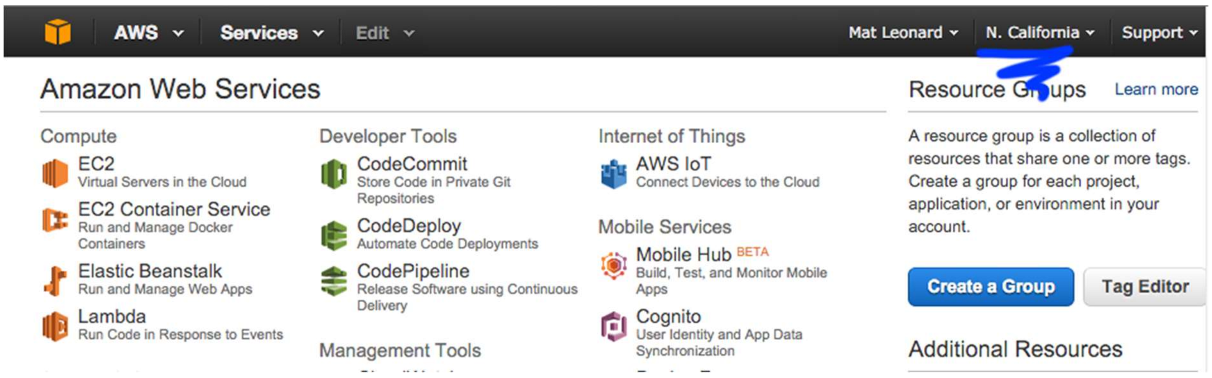
Launch an EC2 Instance

Hadoop is built for running on commodity hardware in a data center. However, I imagine few of you have server racks in your garage. Since you don't have access to a bunch of hardware, you're going to deploy a Hadoop cluster on Amazon Web Service (AWS) EC2 (Elastic Compute Cloud) virtual machines.
These are virtual machines we can request and run from anywhere in the world, typically at some cost per hour, per machine. Each virtual machine, an instance for shorthand, has a configurable number of CPUs and amount of memory and storage. We're going to be using a free offering, but most organizations will need more resources to store and analyze their data.

If you don't have an AWS account, you'll need to create one. It might ask you for your credit card information. This is because the instances are charged by the hour. For this first lesson, you'll be using free EC2 instances. However, completing the next two lessons will require using non-free instances.

After you log in, you should choose a region that is closest to you shown here.



A resource group is a collection of resources that share one or more tags. Create a group for each project, application, or environment in your account.

Now we're ready to create some instances. Here's what you should do:

1. From "Services", select "EC2".
2. Click the "Launch Instance" button.
3. Choose Ubuntu 64-Bit.
4. Choose t2.micro. Click "Next: Configure Instance Details."
5. Leave the number of instances to 1. We're going to install Hadoop on this instance and use it as a generic Hadoop node. Click "Next: Add Storage"

6. This is where you set the storage for your instance, 8 GB is fine. Of course, if you're doing this for an organization, you'll need to choose an appropriate storage size to fit your data. Click "Next: Tag Instance"
7. Here you can assign a tag to your instance. Anything is fine, I used "Hadoop node". Click "Next: Configure Security Group."
8. The security group determines who can connect to your instances. The default is "SSH" and 0.0.0.0/0. This means that any computer can connect to your instance over ssh. In practice, you would assign a restricted list of IP addresses that can access your cluster. However, for ease, you should leave it open. You'll also need to change the type to "All Traffic" so that we can access the instances over HTTP as well as SSH. Then, click "Review and Launch" to launch your instances.
   At the end of the process, you'll be prompted to create and download a private key. This key will allow you to connect to your instances with SSH. If you don't download this, or delete it somehow, you won't be able to connect to your cluster. If you lose it, don't panic, but you'll have to shut down the instances and start up new ones. Name it whatever you like, I just used "hadoop." Click "View Instances" to see your instance booting up in the EC2 dashboard. You'll want to write down the public hostname, called "Public DNS" on the instance panel.

