Spinning Up a Free Hadoop Cluster: Step by Step

Want to learn Hadoop and other big data tools from top data engineers in Silicon Valley or New York? *The Insight Data Engineering Fellows Program is free 7-week professional training where you can build cutting edge big data platforms and transition to a career in data engineering at top teams like Facebook, Uber, Slack and Squarespace.*

Learn more about the program and apply today.

. . .

The DevOps series covers how to get started with the leading open source distributed technologies. In the first post of this series we step through how *to spin up a small cluster on Amazon Web Services and deploy a basic Hadoop framework for distributed storage and processing.*

One of the significant hurdles in learning to build distributed systems is understanding how these various technologies are installed and their inter-dependencies. In our experience, the best way to get started with these technologies is to roll up your sleeves and build projects you are passionate about.

The following tutorial shows how you can spin up your own personal cluster on AWS and deploy Hadoop. By no means are these production level setups, but it helps you quickly start interacting with Hadoop's distributed file system and even run MapReduce jobs.

- Spin Up AWS Micro-Instances
- SSH Configuration
- Install Hadoop
- Start Hadoop Cluster
- Working with HDFS

Before we begin, it's important to roughly understand the three components of Apache Hadoop project:

1. Hadoop Distributed File System (HDFS) is a distributed file system based off Google File System (GFS) that splits files into "blocks"

and stores them redundantly on several relatively cheap machines, known as DataNodes. Access to these DataNodes is coordinated by a relatively high-quality machine, known as a NameNode.

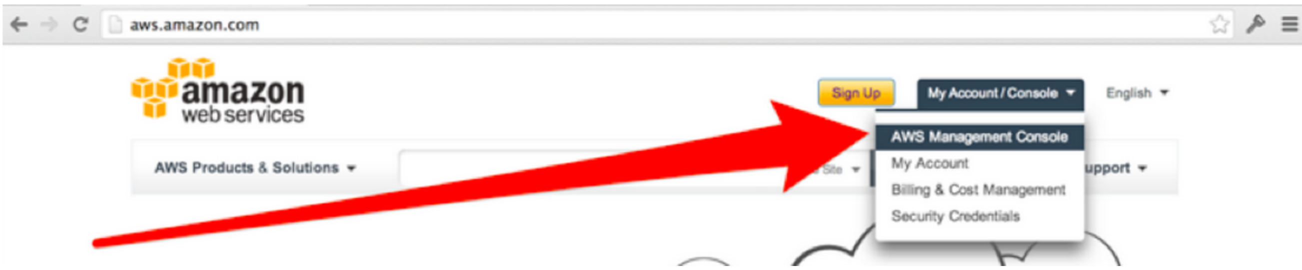
2. Hadoop MapReduce (based off Google MapReduce) is a paradigm for distributed computing that splits computations across several machines. In the Map task, each machine in the cluster calculates a user-defined function on a subset of the input data. The outputted data of the Map task is shuffled around the cluster of machines to be grouped or aggregated in the Reduce task.

3. YARN (unofficially Yet Another Resource Negotiator) is a new feature in Hadoop 2.0 (released in 2013) that manages resources and job scheduling, much like an operating system. This is an important improvement, especially in productions with multiple applications and users, but we won't focus on this for now.

Spin up AWS EC2 Instances

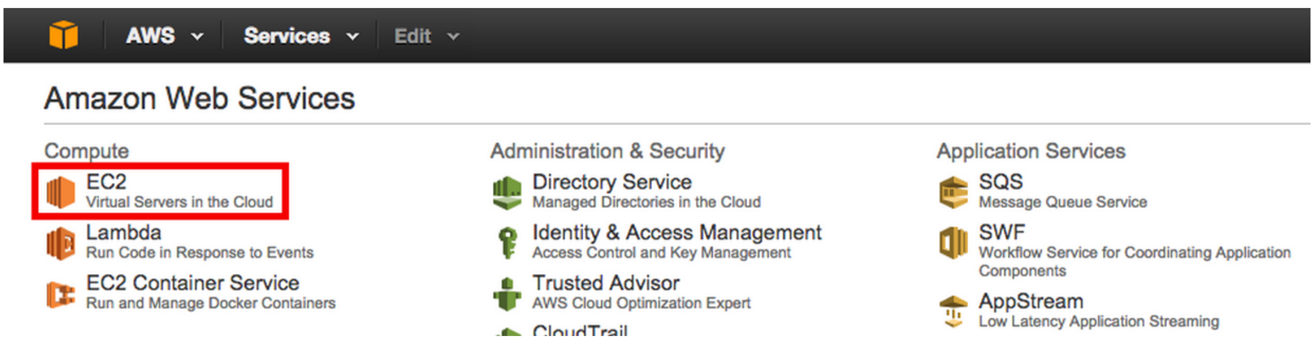
AWS provides on-demand computing resources and services in the cloud, with pay-as-you-go pricing. For example, you can run a server on AWS that you can log on to, configure, secure, and run just as you would a server that's sitting in front of you. As well as selling premium instances, AWS also offers free micro-instances for one year which you can use for this tutorial.

Log into your AWS account under aws.amazon.com. If you currently don't have one, go ahead and sign up for one.

