

HDP CLoud Deployment Guide

Reference Architecture and How-To for HDP on Amazon AWS

Table of Contents

Installation and Deployment Process 2

Prerequisites 2

Step 1 – Create SSH Keys 2

Step 2—Create EC2 Instances 3

Step 3 – Prepare the Nodes 10

Step 4 – Install Ambari 16

Step 5 – Install HDP 20

Appendix 29

Scripting It Out 29

Why Hortonworks for Hadoop? 32

About Hortonworks 32

# Installation and Deployment Process

This section will demonstrate the steps required to successfully setup a HDP cluster on AWS. We will be setting up a seven node cluster:

* Name Node
* Secondary NameNode
* Four Data Nodes
* Utility Node

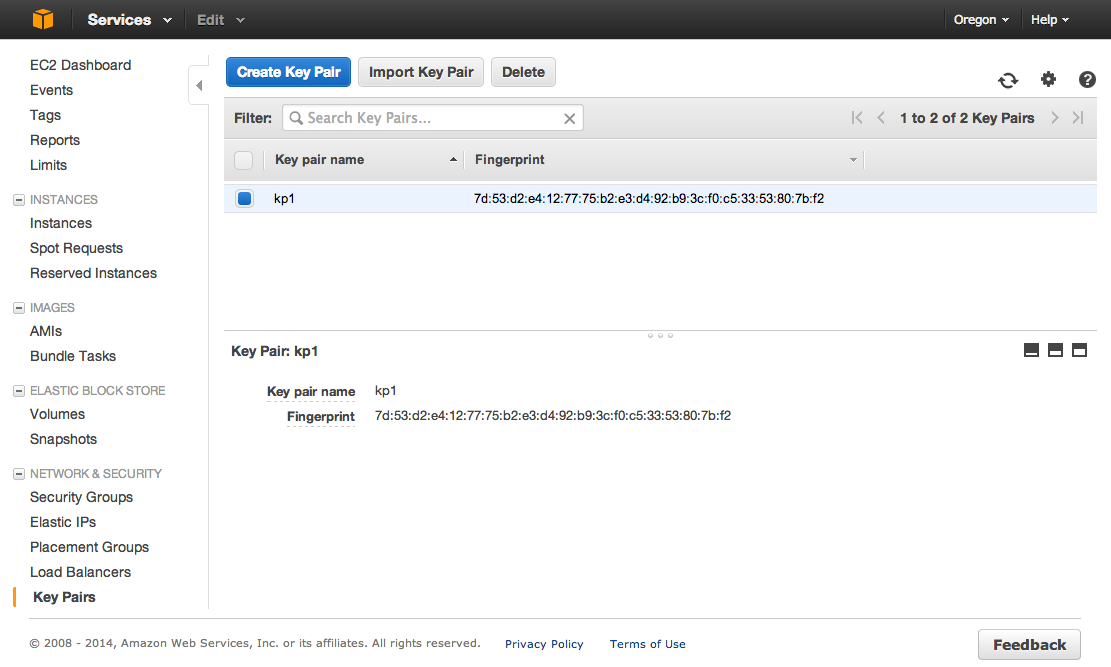
For the sake of simplicity all nodes in the cluster will be of the same instance type (m3.2xlarge).

## Prerequisites

* Amazon Web Services account with the ability to launch 7 large instances of EC2 nodes.
* A Mac or a Linux machine. You could also use Windows but you will have to install additional software for SSH
* A basic understanding of Amazon Web Services, EC2, and the Hortonworks Data Platform (HDP)

## Step 1 – Create SSH Keys

This step is only necessary if you have not already created SSH keypair for AWS. From the AWS console, select “Key Pairs”. Select the “Create Key Pair” button to create a new key pair. Name it whatever you like. When this keypair is first created it will automatically download the private key (.pem file) to your local machine. You will need this private key later on so do not lose this keyfile.



## Step 2—Create EC2 Instances

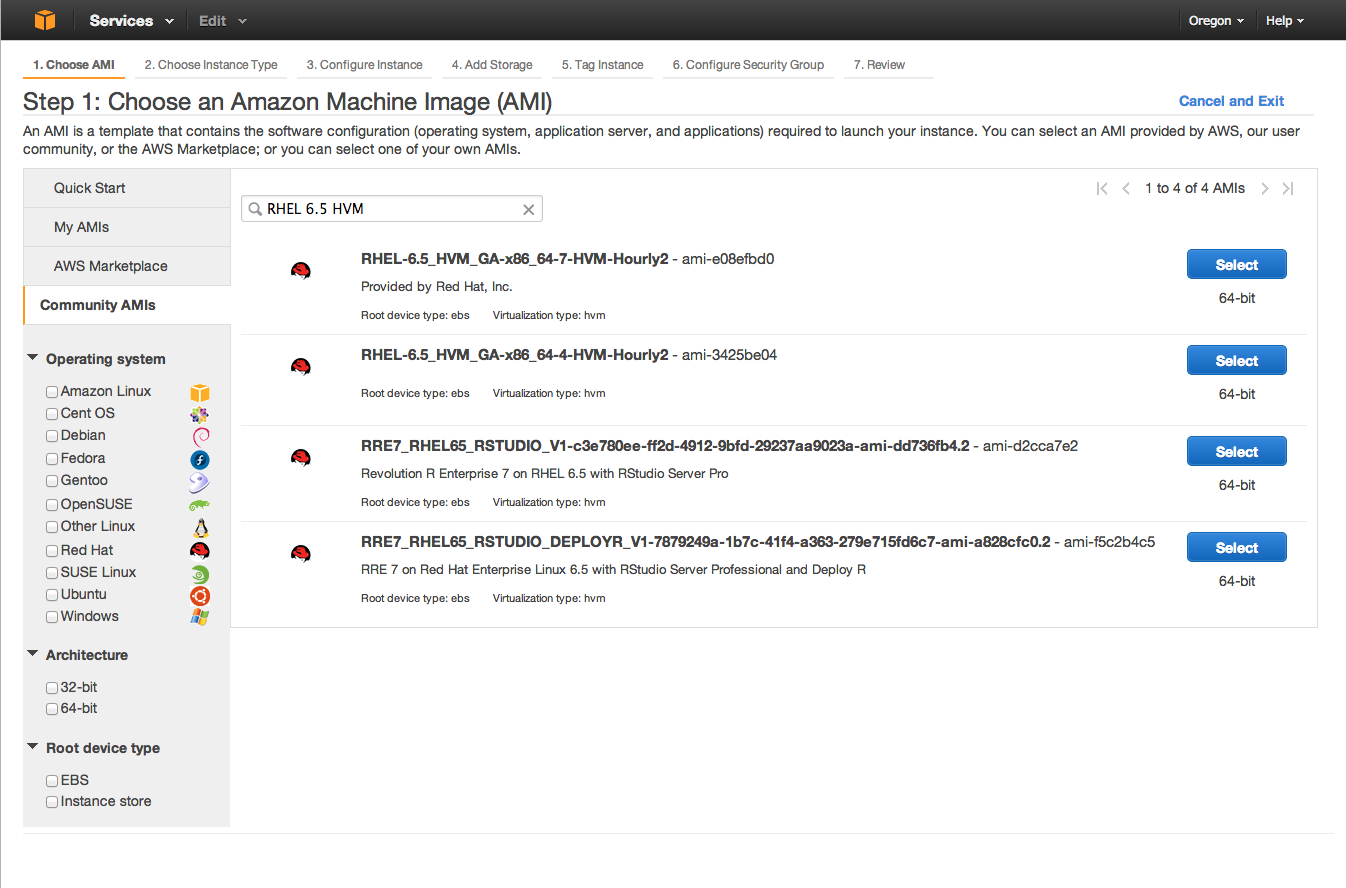
To create an EC2 instance select the “Launch Instance” button form the EC2 Dashboard. This will start the EC2 instance wizard.

### Select your AMI

The first step is to select the AMI we wish to use. HDP supports RHEL, CentOS, SLES, Oracle Linux or Windows Server. For this discussion we will be using Red Hat Enterprise Linux 6.5. AMIs come in two different types of virtualization:

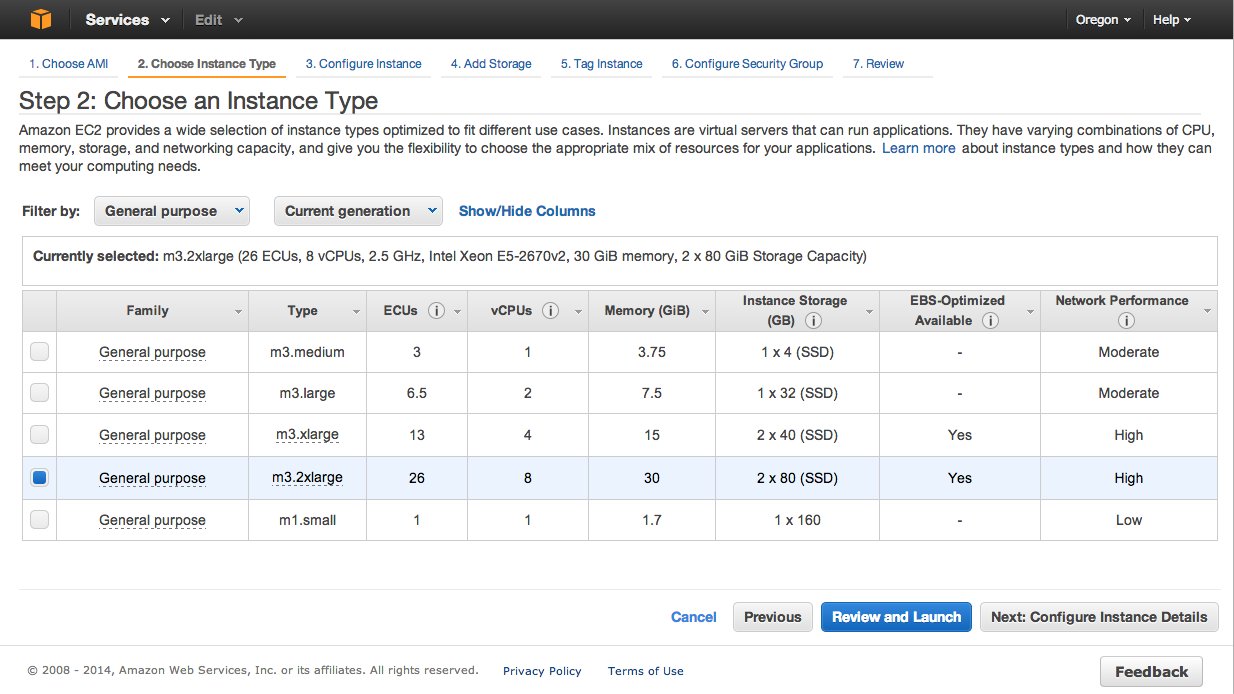
* Paravirtual (Para)
* Hardware-Assisted Virtualized Mode (HVM)

A discussion of the differences between the two is beyond the scope of this document, but for production clusters where performance and optimization are essential we recommend HVM AMI types. For development or test clusters Paravirtual is fine. AMIs from the free tier are only available as Paravirtual. For our exercise [current as of 28May2014 but subject to change by Amazon] we will be selecting Red Hat Enterprise Linux 6.5 (HVM) 64-bit (ami-e08efbd0). This AMI is EBS Backed.