Configure SSH

You're going to access the instance by logging in remotely with SSH. If you are using Linux or Mac OS X, this should be available in the terminal. If you're on Windows, I suggest installing Git Bash which comes with Git. We are going to make heavy use of shell commands, so if you are unfamiliar with it, please take our Linux Command Line Basics course. Going forward, I am going to assume that you have access to a terminal that can connect to the instances.

Firstly, you'll want to change the permissions on the private key file, or you'll get an error when you try connecting through ssh. To change permission on Linux and OS X, `$ sudo chmod 600 /path/to/key_file.pem`, where `key_file.pem` is the private key you created earlier.

Test logging in to the instance. Here `instance_hostname` is the public hostname of your instance:

```
$ ssh -i /path/to/key_file.pem ubuntu@instance_hostname
```

Now copy the private key to the instance:

```
$ scp -i /path/to/key_file.pem /path/to/key_file.pem
ubuntu@instance_hostname:~/.ssh
```

Install Java dan Hadoop

Update and upgrade:

```
$ sudo apt-get update && sudo apt-get dist-upgrade
```

Install OpenJDK:

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

Download Hadoop from one of [these mirrors](). Change the version number appropriately:

```
$ wget http://apache.mirrors.tds.net/hadoop/common/hadoop-2.7.2/hadoop-
2.7.2.tar.gz -P ~/Downloads
```

Extract it to `/usr/local`:

```
$ sudo tar zxvf ~/Downloads/hadoop-* -C /usr/local
```

Rename the directory to just `hadoop`:

```
$ sudo mv /usr/local/hadoop-* /usr/local/hadoop
```

Set environment variables

Find Java with:

```
readlink -f $(which java)
```

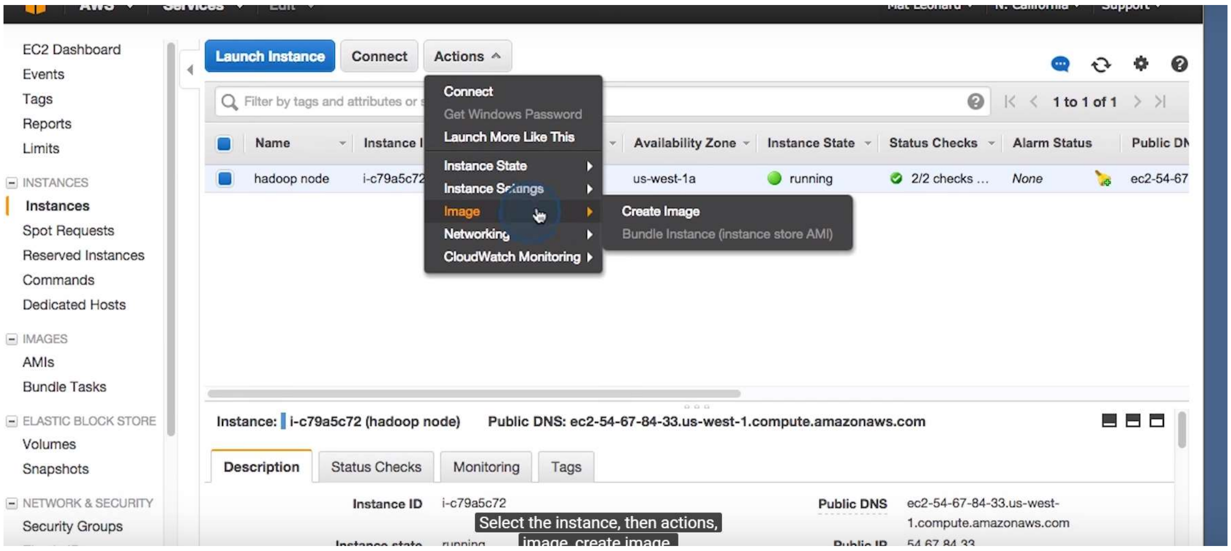
Environment variables:

```
export JAVA_HOME=/path/to/java
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
```