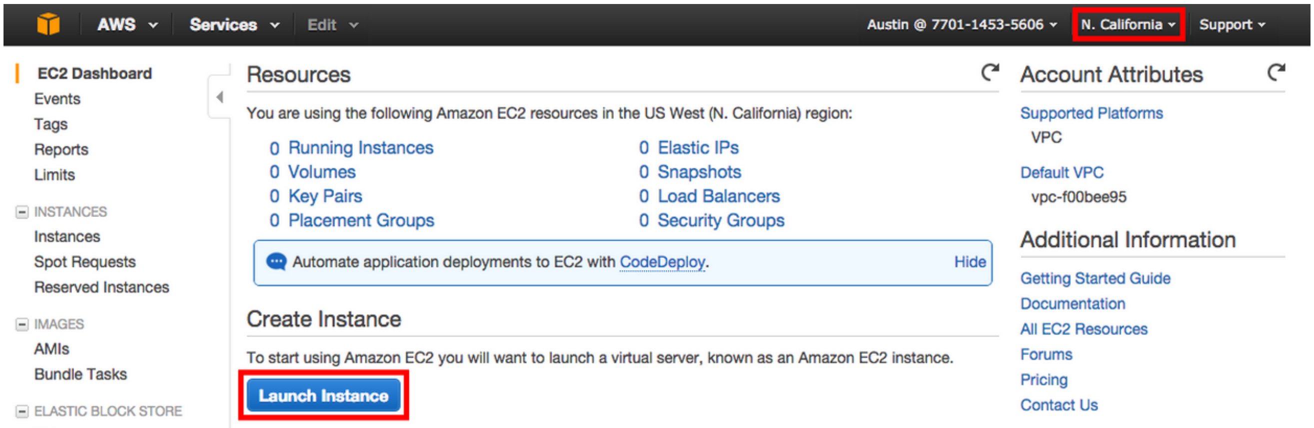


Launching Multiple AWS Micro-Instances

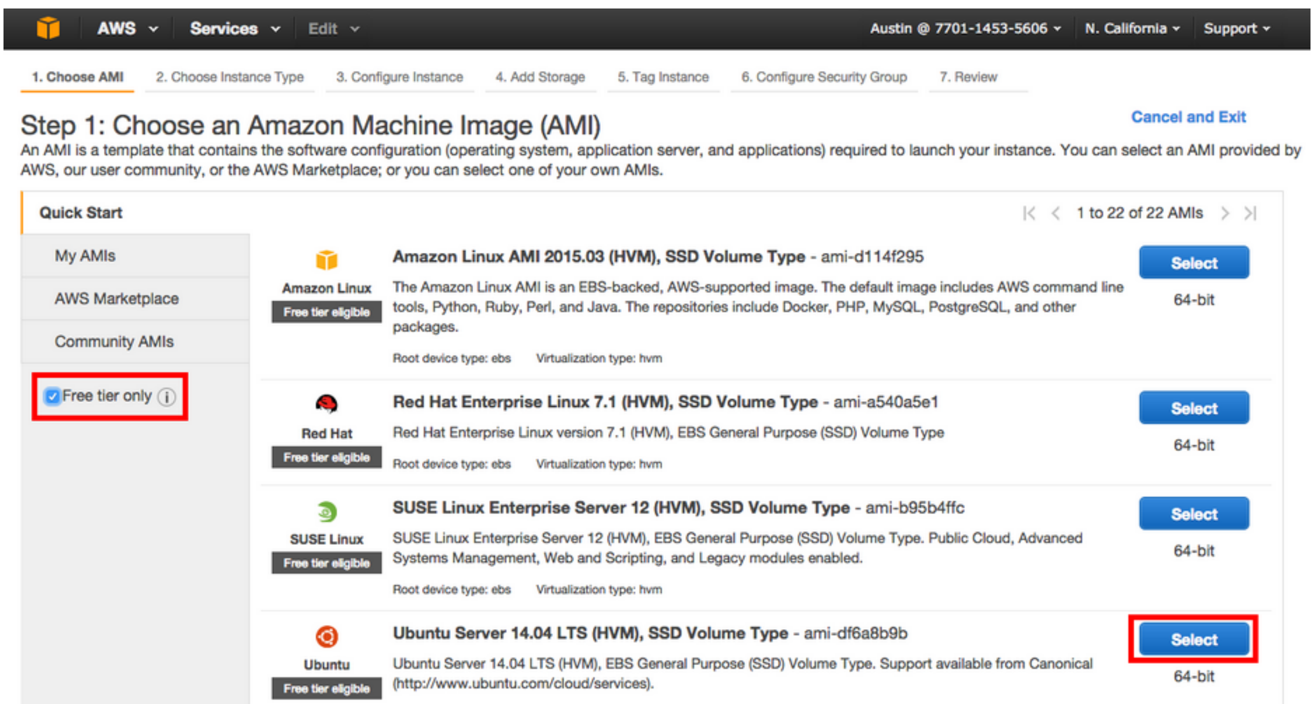
You can do a lot of different things with AWS. For now, let's just start a virtual machine. Click on EC2 ("Elastic Cloud Compute").



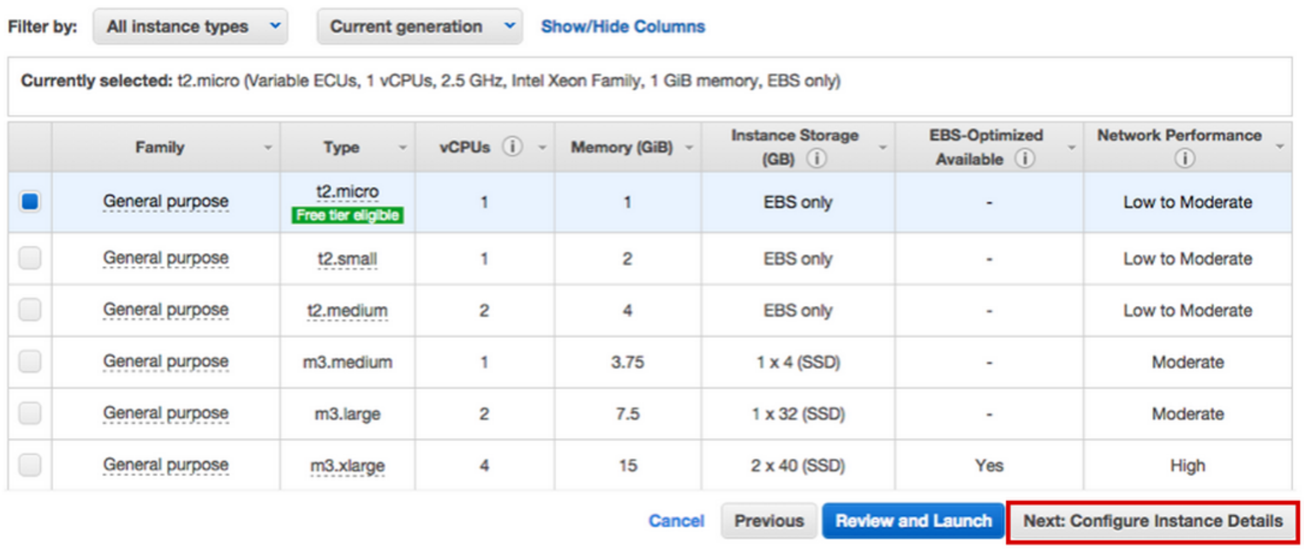
Click on "Launch Instance". It's worth noting that AWS has different regions, and that you can launch an instance in any of them. (So if you've always wanted a server in Brazil or Japan, this is your chance!) Your choice will affect both latency and price—by default, it's best to go with the region that's closest to you.



Now it’s time to select the image for your VM. For this tutorial, we’ll use Ubuntu Server 14.04 LTS (HVM). Be sure to select the 64-bit.

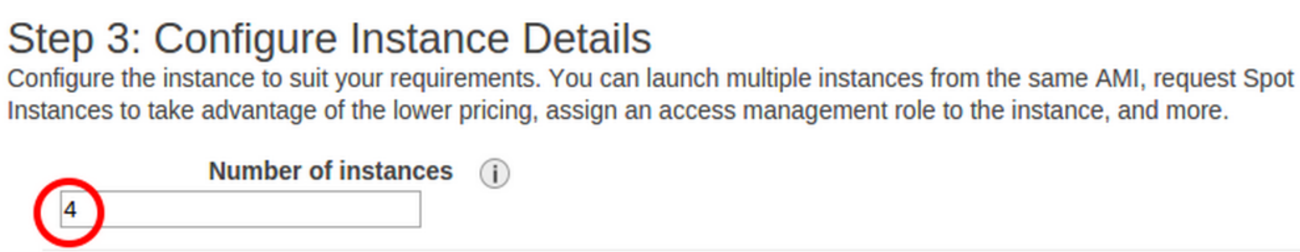


Now choose your instance size. t2.micro instances should be sufficient to start playing with HDFS and MapReduce. Be sure to terminate the instances if you want AWS to keep more free credits for the month. For practice you can try spinning up 4 nodes, treating one as the NameNode(master) and the remaining three as DataNodes.