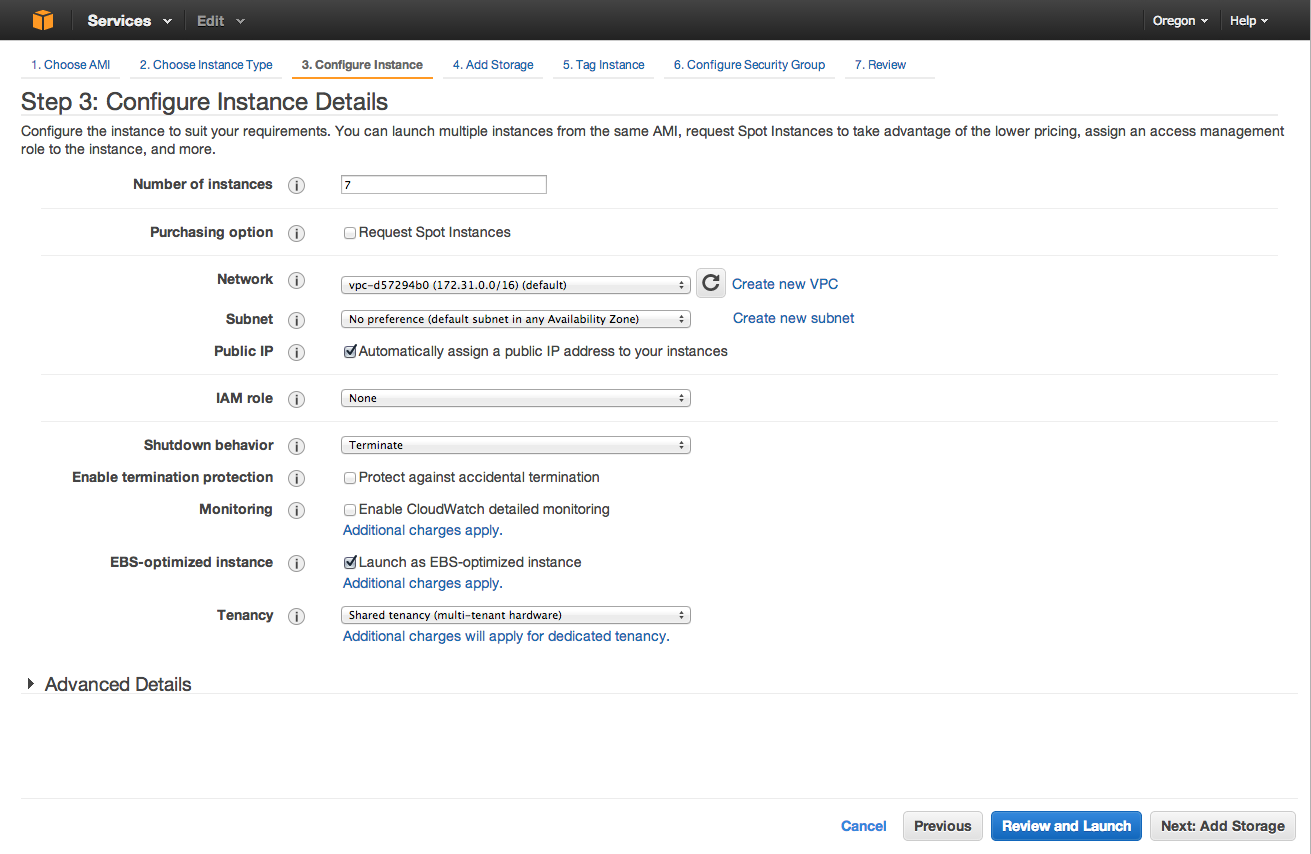
### Choose Instance Types

From the “Filter by” drop down select “General purpose” and then select the “m3.2xlarge” instance type. Notice this instance type has 2x80GB instance stores which we will be using for /var/log and /opt. Press the “Next: Configure Instance Details” button to proceed.



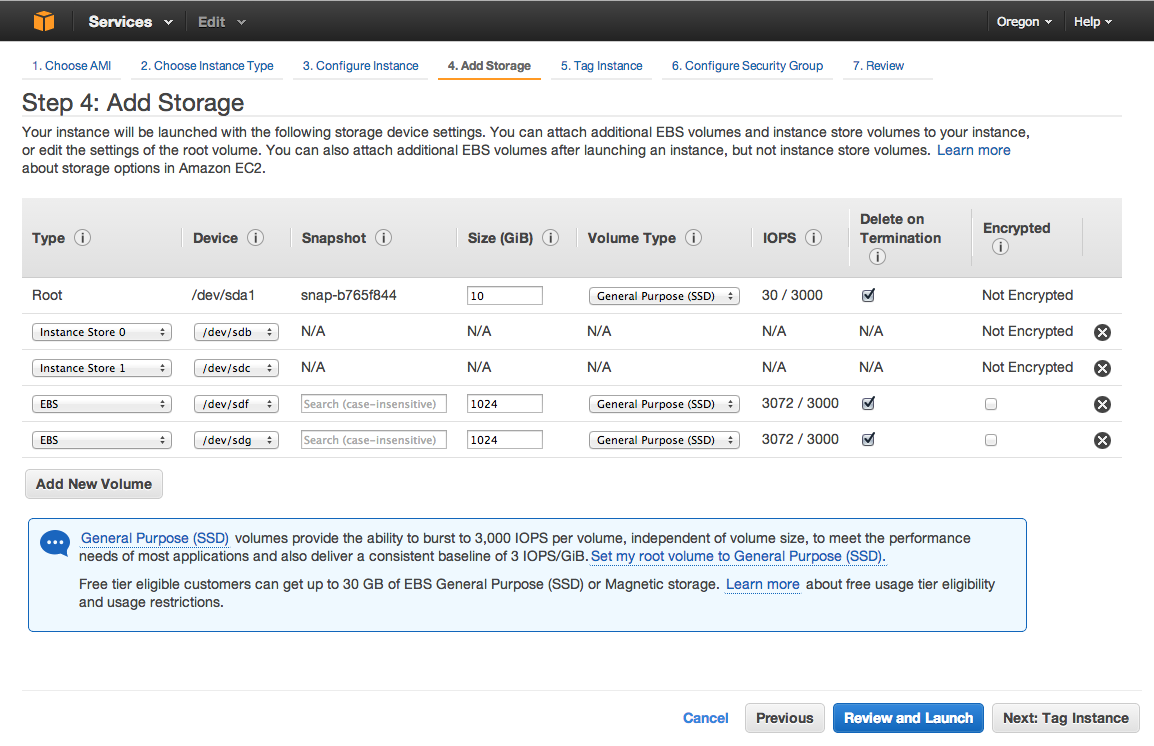
### Configure Instance Details

Here we will configure the instance details. To start lest put “7” in the “Number of Instances” text box. Select a VPC network form the dropdown menu. If you do not have a VPC established go ahead and set one up. We don’t want to use the AWS public zone here for a variety of reasons. Make sure the box is checked to automatically assign a public IP address. Since this is a demo cluster we will also change the Shutdown behavior to “Terminate” form the dropdown menu. In a real cluster you would probably want to leave this as “Stop” and check the “Enable termination protection” box. Also in a real cluster, and if your instance type has this option, you would want to check the box to launch as EBS-optimized instances. There are some additional charges for this but there is a performance gain. We will be placing all of our HDFS data on EBS optimized storage.



### Add and Configure Storage

Partitioning recommendations for HDP can be found at <http://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.1.2/bk_cluster-planning-guide/content/file_system.html> . We will leave the root partion alone as AWS would only allocate 6GB to / in any case. However the HDP docs recommend at least 20Gb for the root partition. To get the additional space we will be using the instance stores. Add in the two instance stores which we will use one for /var/log and the other for /mnt. By default AWS will provision the /dev/xvdb partition to /mnt. For this exercise we will add two EBS volumes for our HDFS data; each at 1024GB (1TB). See the Storage Option section above for a discussion of EBS vs Instance stores for storing HDFS data. Hit the “Next Tag Instance” button. Amazon also recommends that any EBS partitions use /dev/xvd[f-p].



### Tag Instance

We will skip this step for this exercise but in a production environment: tag, tag, tag. Especially if you are running a cluster of significant size or even multiple clusters.

### Configure Security Groups

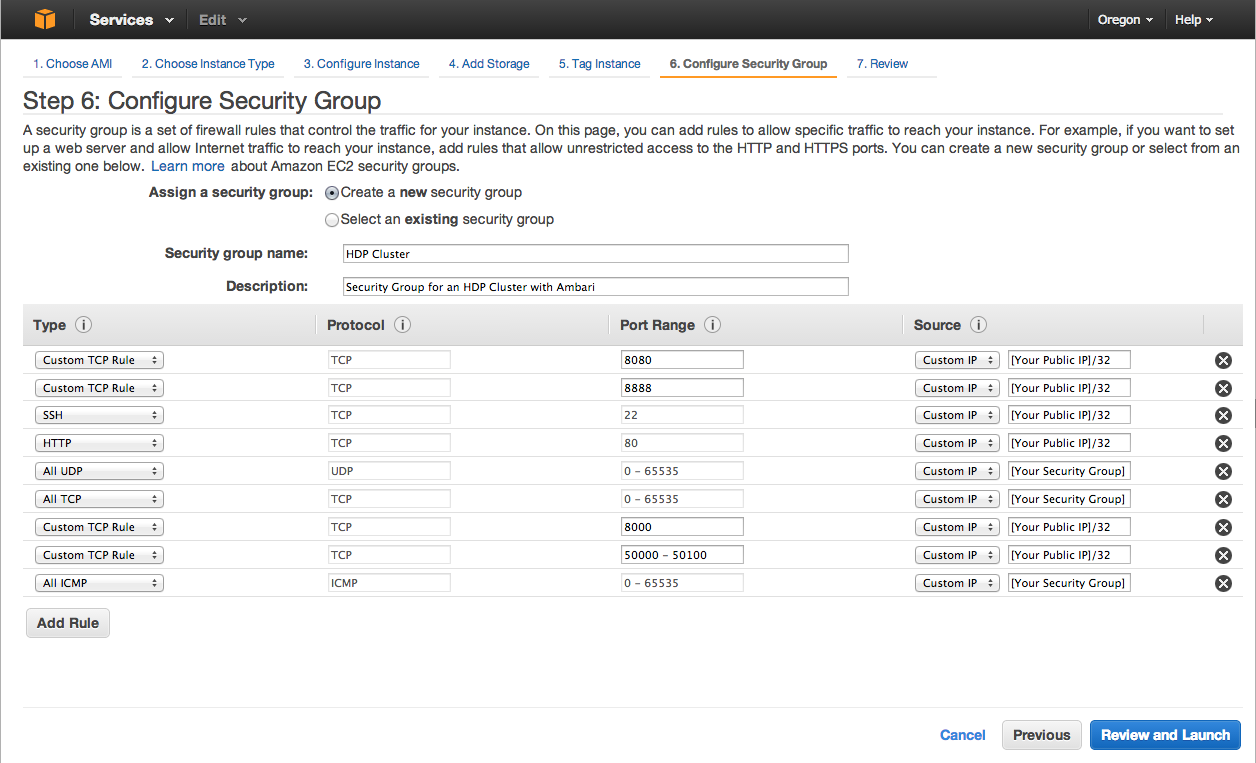
We are summing here that you will only be accessing this cluster from a small (or single) number of external hosts.

