximum extent possible under the law.

6.4 – Microsoft Visual Studio Code – Version 1.56.2

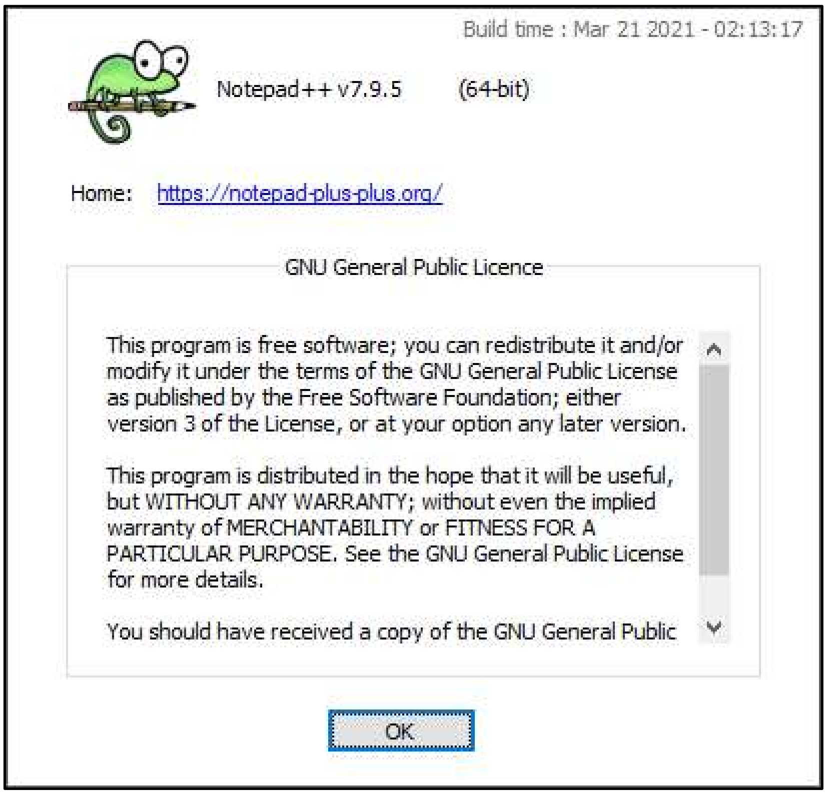


Visual Studio Code

Visual Studio Code

Version: 1.56.2 (user setup)
Commit: 054a9295330880ed74ceaedda236253b4f39a335
Date: 2021-05-12T17:13:13.157Z (1 wk ago)
Electron: 12.0.4
Chrome: 89.0.4389.114
Node.js: 14.16.0
V8: 8.9.255.24-electron.0
OS: Windows_NT x64 10.0.19042

6.5 – Notepad++ - Version 7.9.5



Build time : Mar 21 2021 - 02:13:17

Notepad++ v7.9.5 (64-bit)

Home: https://notepad-plus-plus.org/

GNU General Public Licence

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or at your option any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

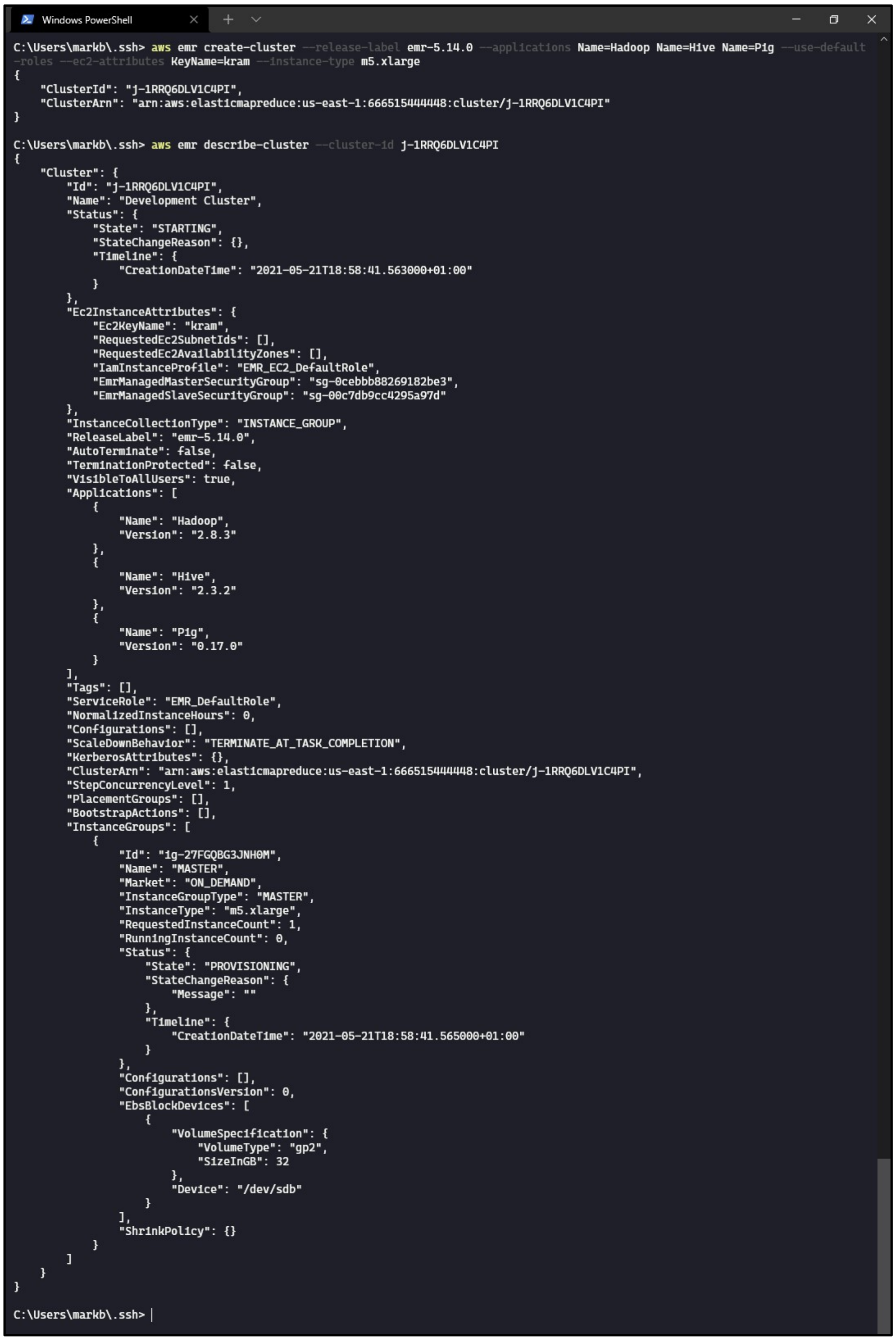
You should have received a copy of the GNU General Public