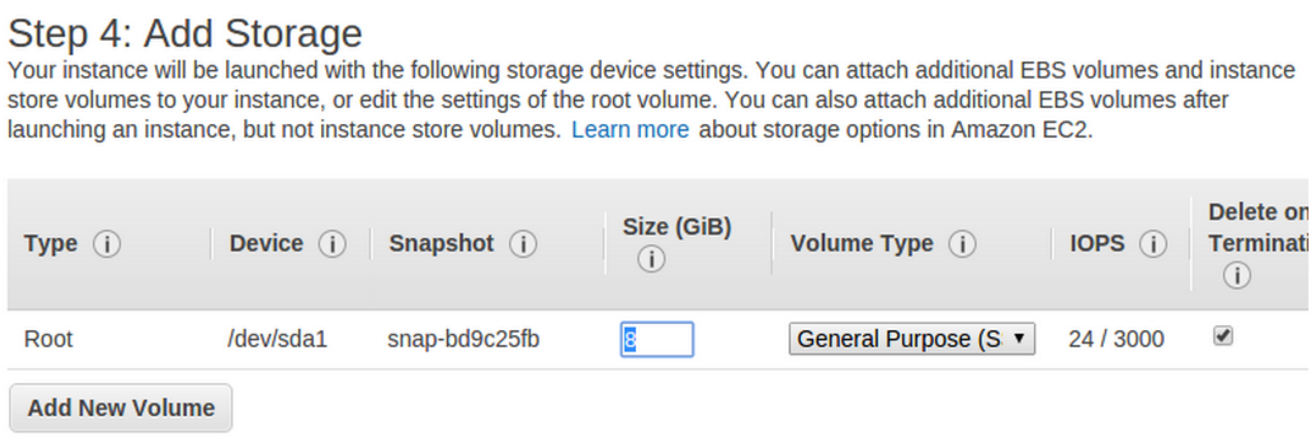


	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
■	General purpose	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate
□	General purpose	t2.small	1	2	EBS only	-	Low to Moderate
□	General purpose	t2.medium	2	4	EBS only	-	Low to Moderate
□	General purpose	m3.medium	1	3.75	1 x 4 (SSD)	-	Moderate
□	General purpose	m3.large	2	7.5	1 x 32 (SSD)	-	Moderate
□	General purpose	m3.xlarge	4	15	2 x 40 (SSD)	Yes	High

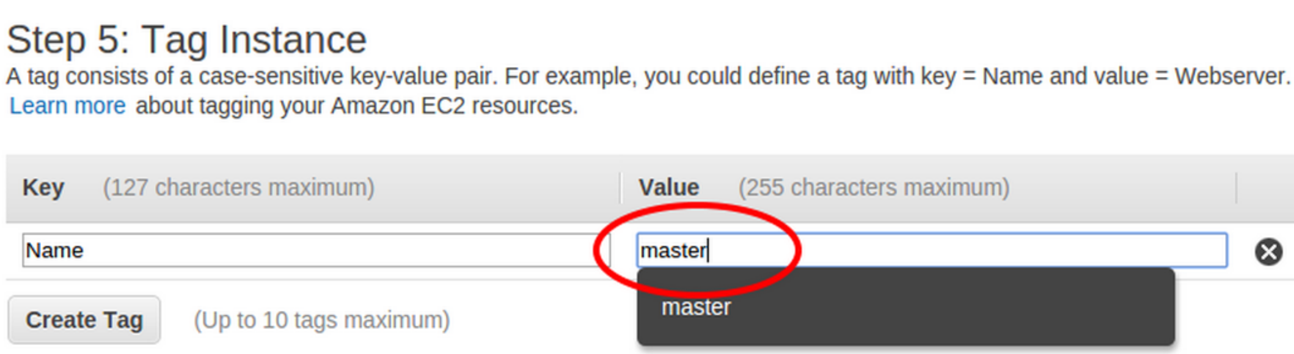
We can then specify the number of instances to spin up. In this tutorial we will spin up 4 instances where 1 node will act as the NameNode and the remaining 3 will be DataNodes.



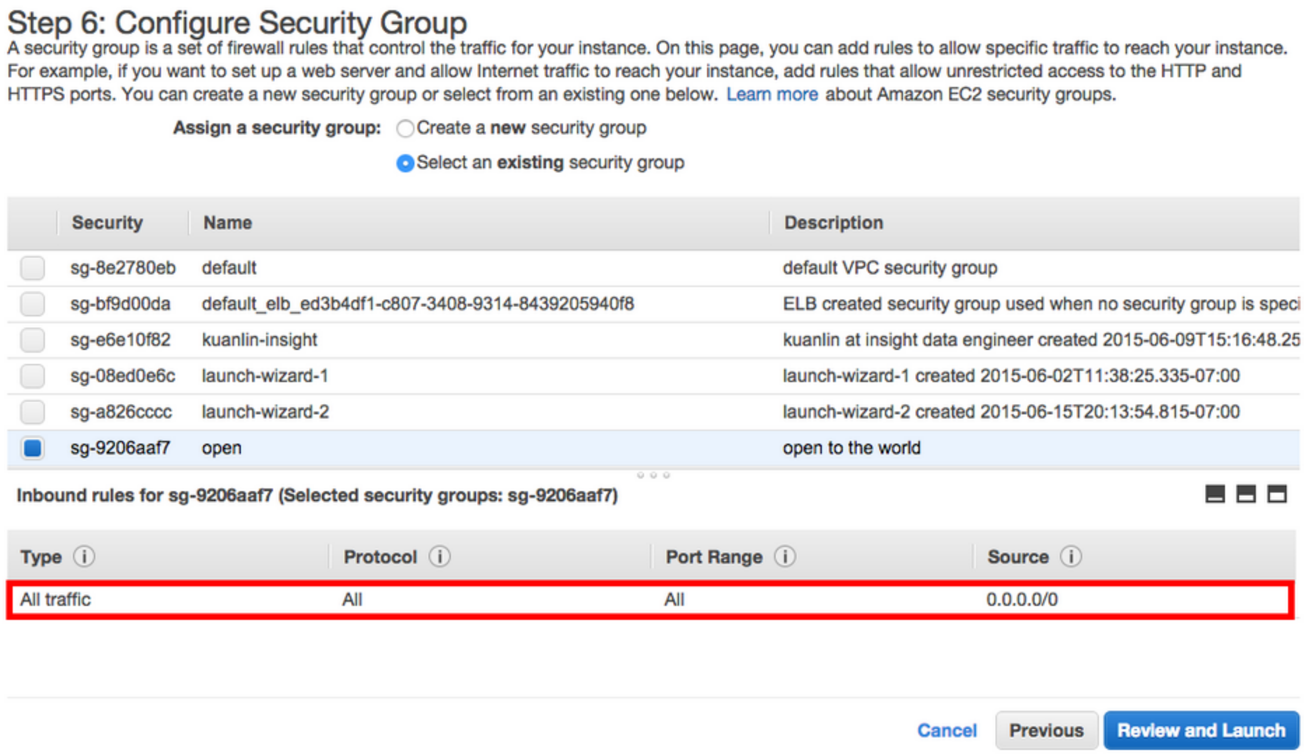
We will use the default storage size for each instance. In the future, if you plan on storing larger amounts of data, here is the place to change it before launching the instances.