A complete list of the required ports, is beyond the scope of this exercise but the complete list can be found on the Hortonworks website:

* HDP2.1: <http://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.1.2/bk_reference/content/reference_chap2.html>
* Ambari: <http://docs.hortonworks.com/HDPDocuments/Ambari-1.6.0.0/bk_ambari_reference/content/amb-ref-network-ports.html>
* Nagios: <http://docs.hortonworks.com/HDPDocuments/Ambari-1.6.0.0/bk_ambari_reference/content/amb-ref-amb-nagios-ports.html>
* Ganglia: <http://docs.hortonworks.com/HDPDocuments/Ambari-1.6.0.0/bk_ambari_reference/content/amb-ref-ganglia-ports.html>

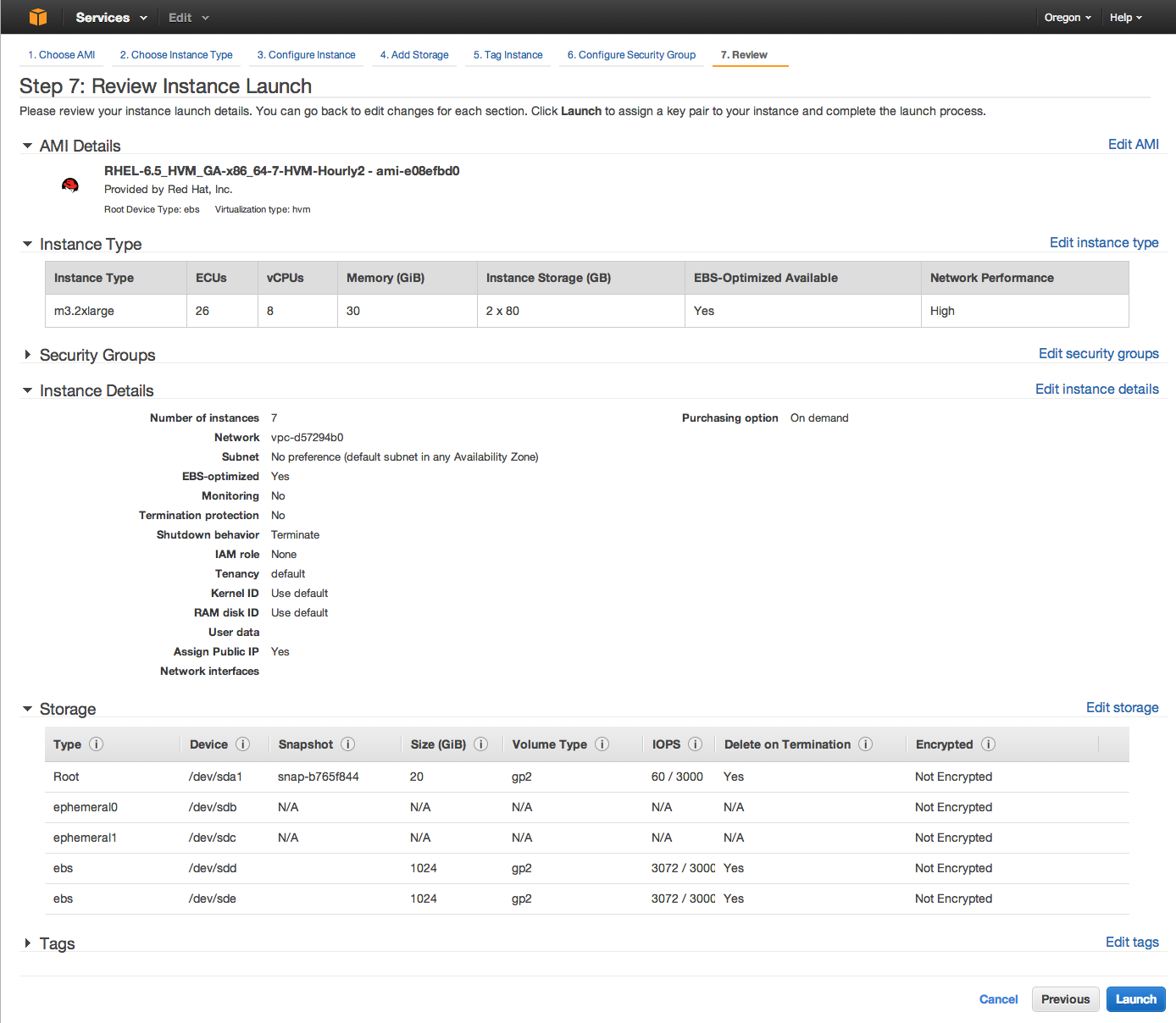
For our exercise we will configure a security group as displayed in the following image. You will obviously need to replace “[Your Public IP]/32” with your IP address and subnet mask you wish to access the cluster from. In this example the /32 denotes a single host that we will access the cluster from. Also replace “[Your Security Group]” with the id of your actual security group. You will notice that for brevity sake we have just opened all ports internally to the security group. They are not accessible to the outside world but only between the nodes in the cluster. In a production environment you would probably want to be very specific with regards to the ports opened internally as well.

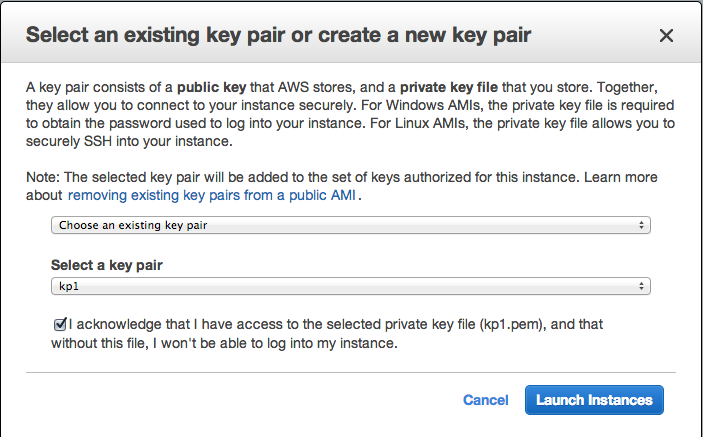
Hit the “Review and Launch” button



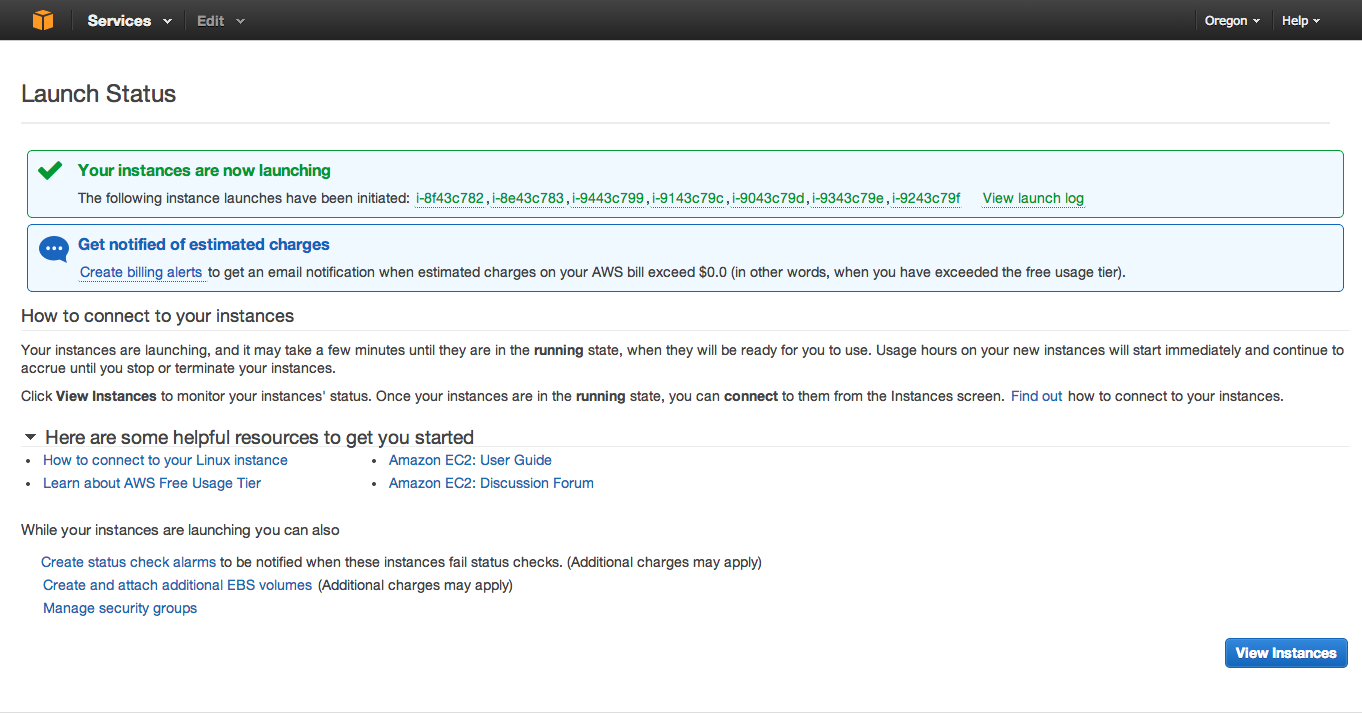
### Review and Launch Instances

You should see a screen similar to the one below. Take a look and make sure all your instance details are correct and hit the “Launch” button. It should bring up a popup asking you to upload your keypair. Select the keypair we created way back in step 1 and check the box that says you still have the private key file. You do still have that and know where it is don’t you? If not, create a new keypair and this time don’t lose the private key file. Hit the “Launch Instances” button.