Load variables:

```
$ source ~/.bashrc
```

Save to image

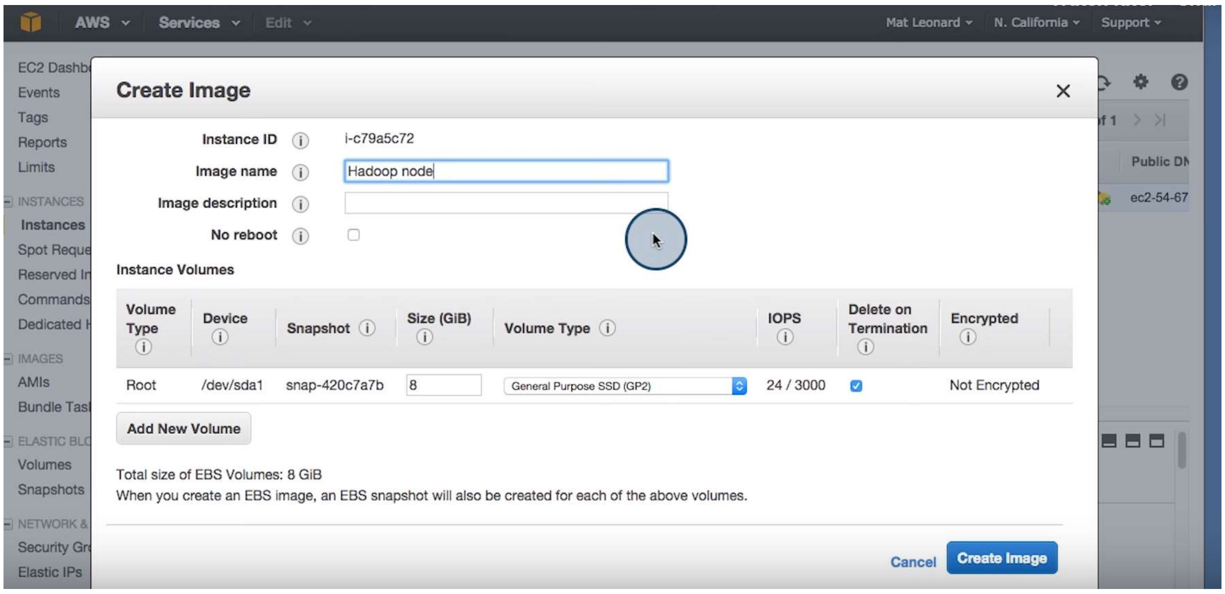


Name	Instance ID	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS
hadoop node	i-c79a5c72	us-west-1a	running	2/2 checks ...	None	ec2-54-67

Instance: i-c79a5c72 (hadoop node) Public DNS: ec2-54-67-84-33.us-west-1.compute.amazonaws.com

Instance ID i-c79a5c72
Public DNS ec2-54-67-84-33.us-west-1.compute.amazonaws.com