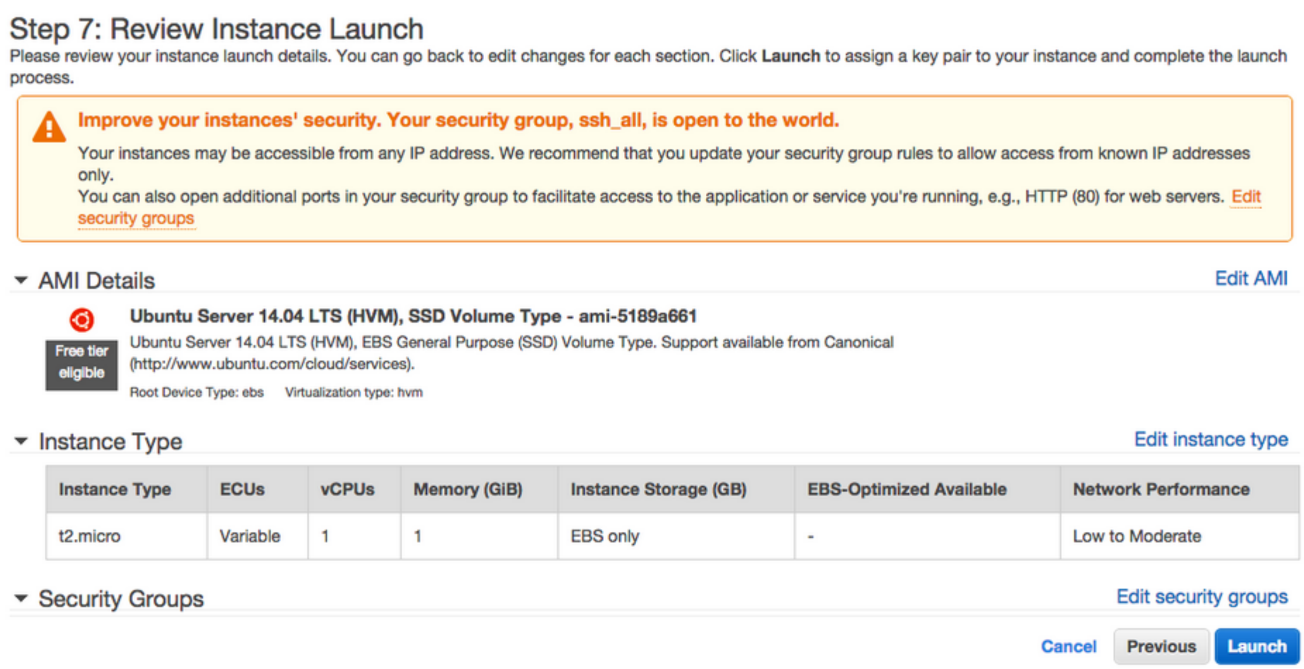
Next we can give these instances a name so they are easily recognizable among other potential instances in your account. Here we gave all the instances the name 'master'. This can be modified as well after you have launched your instances.



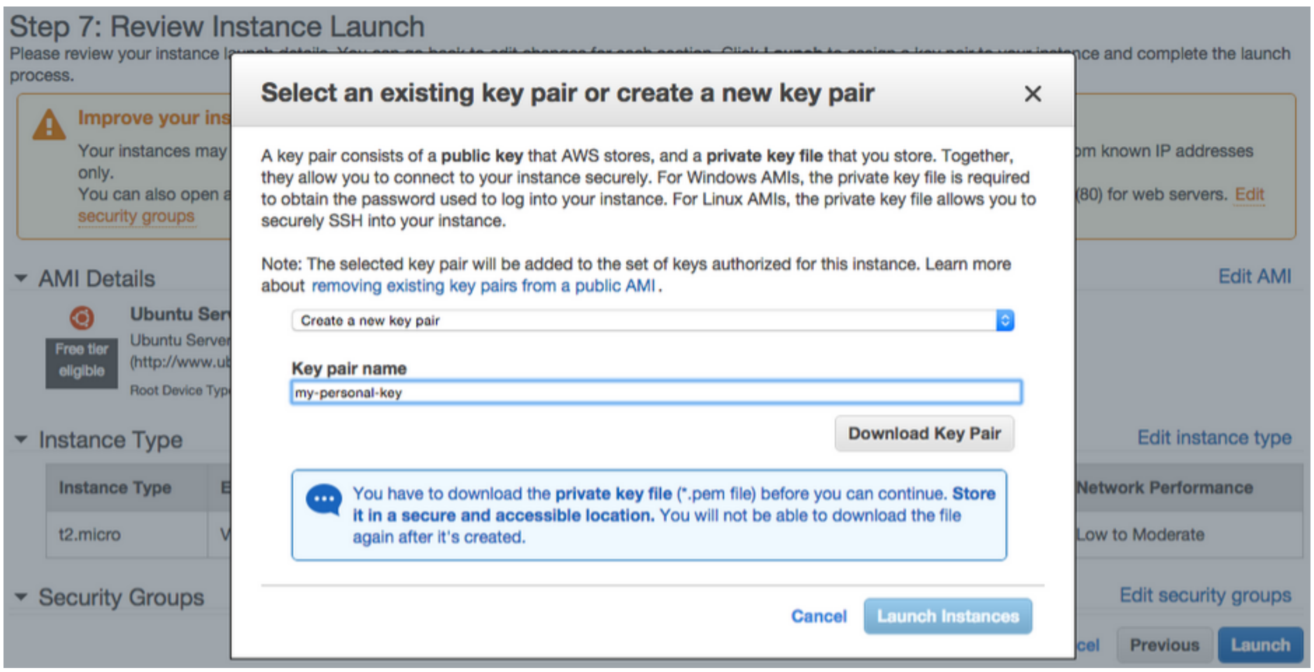
The next step will be to configure the security groups setting for these instances. For this exercise, all the ports are open for ease of access. It should be noted that these settings should be much more strict if put in production. If a security group does not exist with the following configuration, you can create a new security group with the following settings.



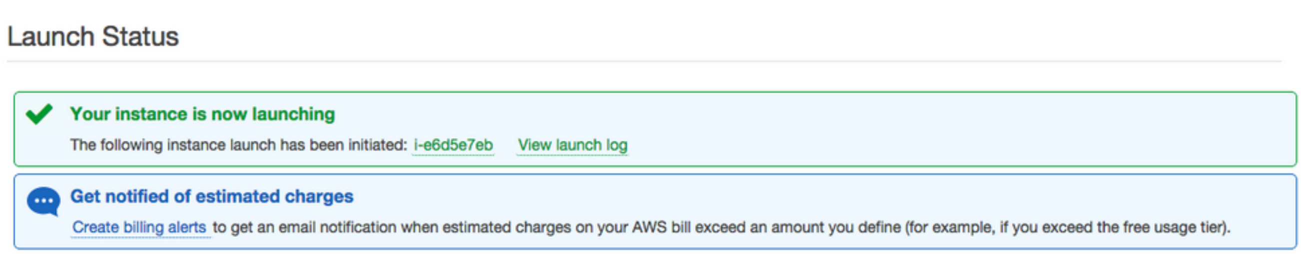
We will then review our instances and launch.