Hit the “Launch Instances” button. You should see a screen like this saying all is well:



If you see this, congratulations you have successfully setup your instances. Not let’s install HDP on them.

## Step 3 – Prepare the Nodes

We will need to prepare all the nodes for use as part of the HDP cluster. Here are the steps we will need to complete before installing HDP.

* Disable SELinux
* Turn off autostart of iptables and ip6tables
* Format all new storage volumes
* Set swappiness
* Set file access time
* Set root reserved space
* Turn on Name Service Cache Daemon
* Setup NTP
* Set File Handle Limits
* Create /var/log as a mount point
* Create the /data0 and /data1 directories as mount points
* Edit /etc/fstab for our new partitions and mount points
* Add all hosts to /etc/host (Do not use /etc/host for a real cluster)

We will be walking through these individually but it is highly recommended to script this out for automation.

### Create a List of Hostnames

Before we start it will be very helpful to create a file on your local machine called cluster-hosts.txt and list the **public IP addresses** for all your nodes in it:

myLaptop$ cat cluster-hosts.txt

[public IP 1]

[public IP 2]

[public IP 3]

[public IP 4]

[public IP 5]

[public IP 6]

[public IP 7]

### Test Connection to Each Node

Test each node by running a command like hostname -f from your local machine to each node in the cluster. You can do this by using a script to loop through each listing in the cluster-hosts.txt file. The following script shows an example:

#!/bin/bash

# testHosts.sh

echo 'Opening File:' $1

for i in $(< $1)

do

 #echo -e "\n\*\*\* ${i} \*\*\*";

 ssh -o StrictHostKeyChecking=no -i $2 ec2-user@${i} "hostname -f";

done

To run this script:

myLaptop$ ./testHosts.sh cluster-hosts.txt [PATH/TO/SSH/KEY]

The results should look something like this:

\*\*\*54.218.11.194\*\*\*

ip-172-31-23-77.us-west-2.compute.internal

\*\*\*54.186.140.175\*\*\*

ip-172-31-23-76.us-west-2.compute.internal

\*\*\*54.213.240.199\*\*\*

ip-172-31-23-79.us-west-2.compute.internal

\*\*\*54.218.11.120\*\*\*

ip-172-31-23-78.us-west-2.compute.internal

\*\*\*54.218.11.157\*\*\*

ip-172-31-23-81.us-west-2.compute.internal

\*\*\*54.218.11.106\*\*\*

ip-172-31-23-75.us-west-2.compute.internal

\*\*\*54.218.11.196\*\*\*

ip-172-31-23-80.us-west-2.compute.internal

Make sure you get a valid response from every node in the cluster before proceeding. SSH to the Ambari Manager node's external address from your local machine as ec2-user using your private key:

### SSH to Ambari Host

First let’s scp our private key to the Ambari host. We will use this to establish passwordless ssh form the Ambari host to all other hosts in the cluster. In a production environment you would want to establish a new keypair and use that keypair for all hosts in the cluster. This limits the potential security risk of exposing the AWS private key.

myLaptop$ scp –i [your-private-key].pem [your-private-key].pem ec2-user@[Ambari hosts public IP]:~/.ssh/[your-private-key].pem

myLaptop$ ssh -i [your-private-key].pem ec2-user@[Ambari hosts public IP]

[ambari ~]$ chmod 600 .ssh/[ your-private-key]

Lets take a look at the available disks, partitions and space available.

[ambari ~]$ lsblk

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT

xvda 202:0 0 10G 0 disk

└─xvda1 202:1 0 6G 0 part /

xvdb 202:16 0 75G 0 disk /mnt

xvdc 202:32 0 75G 0 disk

xvdg 202:96 0 1T 0 disk

xvdf 202:80 0 1T 0 disk

We can see that AWS has provision the / partition on /dev/xvda1 and automatically set up /dev/xvdb for the /mnt partition. These are fine for now. We will be using the /dev/xvdc partition for /var/log, and /dev/xvdf and /dev/xvdg for our HDFS data at /data0 and /data1

### Hosts File

First we will edit the host file on each node so that all nodes can see each other. In a production cluster or just as a normal practice we should not use /etc/hosts for name resolution. Rather we should be using DNS. By default AWS has an internal DNS mechanism for the private IP and hostnames of all the nodes in our cluster. But for our exercise we will setup /etc/hosts so that we can also use name aliasing which will reduce our typing in the long run. To start let’s edit the /etc/hosts file on our Ambari host.

[ambari ~]$ sudo vi /etc/hosts

127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4

::1 localhost localhost.localdomain localhost6 localhost6.localdomain6

#USE THE INTERNAL IP AND FQDN HOSTNAME HERE

172.31.23.217 ip-172-31-23-217.us-west-2.compute.internal ambari

172.31.23.218 ip-172-31-23-218.us-west-2.compute.internal nn1

172.31.23.219 ip-172-31-23-219.us-west-2.compute.internal nn2

172.31.23.219 ip-172-31-23-214.us-west-2.compute.internal data1

172.31.23.220 ip-172-31-23-220.us-west-2.compute.internal data2

172.31.23.215 ip-172-31-23-215.us-west-2.compute.internal data3

172.31.23.216 ip-172-31-23-216.us-west-2.compute.internal data4

[ambari ~]$ ping ambari

PING ip-172-31-23-217.us-west-2.compute.internal (172.31.23.217) 56(84) bytes of data.

64 bytes from ip-172-31-23-217.us-west-2.compute.internal (172.31.23.217): icmp\_seq=1 ttl=64 time=0.019 ms

[ambari ~]$ ping nn1

PING ip-172-31-23-218.us-west-2.compute.internal (172.31.23.218) 56(84) bytes of data.

64 bytes from ip-172-31-23-218.us-west-2.compute.internal (172.31.23.218): icmp\_seq=1 ttl=64 time=2.09 ms

[ambari ~]$ ping nn2

PING ip-172-31-23-219.us-west-2.compute.internal (172.31.23.219) 56(84) bytes of data.

64 bytes from ip-172-31-23-219.us-west-2.compute.internal (172.31.23.219): icmp\_seq=1 ttl=64 time=2.22 ms

[ambari ~]$ ping data1

PING ip-172-31-23-214.us-west-2.compute.internal (172.31.23.219) 56(84) bytes of data.

64 bytes from ip-172-31-23-219.us-west-2.compute.internal (172.31.23.219): icmp\_seq=1 ttl=64 time=0.233 ms

[ambari ~]$ ping data2

PING ip-172-31-23-220.us-west-2.compute.internal (172.31.23.220) 56(84) bytes of data.

64 bytes from ip-172-31-23-220.us-west-2.compute.internal (172.31.23.220): icmp\_seq=1 ttl=64 time=0.727 ms

[ambari ~]$ ping data3

PING ip-172-31-23-215.us-west-2.compute.internal (172.31.23.215) 56(84) bytes of data.

64 bytes from ip-172-31-23-215.us-west-2.compute.internal (172.31.23.215): icmp\_seq=1 ttl=64 time=2.55 ms

[ambari ~]$ ping data4

PING ip-172-31-23-216.us-west-2.compute.internal (172.31.23.216) 56(84) bytes of data.

64 bytes from ip-172-31-23-216.us-west-2.compute.internal (172.31.23.216): icmp\_seq=1 ttl=64 time=1.93 ms

As an extra precaution we could ping each node form each node. Obviously this should be scripted. Once we are satisfied we will need to copy the hosts file to each other host. To do this let’s first create a file with just the private IP addresses of each host on the cluster:

[ambari ~]$ cat /etc/hosts | xargs -L 1 -I xx grep xx /etc/hosts | awk '!/localhost/ {print $1}' > clusterHostIps.txt

**Pro-Tip: Parsing the /etc/hosts file:**

#Prints the IP address of each entry in /etc/hosts

cat /etc/hosts | xargs -L 1 -I xx grep xx /etc/hosts | awk '!/localhost/ {print $1}'

#Prints the FQDN of each entry in /etc/hosts

cat /etc/hosts | xargs -L 1 -I xx grep xx /etc/hosts | awk '!/localhost/ {print $2}'

#Prints the first alias of each entry in /etc/hosts

cat /etc/hosts | xargs -L 1 -I xx grep xx /etc/hosts | awk '!/localhost/ {print $3}'

Now let’s copy the hosts file. Use the following script.

#!/bin/bash

for i in $(cat clusterHostIps.txt)

do

echo "=== ${i} ==="

ssh -t -oStrictHostKeyChecking=no -i [your-private-key].pem ec2-user@${i} "hostname -f";

scp -oStrictHostKeyChecking=no -i [your-private-key].pem /etc/hosts ec2-user@${i}:hosts

ssh -t -oStrictHostKeyChecking=no -i [your-private-key].pem ec2-user@${i} "sudo cp /etc/hosts /etc/hosts.bak; sudo cp ~/hosts /etc/hosts";

done

### Set umask

The HDP docs specify that we need to set the umask to 022 so let’s just go ahead and do that now on all hosts.

[ambari ~]$ umask

umask 002

[ambari ~]$ echo “umask 022” > ~/.bashrc

[ambari ~]$ source .bashrc

[ambari ~]$ umask

umask 022

### Disable SELinux

SELinux needs to be disabled on all hosts. Check the SELinux status with:

[ambari ~]$ sestatus

SELinux status: enabled

SELinuxfs mount: /selinux

Current mode: enforcing

Mode from config file: enforcing

Policy version: 24

Policy from config file: targeted

Disable it with:

[ambari ~]$ sudo sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/selinux/config

### NTP

Hadoop likes all the nodes to be in sync with each other form a time perspective so lets make sure NTP is installed, running and syncd on all hosts.

[ambari ~]$ sudo chkconfig ntpd on

[ambari ~]$ sudo ntpd -q

[ambari ~]$ sudo service ntpd start

### IPTables

Turn off autostart of iptables and ip6tables on all hosts in the cluster

[ambari ~]$ sudo service iptables stop

[ambari ~]$ sudo chkconfig iptables off

[ambari ~]$ sudo service ip6tables stop

[ambari ~]$ sudo chkconfig ip6tables off

### Storage Volumes

To start we will format all new storage volumes except for the root volume using ext4. This step can take a while, about 10 minutes-ish, depending on the size of the partitions you have. Skip ‘/’ and ‘mnt’ disks typically xvda and xvdb.