Select the instance, then actions, image, create image

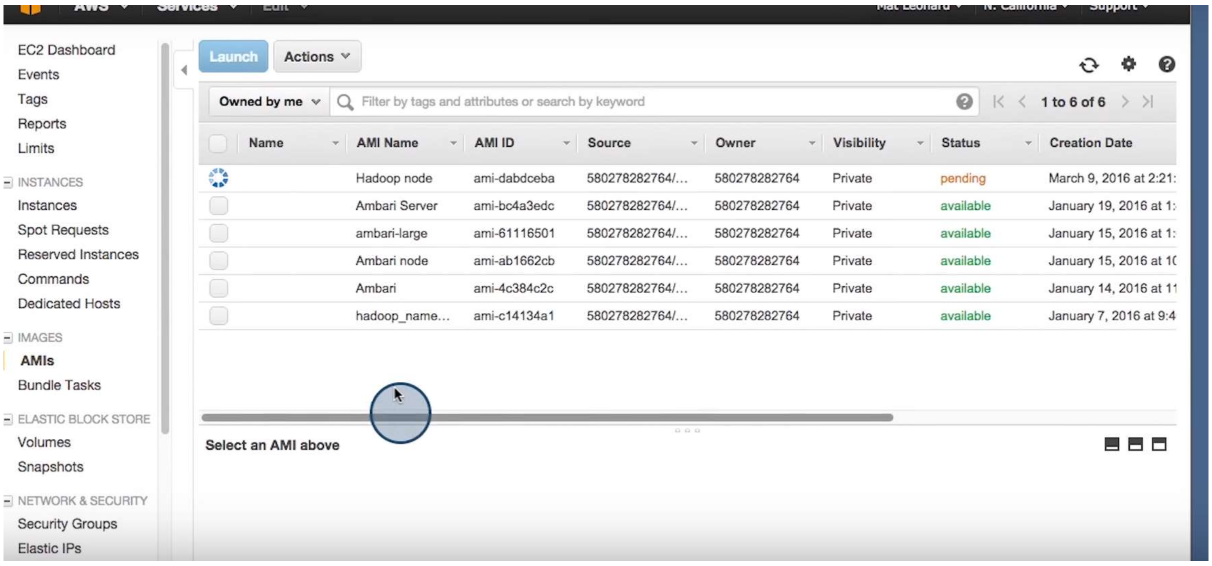


Instance ID i-c79a5c72
Image name Hadoop node
Image description
No reboot ☐

Instance Volumes

Volume Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Delete on Termination	Encrypted
Root	/dev/sda1	snap-420c7a7b	8	General Purpose SSD (GP2)	24 / 3000	✓	Not Encrypted

Total size of EBS Volumes: 8 GiB
When you create an EBS image, an EBS snapshot will also be created for each of the above volumes.



Name	AMI Name	AMI ID	Source	Owner	Visibility	Status	Creation Date
	Hadoop node	ami-dabdceba	580278282764/...	580278282764	Private	pending	March 9, 2016 at 2:21:
	Ambari Server	ami-bc4a3edc	580278282764/...	580278282764	Private	available	January 19, 2016 at 1:
	ambari-large	ami-61116501	580278282764/...	580278282764	Private	available	January 15, 2016 at 1:
	Ambari node	ami-ab1662cb	580278282764/...	580278282764	Private	available	January 15, 2016 at 1(
	Ambari	ami-4c384c2c	580278282764/...	580278282764	Private	available	January 14, 2016 at 11
	hadoop_name...	ami-c14134a1	580278282764/...	580278282764	Private	available	January 7, 2016 at 9:4

Select an AMI above

Launching nodes from the image

Now that you have an image with Java and Hadoop installed, you can use this as nodes in our cluster. Select the Hadoop node image, then click launch. Choose 4 instances, one will be the NameNode, the other three will be DataNodes.

Continue with configuration as before, remember to set the security to “All Traffic.” When launching, select “Choose an existing key pair” and the private key that you created before. The new instances will use this private key for connecting over SSH.

You should see all four nodes in the instances panel. You’ll need the public hostname of each instance often, so it's best to write each one in a file for reference.

To make logging into the instances more convenient, let’s create an SSH config file. First, on your computer, create the file with `$ touch ~/.ssh/config`, then edit `~/.ssh/config` in your favorite editor. Add these lines to the file.

```
Host namenode
  HostName namenode_public_hostname
  User ubuntu
  IdentityFile ~/.ssh/key_file.pem

Host datanode1
  HostName datanode1_public_hostname
  User ubuntu
  IdentityFile ~/.ssh/key_file.pem

Host datanode2
  HostName datanode2_public_hostname
  User ubuntu
  IdentityFile ~/.ssh/key_file.pem

Host datanode3
  HostName datanode3_public_hostname
  User ubuntu
  IdentityFile ~/.ssh/key_file.pem
```

This file lets SSH associate a shorthand name with a hostname, a user, and the private key, so you don't have to type those in each time. This is assuming your private key `key_file.pem` is in `.ssh`. If it isn't be sure to move or copy it there: `cp key_file ~/.ssh/key_file.pem`. Now you can log into the NameNode with just `$ ssh namenode`. Also, copy the config file to the NameNode:

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

You need to make sure the NameNode can connect to each DataNode over ssh without needing a password. You'll do this by creating a public key for the NameNode and adding it to each DataNode.

On the NameNode (remember `$ ssh namenode`), create a public key and copy it into `authorized_keys`:

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

Copy `authorized_keys` to each DataNode to enable passwordless login. On the NameNode:

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

Test this by logging into a DataNode from the NameNode:

```
$ ssh datanode1
```

You can get back to the NameNode with

```
$ exit
```

Configuring the cluster

You'll need the public hostname of each instance going forward, so it's really convenient to copy them down in a file. You can see each public hostname by selecting all the instances.