You will then be asked to choose a pem-key which will be used to login to these instances. If this is your first time, you can generate a new pem-key and download it to you computer. For this tutorial we will assume you have saved the pem key to the ~/.ssh folder. **WARNING: If you lose your pem-key there is no way to recover it and thus lose access to any instances that are associated with this pem-key**.



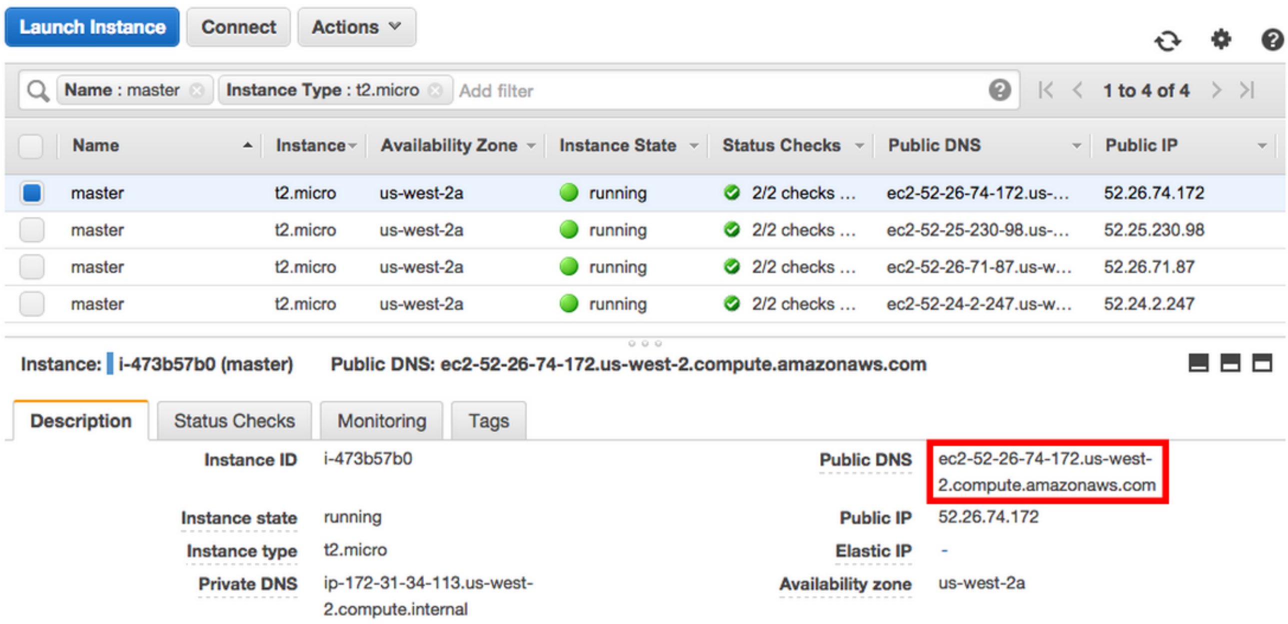
Congratulations! Your AWS instance is now spinning up! Let's log into it.



Logging into an EC2 Instance

Return to the AWS console. You should now see “4 Running Instances” if you chose 4 instances in the beginning. Click on it.

You can then choose one of these instances. You can rename the instances by clicking the pencil to the right of the instance name, which will be helpful for technologies that use a single “master” that needs to be distinguished from than the rest. You should now be able to see the public IP address of your Virtual Machine (VM). This is the endpoint we’re going to use to SSH into the machine.



For the remainder of this tutorial, we will be referencing the Public DNS quite a bit when installing Hadoop. In any code snippets where there is **bolded text**, you will need to swap it out with appropriate values.

For example, namenode_public_dns should be replaced with something like ec2–52–26–74–172.us-west-2.compute.amazonaws.com

```
e.g:
your Namenode public DNS is ec2–52–26–74–172.us-west-
2.compute.amazonaws.com

local$ ssh -i ~/.ssh/my-pem-key.pem
ubuntu@namenode_public_dns
```

```
should translate to

local$ ssh -i ~/.ssh/my-pem-key.pem ubuntu@ec2-52-26-74-172.us-west-2.compute.amazonaws.com
```

Fellows in the past have found it useful to simply copy all the Public DNSs into a text file for easy reference later on. At this point you should decide which Public DNS will be the NameNode and the remaining Public DNSs be the DataNodes.

Example text file:

```
namenode_public_dns => ec2-52-26-74-172.us-west-2.compute.amazonaws.com
datanode1_public_dns => ec2-52-26-34-168.us-west-2.compute.amazonaws.com
datanode2_public_dns => ec2-52-26-27-192.us-west-2.compute.amazonaws.com
datanode3_public_dns => ec2-52-26-36-112.us-west-2.compute.amazonaws.com
```

You need to change the permissions of the pem key that you downloaded from AWS earlier. Do this with the command otherwise SSH'ing will complain that the key is too open (bolded code would be different for your setup):

```
local$ sudo chmod 600 ~/.ssh/pem_key_filename
```

Now let's SSH into the machine with the following SSH command template:

```
local$ ssh -i ~/.ssh/pem_key_filename
ubuntu@namenode_public_dns
```

If prompted "Are you sure that you want to continue?", enter "yes". After verifying that you can SSH into a node, you can exit with the command exit or Ctrl-D.

SSH Configuration

We can make our lives easier by setting up a config file in the ~/.ssh directory such that we do not have to specify the pem key and host address every time we SSH into a node from our local machine. Simply place the following into the ~/.ssh/config file. Create one, if it does not exist.

~/.ssh/config:

```
Host namenode
  HostName namenode_public_dns
  User ubuntu
  IdentityFile ~/.ssh/pem_key_filename

Host datanode1
  HostName datanode1_public_dns
  User ubuntu
  IdentityFile ~/.ssh/pem_key_filename

Host datanode2
  HostName datanode2_public_dns
  User ubuntu
  IdentityFile ~/.ssh/pem_key_filename

Host datanode3
  HostName datanode3_public_dns
  User ubuntu
  IdentityFile ~/.ssh/pem_key_filename
```

Passwordless SSH

The NameNode in the Hadoop cluster needs to be able to communicate with the other DataNodes in the cluster. This is done by configuring passwordless SSH between the NameNode and the DataNodes.