[ambari ~]$ sudo mkfs -t ext4 /dev/xvd<based on your lsblk output>

[ambari ~]$ sudo mkfs -t ext4 /dev/xvd<based on your lsblk output>

[ambari ~]$ sudo mkfs -t ext4 /dev/xvd<based on your lsblk output>

Create our new directories for /var/log, /data0 and /data1 directories as mount points. Let’s first save the existing /var/log directory.

[ambari ~]$ sudo mv /var/log /var/log-original

[ambari ~]$ sudo mkdir /var/log /data0 /data1

Now we will create the needed entries in /etc/fstab for our new partitions and mount points. Back up the existing /etc/fstab.

[ambari ~]$ sudo cp /etc/fstab /etc/fstab.bak

[ambari ~]$ sudo sed -i '$ a/dev/xvdc /var/log ext4 defaults 0 0' /etc/fstab

[ambari ~]$ sudo sed -i '$ a/dev/xvdf /data0 ext4 defaults,noatime 0 0' /etc/fstab

[ambari ~]$ sudo sed -i '$ a/dev/xvdg /data1 ext4 defaults,noatime 0 0' /etc/fstab

### Performance Improvements

All these steps are completely optional but definitely recommended in a production environment.

* Set swappiness

On all hosts in the cluster.

[ambari ~]$ echo 0 > /proc/sys/vm/swappiness

Note: Have seen that the above command throws a permissions denied error. In that case use the following:

[ambari ~]$ sudo /bin/su -c "echo 0 > /proc/sys/vm/swappiness"

[ambari ~]$ echo “vm.swappiness = 0” >> /etc/sysctl.conf

Note: Have seen that the above command throws a permissions denied error. In that case use the following:

[ambari ~]$ sudo /bin/su “echo ‘vm.swappiness = 0’ >> /etc/sysctl.conf”

* Set file access time

In the fstab set noatime on all hdfs partitions on all hosts in the cluster. This was done when we edited the fstab

/dev/xvdf /data01 ext3 defaults, noatime 0

/dev/xvdg /data1 ext3 defaults, noatime 0

[ambari ~]$ sudo mount –o remount /data1, /data2

* Set root reserved space

On all nodes in the cluster. This is just for the HDFS disks

[ambari ~]$ tune2fs –m 0 /dev/xvdf

[ambari ~]$ tune2fs –m 0 /dev/xvdg

* Turn on Name Service Cache Daemon

On all hosts in the cluster.

[ambari ~]$ chkconfig --level 345 ncsd on

[ambari ~]$ ncsd -g

* Set File Handle Limits

On all hosts in the cluster.

[ambari ~]$ echo hdfs – nofile 32768 >> /etc/security/limits.conf

[ambari ~]$ echo mapred – nofile 32768 >> /etc/security/limits.conf

[ambari ~]$ echo hbase – nofile 32768 >> /etc/security/limits.conf

[ambari ~]$ echo hdfs – nproc 32768 >> /etc/security/limits.conf

[ambari ~]$ echo mapred – nofile 32768 >> /etc/security/limits.conf

[ambari ~]$ echo hbase – nofile 32768 >> /etc/security/limits.conf

Whew! That’s a lot of configuration we need to do on all the nodes. Look at the Appendix of this document for a script of all the steps, as well as some ways to parallelize this to run on all the nodes at once.

Now let’s install Ambari.

## Step 4 – Install Ambari

Now that the heavy lifting is done, installing Ambari should be a relatively easy exercise. We will be following the HDP Automated Installation with Ambari installation docs. (<http://docs.hortonworks.com/HDPDocuments/Ambari-1.6.0.0/bk_using_Ambari_book/content/ambari-chap2-1.html> )

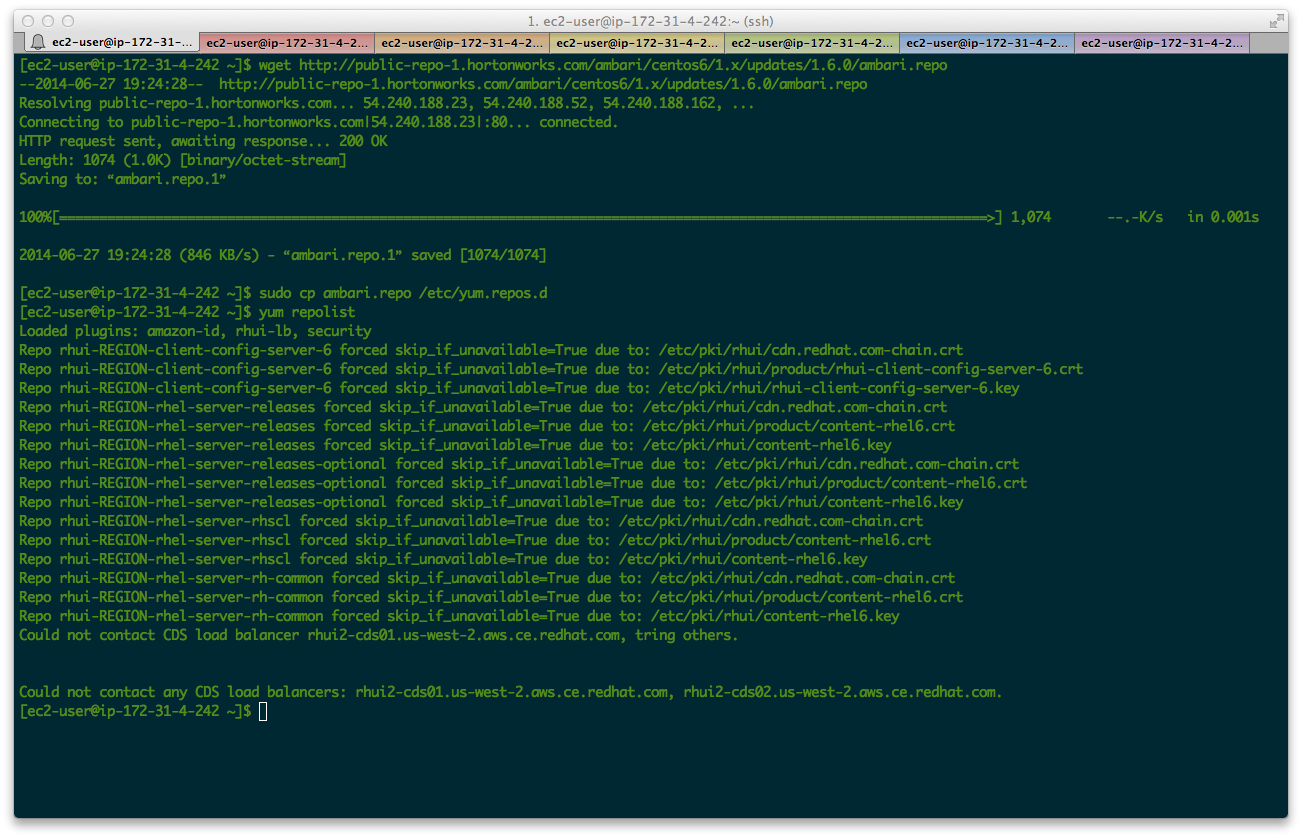
You will notice that in the screenshot below I have open a SSH session to every node in the cluster. With a seven node cluster this is somewhat doable in a larger cluster this would not be realistic to do and steps should be taken to automate and parallelize as much of the work as possible.

The first step is to get the Ambari bits by downloading the Ambari repository.

[ambari ~]$ wget <http://public-repo-1.hortonworks.com/ambari/centos6/1.x/updates/1.6.0/ambari.repo>

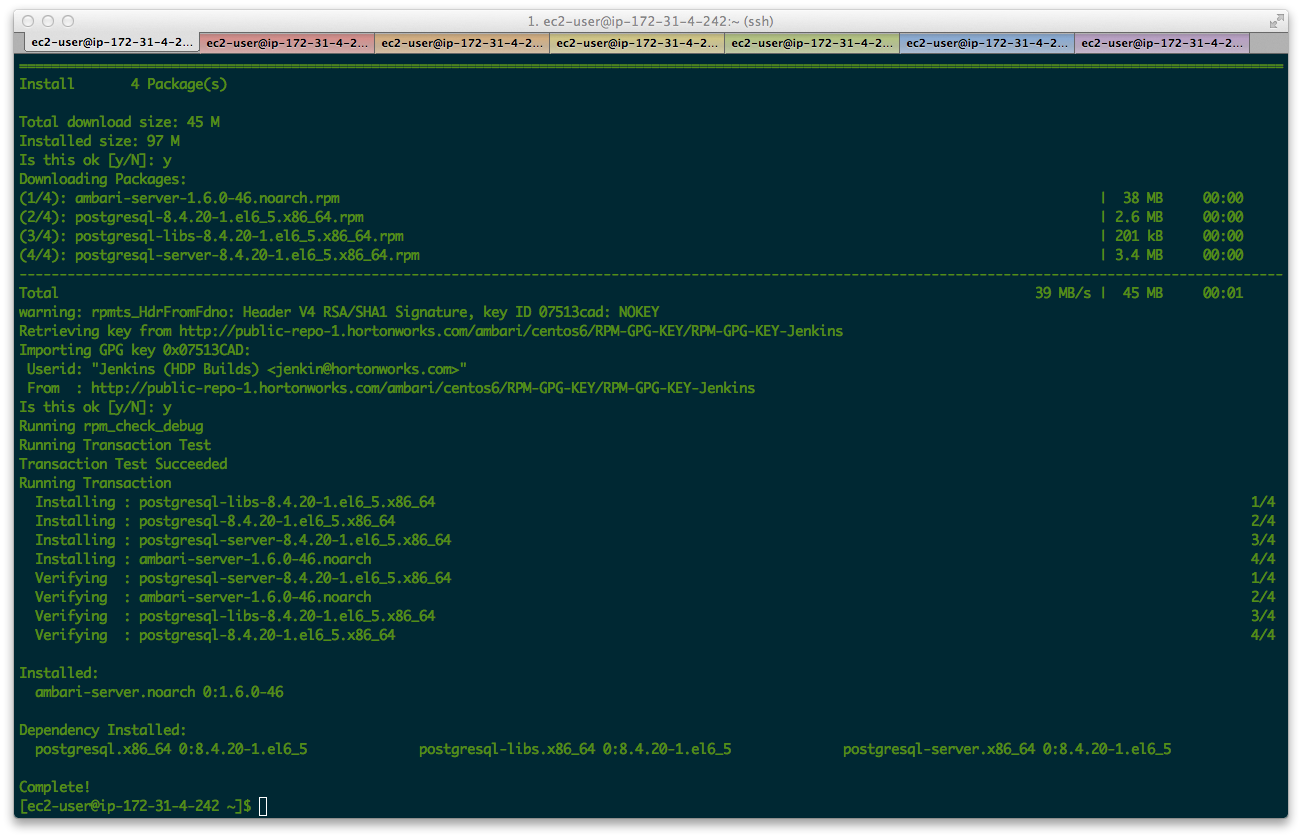
[ambari ~]$ sudo cp ambari.repo /etc/yum.repos.d