6.6 – Loop through tweets for X

```
# Now that we have created a way to get the listed entities, lets do that fo
r the rest.
userNames = []
# get userName
i = 0
for user_names in df['user']:
    userNames.append(df['user'][i]['name'])
    i+=1
# add userNames to myDF
myDF['user_name'] = userNames

# create list
userLoc = []
# loop through
i = 0
for user_locations in df['user']:
    userLoc.append(df['user'][i]['location'])
    i+=1
# add userLoc to myDF
myDF['location'] = userLoc

# create list
hashTags = []
# loop through
i = 0
```

```
for user_names in df['user']:
    hashTags.append(df['entities'][i]['hashtags'])
    i+=1
# add userLoc to myDF
myDF['hashtags'] = hashTags

# create list
userMentions = []
# loop through
i = 0
for user_names in df['user']:
    userMentions.append(df['entities'][i]['user_mentions'])
    i+=1
# add userLoc to myDF
myDF['mentions'] = userMentions
```

6.7 – Created and Describe EMR cluster via CLI



```
C:\Users\markb\.ssh> aws emr create-cluster --release-label emr-5.14.0 --applications Name=Hadoop Name=Hive Name=Pig --use-default
-roles --ec2-attributes KeyName=kram --instance-type m5.xlarge
{
    "ClusterId": "j-1RRQ6DLV1C4PI",
    "ClusterArn": "arn:aws:elasticmapreduce:us-east-1:666515444448:cluster/j-1RRQ6DLV1C4PI"
}

C:\Users\markb\.ssh> aws emr describe-cluster --cluster-id j-1RRQ6DLV1C4PI
{
    "Cluster": {
        "Id": "j-1RRQ6DLV1C4PI",
        "Name": "Development Cluster",
        "Status": {
            "State": "STARTING",
            "StateChangeReason": {},
            "Timeline": {
                "CreationDateTime": "2021-05-21T18:58:41.563000+01:00"
            }
        },
        "Ec2InstanceAttributes": {
            "Ec2KeyName": "kram",
            "RequestedEc2SubnetIds": [],
            "RequestedEc2AvailabilityZones": [],
            "IamInstanceProfile": "EMR_EC2_DefaultRole",
            "EmrManagedMasterSecurityGroup": "sg-0cebbb88269182be3",
            "EmrManagedSlaveSecurityGroup": "sg-00c7db9cc4295a97d"
        },
        "InstanceCollectionType": "INSTANCE_GROUP",
        "ReleaseLabel": "emr-5.14.0",
        "AutoTerminate": false,
        "TerminationProtected": false,
        "VisibleToAllUsers": true,
        "Applications": [
            {
                "Name": "Hadoop",
                "Version": "2.8.3"
            },
            {
                "Name": "Hive",
                "Version": "2.3.2"
            },
            {
                "Name": "Pig",
                "Version": "0.17.0"
            }
        ],
        "Tags": [],
        "ServiceRole": "EMR_DefaultRole",
        "NormalizedInstanceHours": 0,
        "Configurations": [],
        "ScaleDownBehavior": "TERMINATE_AT_TASK_COMPLETION",
        "KerberosAttributes": {},
        "ClusterArn": "arn:aws:elasticmapreduce:us-east-1:666515444448:cluster/j-1RRQ6DLV1C4PI",
        "StepConcurrencyLevel": 1,
        "PlacementGroups": [],
        "BootstrapActions": [],
        "InstanceGroups": [
            {
                "Id": "ig-27FGQBG3JNH0M",
                "Name": "MASTER",
                "Market": "ON_DEMAND",
                "InstanceGroupType": "MASTER",
                "InstanceType": "m5.xlarge",
                "RequestedInstanceCount": 1,
                "RunningInstanceCount": 0,
                "Status": {
                    "State": "PROVISIONING",
                    "StateChangeReason": {
                        "Message": ""
                    },
                    "Timeline": {
                        "CreationDateTime": "2021-05-21T18:58:41.565000+01:00"
                    }
                },
                "Configurations": [],
                "ConfigurationsVersion": 0,
                "EbsBlockDevices": [
                    {
                        "VolumeSpecification": {
                            "VolumeType": "gp2",
                            "SizeInGB": 32
                        },
                        "Device": "/dev/sdb"
                    }
                ],
                "ShrinkPolicy": {}
            }
        ]
    }
}

C:\Users\markb\.ssh> |
```