We will first need to transfer your pem key from your local computer to the NameNode. This allows us to initially SSH into DataNodes from the NameNode. We will also copy over the config file in our ~/.ssh folder.

```
local$ scp ~/.ssh/pem_key_filename ~/.ssh/config
namenode:~/.ssh
```

Next we will SSH into the NameNode and create an authorization key. This fingerprint will then be added to the authorized_keys file on the NameNode and all the DataNodes. The first time you SSH into the node it will probably ask you if you are sure you want to connect. Answer yes. For example you may see something like this:

```
local$ ssh namenode
```

```
The authenticity of host ‘ec2–52–26–234–93.us-west-
2.compute.amazonaws.com (172.31.35.245)’ can’t be
established. ECDSA key fingerprint is
df:2f:34:e7:2d:51:7b:b2:38:86:b8:f0:c6:c2:8d:0b.
Are you sure you want to continue connecting (yes/no)? yes
```

On the NameNode we can create the public fingerprint, found in ~/.ssh/id_rsa.pub, and add it first to the NameNode’s authorized_keys

```
namenode$ ssh-keygen -f ~/.ssh/id_rsa -t rsa -P ""
namenode$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

Now we need to copy the public fingerprint to each DataNode's~/.ssh/authorized_keys. This should enable the password-less SSH capabilities from the NameNode to any DataNode.

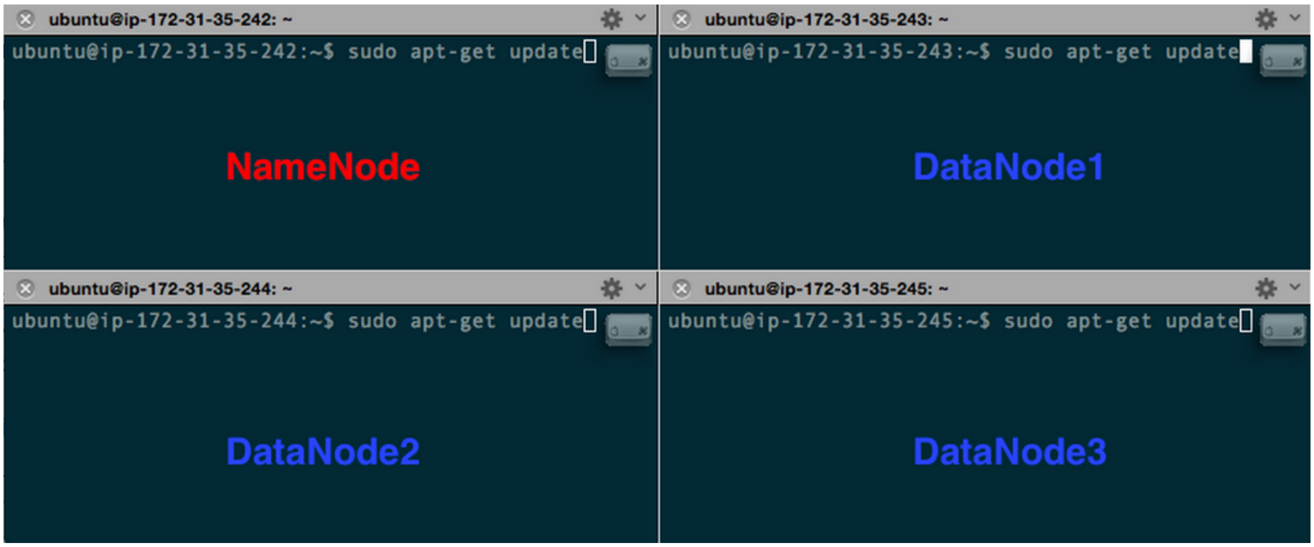
```
namenode$ cat ~/.ssh/id_rsa.pub | ssh datanode1 'cat >>
~/.ssh/authorized_keys'
namenode$ cat ~/.ssh/id_rsa.pub | ssh datanode2 'cat >>
~/.ssh/authorized_keys'
namenode$ cat ~/.ssh/id_rsa.pub | ssh datanode3 'cat >>
~/.ssh/authorized_keys'
```

We can check this by trying to SSH into any of the DataNodes from the NameNode. You may still be prompted if you are sure you want to connect, but there should be no password requirement.

```
namenode$ ssh ubuntu@datanode1_public_dns
```

Install Hadoop

Now that password-less SSH is setup, we can begin installation of Hadoop and modify common configurations across the NameNode and DataNodes. Typically this is easier to do when you have 4 terminals open with each terminal representing a node. Below shows an example. If you're using iTerm2 or Terminator, toggling the broadcast input to all panes can help reduce mistakes during installation.



On fresh AWS instances, Java is not installed. We will be installing the openjdk-7-jdk package to be used by Hadoop.

```
allnodes$ sudo apt-get update
allnodes$ sudo apt-get install openjdk-7-jdk
```

We can test to see if Java installed correctly with the following command

```
allnodes$ java -version
```

```
java version "1.7.0_79"
OpenJDK Runtime Environment (IcedTea 2.5.5) (7u79–2.5.5–
0ubuntu0.14.04.2)
OpenJDK 64–Bit Server VM (build 24.79–b02, mixed mode)
```

Next we'll install Hadoop onto all the nodes by first saving the binary tar files to ~/Downloads and extracting it to the /usr/local folder.

```
allnodes$ wget
http://apache.mirrors.tds.net/hadoop/common/hadoop-
2.7.1/hadoop-2.7.1.tar.gz -P ~/Downloads
allnodes$ sudo tar zxvf ~/Downloads/hadoop-* -C /usr/local
allnodes$ sudo mv /usr/local/hadoop-* /usr/local/hadoop
```

Environment Variables

Now we'll need to add some Hadoop and Java environment variables to ~/.profile and source them to the current shell session.

~/.profile:

```
export JAVA_HOME=/usr
export PATH=$PATH:$JAVA_HOME/bin

export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
```

```
export HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop
```

Then load these environment variables by sourcing the profile

```
allnodes$ . ~/.profile
```

Hadoop Configurations

For a basic setup of Hadoop, we'll be changing a few of the configurations in the Hadoop directory defined now by HADOOP_CONF_DIR environment variable. All the current configuration changes will be applied to the NameNode and all the DataNodes. After these changes, we will apply configurations specific to the NameNode and DataNodes.

Here are the following files to focus on:

- $HADOOP_CONF_DIR/hadoop-env.sh
- $HADOOP_CONF_DIR/core-site.xml
- $HADOOP_CONF_DIR/yarn-site.xml
- $HADOOP_CONF_DIR/mapred-site.xml (This file currently does not exist in the default Hadoop installation, but a template is

available. We'll make a copy of the template and rename it to mapred-site.xml)

Common Hadoop Configurations on all Nodes

Let's start with $HADOOP_CONF_DIR/hadoop-env.sh. Currently only root users can edit files in the Hadoop directory, but we'll change this after all configurations have been applied. To edit the configurations files, you can simply add a sudo before the text editor of your choice, for example

```
allnodes$ sudo vim $HADOOP_CONF_DIR/hadoop-env.sh
```

The only thing that needs changing is the location of JAVA_HOME in the file. Simply replace${JAVA_HOME} with /usr which is where Java was just previously installed.

$HADOOP_CONF_DIR/hadoop-env.sh:

```
# The java implementation to use.
export JAVA_HOME=/usr
```

The next file to modify is the $HADOOP_CONF_DIR/core-site.xml. Here we will declare the default Hadoop file system. The default configuration is set to the localhost, but here we will want to specify the NameNode's public DNS on port 9000. Scroll down in the xml file to find the configurations tag and be sure to change the file to look like the following

$HADOOP_CONF_DIR/core-site.xml:

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://namenode_public_dns:9000</value>
  </property>
</configuration>
```

The next file to modify is the $HADOOP_CONF_DIR/yarn-site.xml. Scroll down in the xml file to find the configurations tag and be sure to change the file to look like the following

$HADOOP_CONF_DIR/yarn-site.xml:

```
<configuration>

<! — Site specific YARN configuration properties →
```

```
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
  <property>
    <name>yarn.nodemanager.aux-
services.mapreduce.shuffle.class</name>
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>
  </property>
  <property>
    <name>yarn.resourcemanager.hostname</name>
    <value>namenode_public_dns</value>
  </property>
</configuration>
```

The last configuration file to change is the $HADOOP_CONF_DIR/mapred-site.xml. We will first need to make a copy of the template and rename it.

```
allnodes$ sudo cp $HADOOP_CONF_DIR/mapred-site.xml.template
$HADOOP_CONF_DIR/mapred-site.xml
```

Scroll down in the xml file to find the configurations tag and be sure to change the file to look like the following

$HADOOP_CONF_DIR/mapred-site.xml:

```
<configuration>
  <property>
    <name>mapreduce.jobtracker.address</name>
    <value>namenode_public_dns:54311</value>
  </property>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

NameNode Specific Configurations

Now that all the common configurations are complete, we'll finish up the NameNode specific configurations. On the NameNode, all that remains are the following:

- adding hosts to /etc/hosts
- modifying the configurations in $HADOOP_CONF_DIR/hdfs-site.xml
- defining the Hadoop master in $HADOOP_CONF_DIR/masters
- defining the Hadoop slaves in $HADOOP_CONF_DIR/slaves

Let's start with adding to the hosts file located under /etc/hosts. We will need to add each node's public DNS and hostname to the list. The hostname can be found with the following

```
allnodes$ echo $(hostname)
```

or by taking the first part of the private DNS (e.g. ***ip-172–31–35–242***.us-west-2.compute.internal)

By default, 127.0.0.1 localhost is present, so we can add under it to look like the following (ignoring the IPv6 settings):

/etc/hosts:

```
127.0.0.1 localhost
namenode_public_dns namenode_hostname
datanode1_public_dns datanode1_hostname
datanode2_public_dns datanode2_hostname
datanode3_public_dns datanode3_hostname

# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
```

We can now modify the $HADOOP_CONF_DIR/hdfs-site.xml file to specify the replication factor along with where the NameNode data will reside. For this setup, we will specify a replication factor of 3 for each data block in HDFS.

Scroll down in the xml file to find the configurations tag and be sure to change the file to look like the following

$HADOOP_CONF_DIR/hdfs-site.xml:

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>3</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>

<value>file:///usr/local/hadoop/hadoop_data/hdfs/namenode</v
alue>
  </property>
</configuration>
```

The current path where data on the NameNode will reside does not exist, so we'll need to make this before starting HDFS.

```
namenode$ sudo mkdir -p
$HADOOP_HOME/hadoop_data/hdfs/namenode
```

Next we'll need to add a masters file to the $HADOOP_CONF_DIR directory

```
namenode$ sudo touch $HADOOP_CONF_DIR/masters
```

then insert the NameNode's hostname in that file

$HADOOP_CONF_DIR/masters:

```
namenode_hostname
```

We will also need to modify the slaves file in the $HADOOP_CONF_DIR directory to the following. By default localhost is present, but we can remove this.

$HADOOP_CONF_DIR/slaves

```
datanode1_hostname
datanode2_hostname
datanode3_hostname
```

Now that all configurations are set on the NameNode, we will change the ownership of the$HADOOP_HOME directory to the user ubuntu

```
namenode$ sudo chown -R ubuntu $HADOOP_HOME
```

DataNode Specific Configurations

Let's now move onto the final configurations for the DataNodes. We will need to first SSH into each DataNode and only configure the $HADOOP_CONF_DIR/hdfs-site.xml file

Scroll down in the xml file to find the configurations tag and be sure to change the file to look like the following

$HADOOP_CONF_DIR/hdfs-site.xml:

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>3</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>

<value>file:///usr/local/hadoop/hadoop_data/hdfs/datanode</v
alue>
  </property>
</configuration>
```

Just like on the NameNode, we will need to create the directory specified in the$HADOOP_CONF_DIR/hdfs-site.xml file.

```
datanodes$ sudo mkdir -p
$HADOOP_HOME/hadoop_data/hdfs/datanode
```

Now that all configurations are set on the DataNode, we will change the ownership of the$HADOOP_HOME directory to the ubuntu user

```
datanodes$ sudo chown -R ubuntu $HADOOP_HOME
```

Start Hadoop Cluster

We can now start up HDFS from the Namenode by first formatting it and then starting HDFS. An important thing to notes is that every time the NameNode is formatted, all of the data previously on it is lost.

```
namenode$ hdfs namenode -format
namenode$ $HADOOP_HOME/sbin/start-dfs.sh
```

When asked "The authenticity of host 'Some Node' can't be established. Are you sure you want to continue connecting (yes/no)?" type yes and press enter. **You may need to do this several times—keep typing yes, then enter, even if there is no new prompt**, since it's the first time for Hadoop to log into each of the Datanodes.

You can go to **namenode_public_dns**:50070 in your browser to check if all Datanodes are online. If the webUI does not display, check to make sure your EC2 instances have security group settings that include All Traffic and not just SSH. You should see 3 live nodes, otherwise there was an error in the previous steps.

Summary

Security is off.

Safemode is off.

1 files and directories, 0 blocks = 1 total filesystem object(s).

Heap Memory used 58.57 MB of 186 MB Heap Memory. Max Heap Memory is 889 MB.

Non Heap Memory used 30.46 MB of 31.94 MB Commited Non Heap Memory. Max Non Heap Memory is 214 MB.

Configured Capacity:	23.22 GB
DFS Used:	72 KB
Non DFS Used:	6.17 GB
DFS Remaining:	17.05 GB
DFS Used%:	0%
DFS Remaining%:	73.44%
Block Pool Used:	72 KB
Block Pool Used%:	0%
DataNodes usages% (Min/Median/Max/stdDev):	0.00% / 0.00% / 0.00% / 0.00%
Live Nodes	3 (Decommissioned: 0)
Dead Nodes	0 (Decommissioned: 0)
Decommissioning Nodes	0

Now let's start up YARN as well as the MapReduce JobHistory Server.

```
namenode$ $HADOOP_HOME/sbin/start-yarn.sh
namenode$ $HADOOP_HOME/sbin/mr-jobhistory-daemon.sh start
historyserver
```

You can check to make sure all Java processes are running with the jps command on the NameNode and DataNodes (your process ids will be different though).