[ambari ~]$ yum repolist



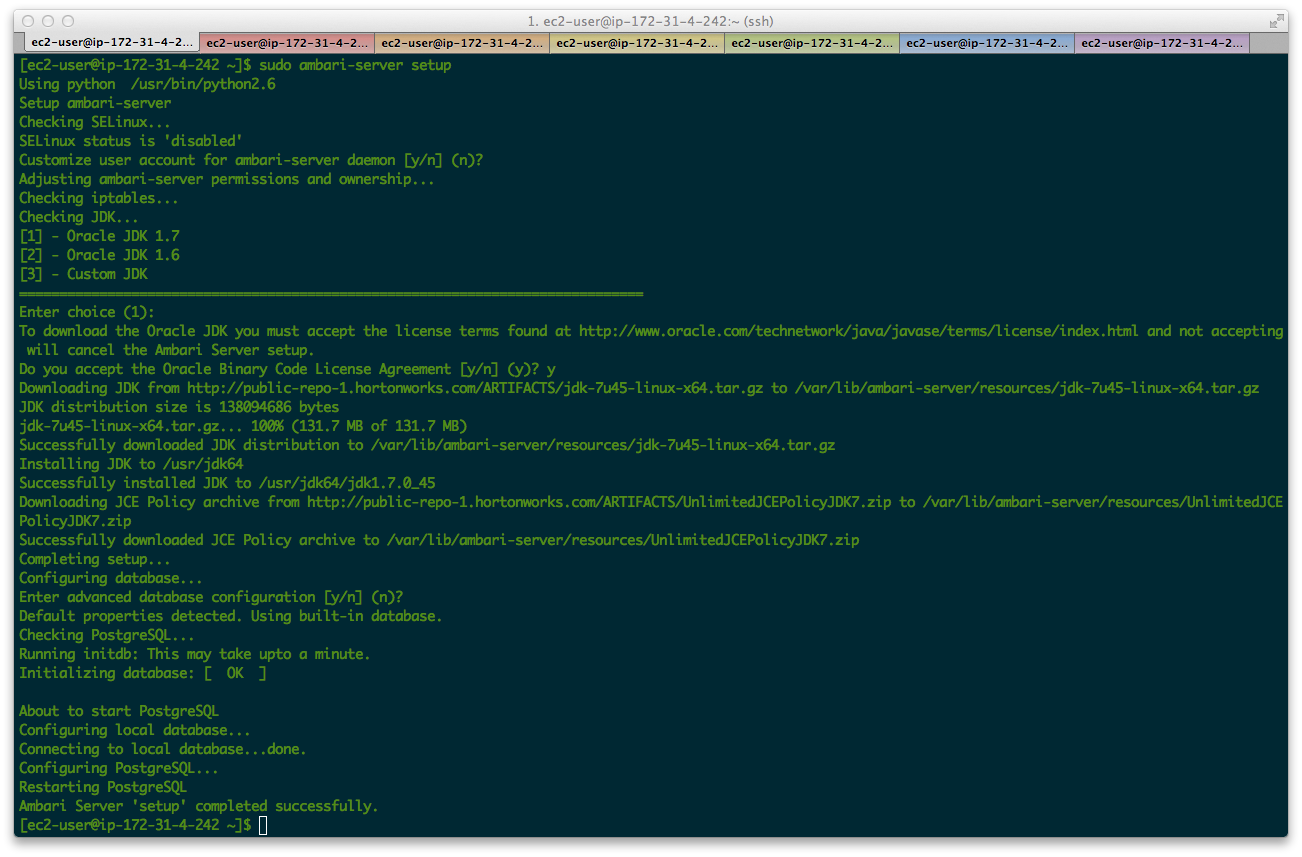
Once we have the repo in place, install Amabari from the Yum repo.

[ambari ~]$ yum install ambari-server



The next step is to setup the Ambari Server. We don’t want to customize any user accounts or the Ambari-server daemon to hit enter to skip this. Ambari will give you the option to use and existing JDK or to download and install JDK 1.6 or JDK 1.7. We will be using JDK 1.7 so select option 1. We do not want to do any advanced database setup so just hit enter here to skip this as well. Ambari will install and setup a copy of PostgreSQL for us.

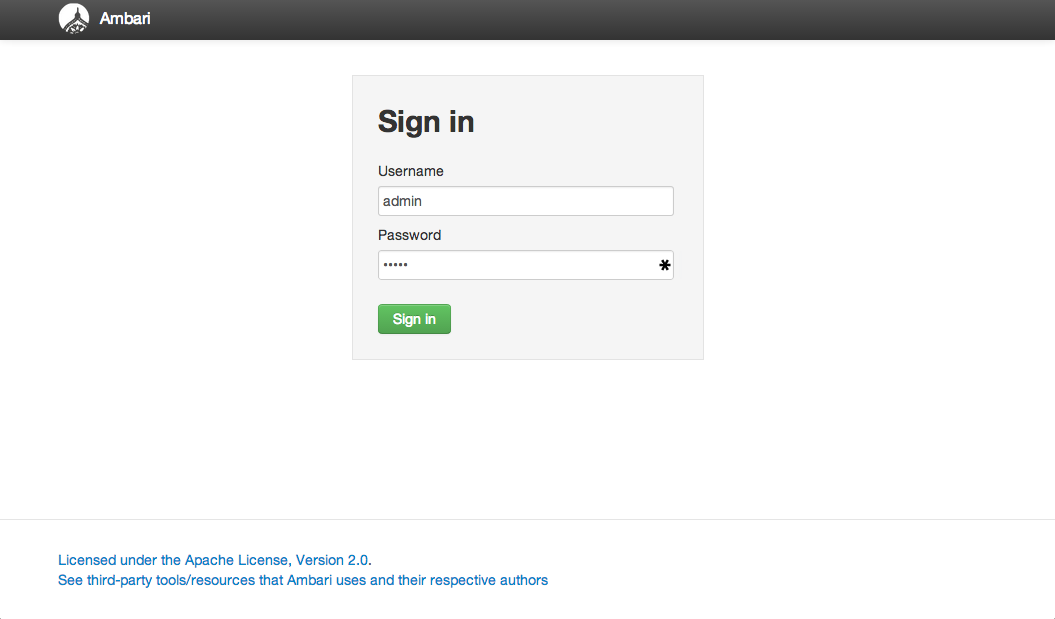
[ambari ~]$ sudo ambari-server setup



If the setup went well, and why shouldn’t it, the last step is to start the Amabari Server

[ambari ~]$ sudo ambari-server start

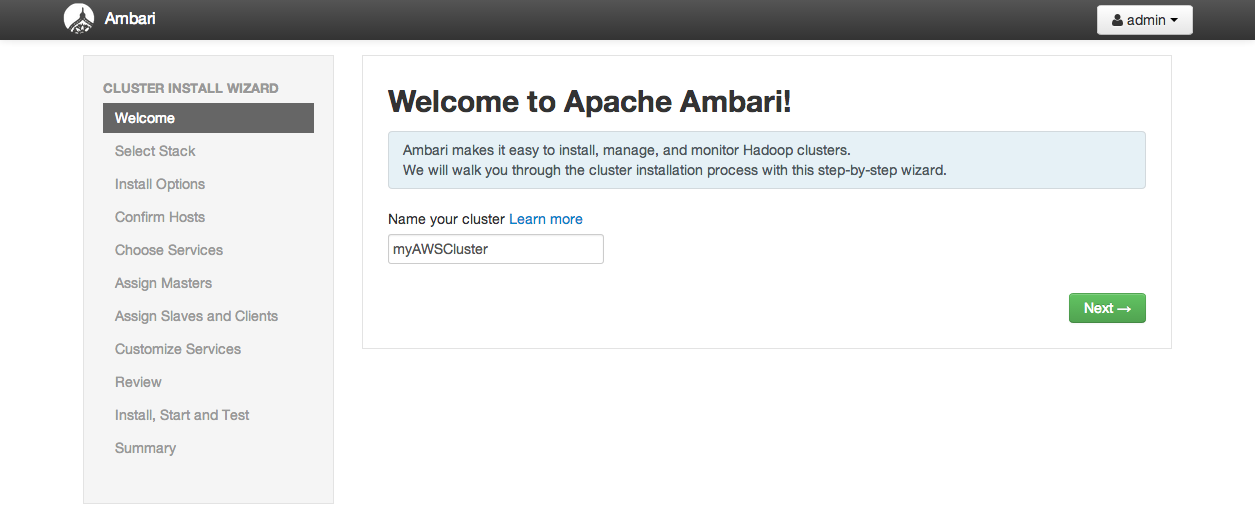
Give Ambari a minute or two to startup. Open a browser window and point it to http://{your.ambari.server}:8080. The default username and password are admin:admin You should see the following:



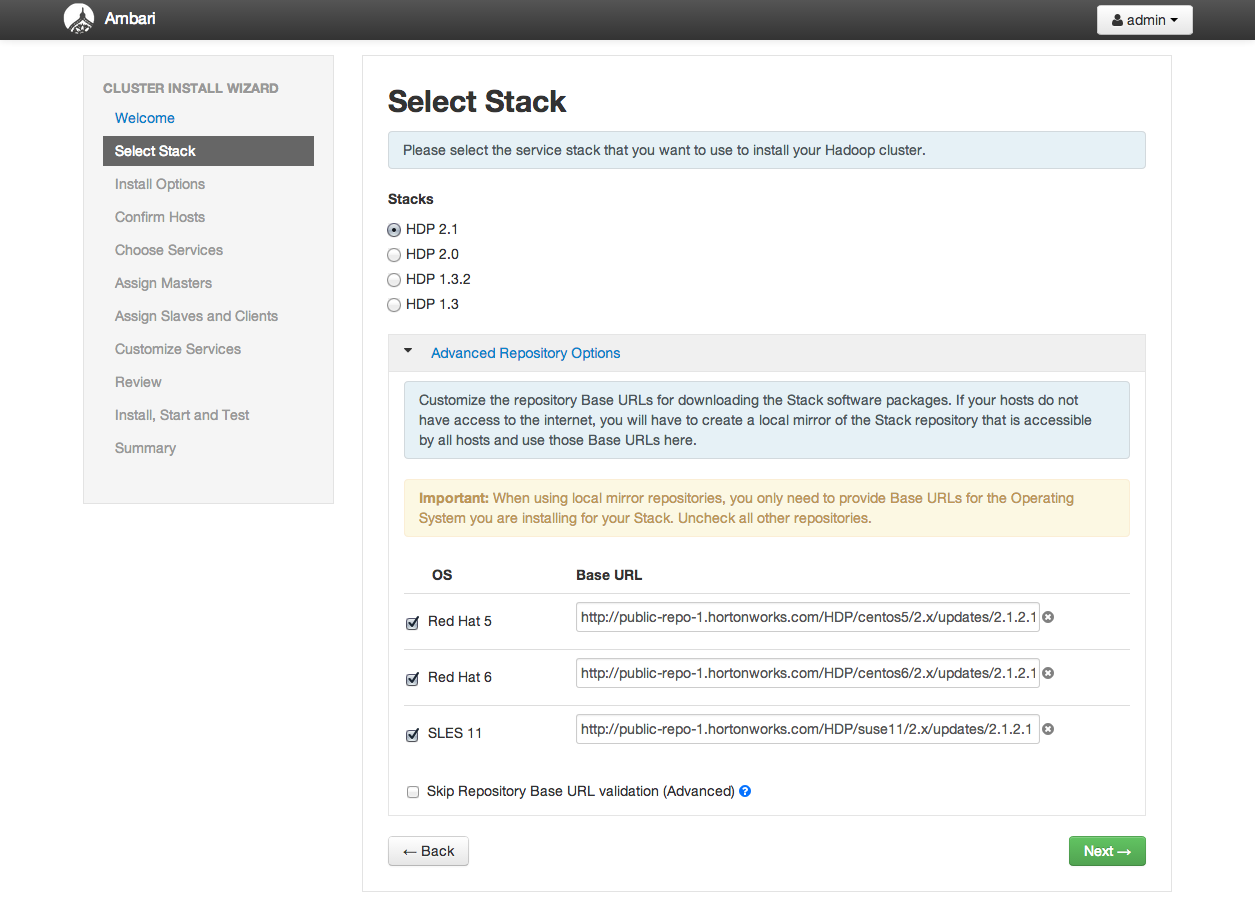
Ok. We are making headway. Now lets install that cluster.

## Step 5 – Install HDP

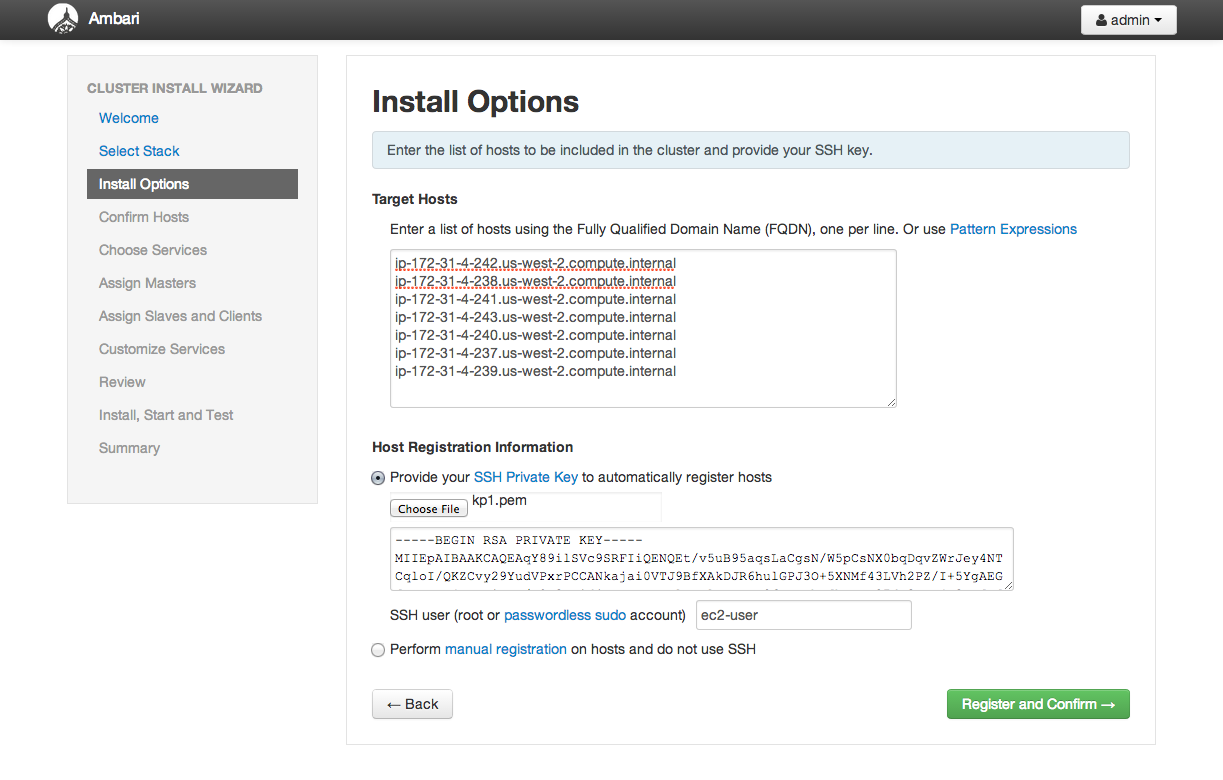
For a new cluster, the Ambari install wizard displays a Welcome page in which you define a cluster name. Type a name for the cluster you want to create. Use no white spaces or special characters in the name. Hit “Next”.



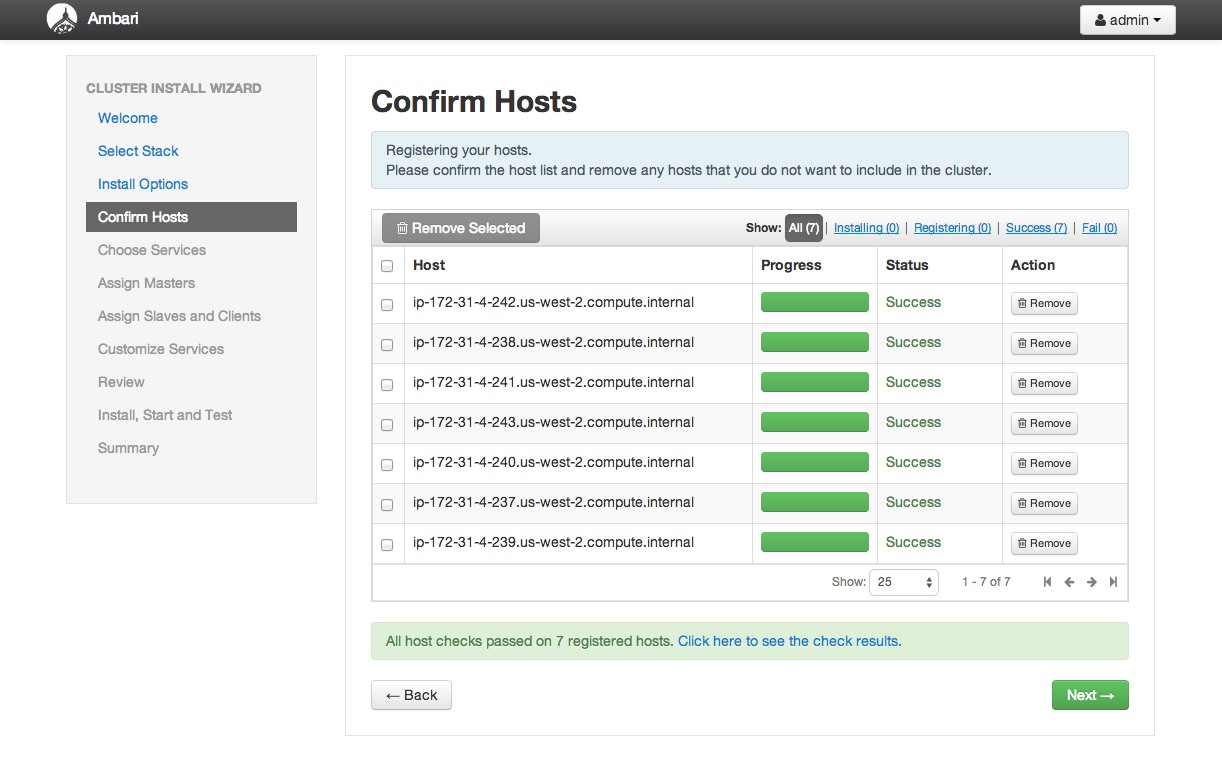
We will be installing a HDP 2.1 cluster. The repository is already set for us and we should not have to change it. If for some reason you do please refer to the HDP installation docs for how to do this. Select that form the choices and hit “Next”



Here we will enter the hostnames for all the nodes in our cluster. You need to supply the FQDN of each of your hosts. The wizard also needs to access the private key file we used. Use the Target Hosts text box to enter your list of host names, one per line. Be sure to use the **internal private DNS FQDN hostnames**. Ambari will need passwordless sudo SSH access to all the nodes so we will also need to supply our private key here. Select the private key file you have been using. Hit the “Register and Confirm” button.