Hadoop has an intimidating number of configuration options. I'm only going to go through a few here, check the Hadoop documentation and Hadoop: The Definitive Guide for more information on configuration options.

Hadoop environment variables

The first file to take note of is hadoop-env.sh, it holds environment variables used by Hadoop. Here, you'll just need to set `JAVA_HOME`

```
export JAVA_HOME=/path/to/java
```

Cluster-wide configuration

First, you'll deal with the configuration on each node, then get into specific configurations for the NameNode and DataNodes. On each node, go to the Hadoop configuration folder, you should be able to get there with `$ cd $HADOOP_CONF_DIR` since we set that in `.bashrc` earlier. When editing these configuration files, you'll need root access so remember to use `$ sudo`. In the configuration folder, edit `core-site.xml`:

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

where `namenode_public_hostname` is the public hostname of your NameNode. This configuration `fs.defaultFS` tells the cluster nodes which machine the NameNode is on and that it will communicate on port 9000.

On each node, in `yarn-site.xml` you set options related to YARN, the resource manager:

```
<configuration>

<!– Site specific YARN configuration properties -->

  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
  <property>
    <name>yarn.resourcemanager.hostname</name>
    <value>namenode_public_hostname</value>
  </property>
</configuration>
```

Similarly with fs.defaultFS, yarn.resourcemanager.hostname sets the machine that the resource manager runs on.

On each node, copy `mapred-site.xml` from `mapred-site.xml.template`

```
$ sudo cp mapred-site.xml.template mapred-site.xml
```

and add:

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

```
</configuration>
```

Again, `mapreduce.jobtracker.address` sets the machine the job tracker runs on, and the port it communicates with. The other option here `mapreduce.framework.name` sets MapReduce to run on YARN.

NameNode specific configuration

Now, NameNode specific configuration, these will all be configured only on the NameNode. First, add the DataNode hostnames to `/etc/hosts`. You can get the hostname for each DataNode by entering `$ hostname`, or `$ echo $(hostname)` on each DataNode.

Now edit `/etc/hosts` and include these lines:

```
127.0.0.1 localhost
namenode_public_hostname namenode_hostname
datanode1_public_hostname datanode1_hostname
datanode2_public_hostname datanode2_hostname
datanode3_public_hostname datanode3_hostname
```

Now, on the NameNode, edit `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/data/hdfs/namenode</value>
  </property>
</configuration>
```

`dfs.replication` sets how many times each data block is replicated across the cluster. `dfs.namenode.name.dir` sets the directory for storing NameNode data (`.fsimage`). You'll also have to create the directory to store the data:

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

Next, you'll create the `masters` file in `HADOOP_CONF_DIR`. The `masters` file sets which machine the secondary namenode runs on. In your case, you'll have the secondary NameNode run on the same machine as the NameNode, so edit `masters`, add the hostname of NameNode (**Note:** Not the public hostname, but the hostname you get from `$ hostname`). Typically though, you would have the secondary NameNode run on a different machine than the primary NameNode.

Next, edit the `slaves` file in `HADOOP_CONF_DIR`, this file sets the machines that are DataNodes. In `slaves`, add the hostnames of each datanode (**Note:** Again, not the public hostname, but `$ hostname` hostnames). The `slaves` file might already contain a line `localhost`, you should remove it, otherwise the NameNode would run as a DataNode too. It should look like this:

```
datanode1_hostname
datanode2_hostname
datanode3_hostname
```

Finally on the NameNode, change the owner of `HADOOP_HOME` to `ubuntu`:

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

DataNode specific configuration

Now on **each** DataNode, edit `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/data/hdfs/datanode</value>
  </property>
</configuration>
```

Again, this sets the directory where the data is stored on the DataNodes. And again, create the directory on each DataNode:

```
$ sudo mkdir -p $HADOOP_HOME/data/hdfs/datanode
```

and change the owner of the Hadoop directory

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

Launch Hadoop Cluster

On the NameNode, format the file system, then start HDFS.

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

Start YARN:

```
$ $HADOOP_HOME/sbin/start-yarn.sh
```

Start the job history server:

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

To see the Java processes (Hadoop daemons for instance), enter

```
$ jps
```