```
namenode$ jps
21817 JobHistoryServer
21853 Jps
21376 SecondaryNameNode
21540 ResourceManager
21157 NameNode
```

```
datanodes$ jps
20936 NodeManager
20792 DataNode
21036 Jps
```

Working with HDFS

You're now ready to start working with HDFS by SSH'ing to the NameNode. The most common commands are very similar to normal Linux File System commands, except that they are preceded by hdfs dfs. Below are some common commands and a few examples to get used to HDFS.

Common HDFS Commands

List all files and folder in directory

- hdfs dfs -ls <folder name>

Make a directory on HDFS

- hdfs dfs -mkdir <folder name>

Copy a file from the local machine (namenode) into HDFS

- hdfs dfs -copyFromLocal <local folder or file name>

Delete a file on HDFS

- hdfs dfs -rm <file name>

Delete a directory on HDFS

- hdfs dfs -rmdir <folder name>

HDFS Examples

```
# create local dummy file to place on HDFS
namenode$ echo "Hello this will be my first distributed and
fault-tolerant data set\!" | cat >> my_file.txt

# list directories from top level of HDFS
namenode$ hdfs dfs -ls /
# This should display nothing but a temp directory

# create /user directory on HDFS
namenode$ hdfs dfs -mkdir /user
namenode$ hdfs dfs -ls /
Found 1 items
drwxr-xr-x - ubuntu supergroup 0 2015-05-06 22:41 /user

# copy local file a few times onto HDFS
namenode$ hdfs dfs -copyFromLocal ~/my_file.txt /user
```

```
namenode$ hdfs dfs -copyFromLocal ~/my_file.txt
/user/my_file2.txt
namenode$ hdfs dfs -copyFromLocal ~/my_file.txt
/user/my_file3.txt
```

```
# list files in /user directory
namenode$ hdfs dfs -ls /user
Found 1 items
-rw-r — r — 3 ubuntu supergroup 50 2015–05–06 22:43
/user/my_file.txt
-rw-r — r — 3 ubuntu supergroup 50 2015–05–06 22:43
/user/my_file2.txt
-rw-r — r — 3 ubuntu supergroup 50 2015–05–06 22:43
/user/my_file3.txt
```

```
# clear all data and folders on HDFS
namenode$ hdfs dfs -rm /user/my_file*
15/05/06 22:49:06 INFO fs.TrashPolicyDefault: Namenode trash
configuration: Deletion interval = 0 minutes, Emptier
interval = 0 minutes.
Deleted /user/my_file.txt
15/05/06 22:49:06 INFO fs.TrashPolicyDefault: Namenode trash
configuration: Deletion interval = 0 minutes, Emptier
interval = 0 minutes.
Deleted /user/my_file2.txt
15/05/06 22:49:06 INFO fs.TrashPolicyDefault: Namenode trash
configuration: Deletion interval = 0 minutes, Emptier
interval = 0 minutes.
Deleted /user/my_file3.txt
namenode$ hdfs dfs -rmdir /user
```

What Next?

Now that you have installed Hadoop, you can start running example MapReduce jobs found in the $HADOOP_HOME/share/hadoop/mapreduce/ folder. The jar file is named hadoop-mapreduce-examples-*.jar and further documentation on using the jar file can be found here, Hadoop MapReduce Examples.

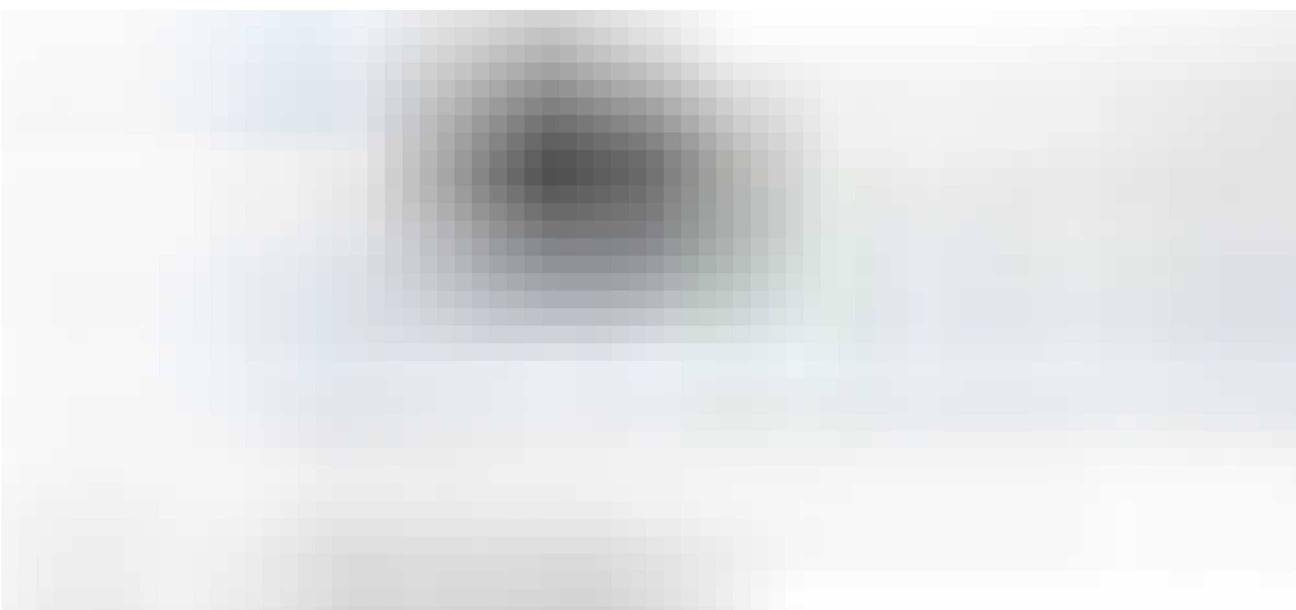
If writing MapReduce in Java isn't your cup of tea, you can also look at higher abstractions such as Pig and Hive to run jobs on your dataset in HDFS.

While running any MapReduce job you can monitor each job by going to port 8088 on the NameNode

and you can see the history of jobs run on Hadoop at **namenode_public_dns**:19888

When you are finished with your instances, be sure to shut them down or you may start incurring charges from AWS for the month

To terminate instances when you are finished with them, you can go to AWS Console and find the Instances tab along the left panel. Next highlight the instances you wish to terminate and the click on Actions -> Instance State -> Terminate



Final Thoughts

These types of development setups have helped Insight Fellows hit the ground running and explore various distributed technologies beyond Hadoop such as Kafka, Spark, Storm, Elasticsearch, and Cassandra. Although setting up these systems are not the primary concerns for data engineers, understanding how distributed systems are installed and connected is still a valuable skill.

. . .

Interested in transitioning to career in data engineering?
Find out more about the Insight Data Engineering Fellows Program in New York and Silicon Valley, apply today, or sign up for program updates.

Already a data scientist or engineer?
Find out more about our Advanced Workshops for Data Professionals. Register for two-day workshops in Apache Spark and Data Visualization, or sign up for workshop updates.