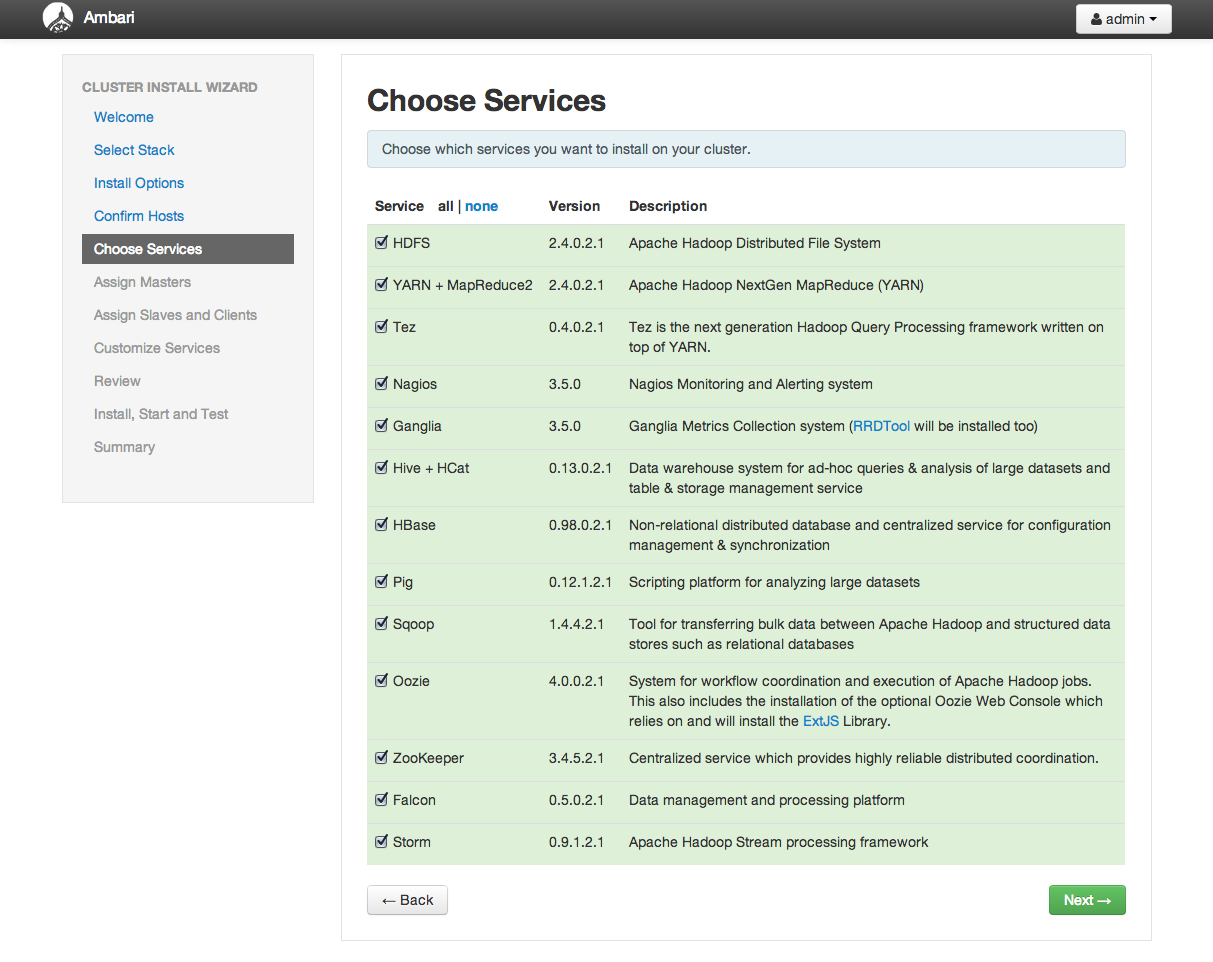


Ambari will now run some checks on all the hosts if all goes well you should see:

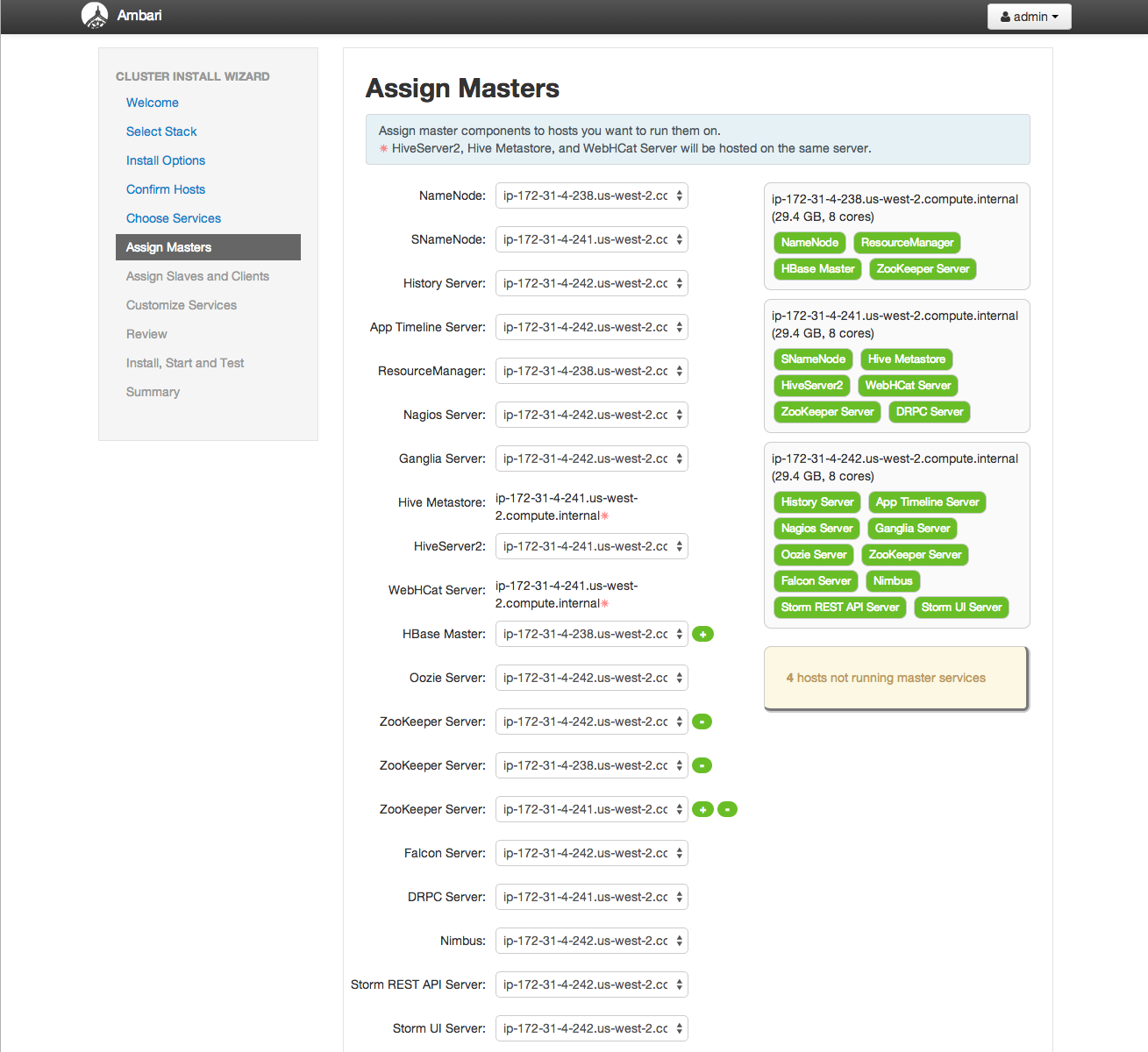


Hit “Next”.

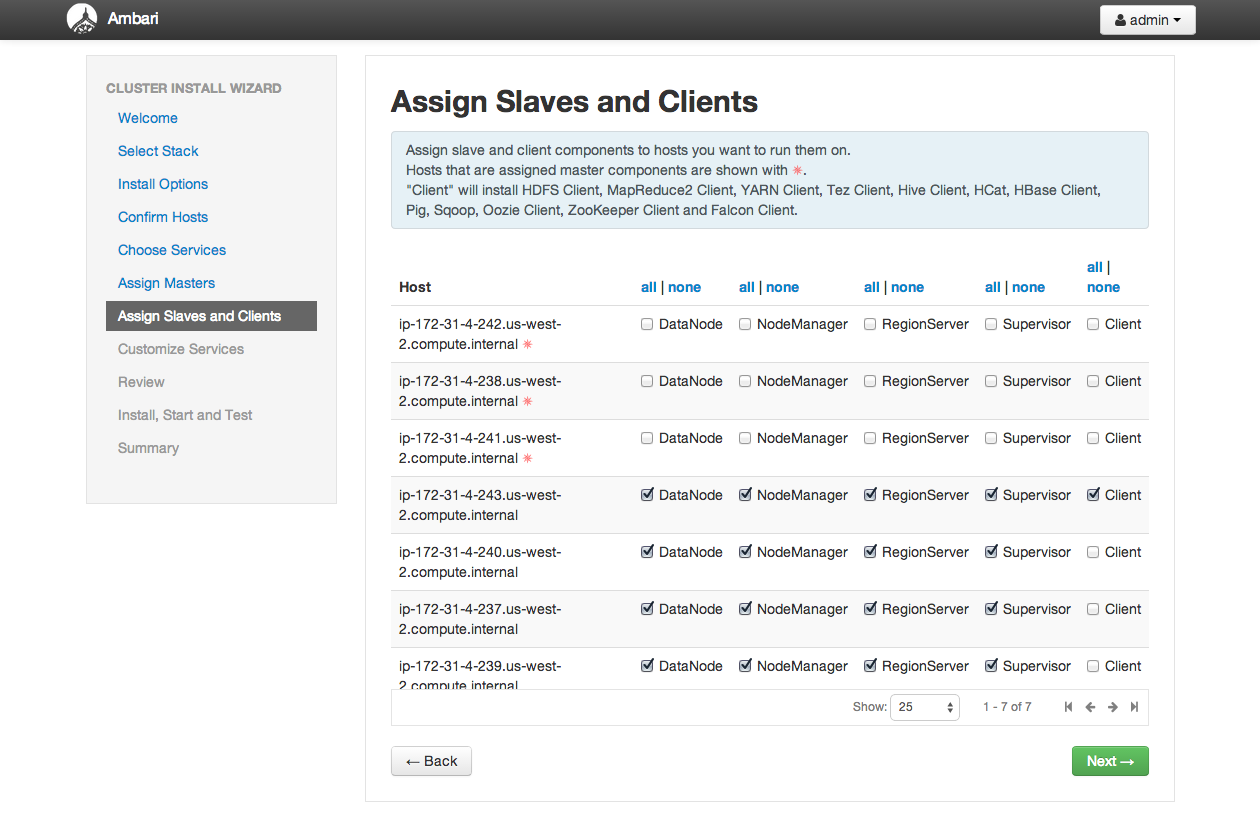
The next step is to select the services we want to have installed on this cluster. For our example we will be installing everything, but feel free to install only the services you want.



The next step is to assign the master services to our master cluster nodes. By default Ambari will more or less randomly assign these servers spread out amongst all the cluster hosts. But for our cluster we have three master nodes: nn1, nn2 and ambari. The other four hosts will be used as our worker or data nodes. As a side note you want to have at least 4 data nodes in any cluster. The reason is simple; if we only have three data nodes and have 3x replication in HDFS, if one of our nodes goes down all our HDFS blocks will be in an under-replicated state. Let’s assign all the master services to those hosts.



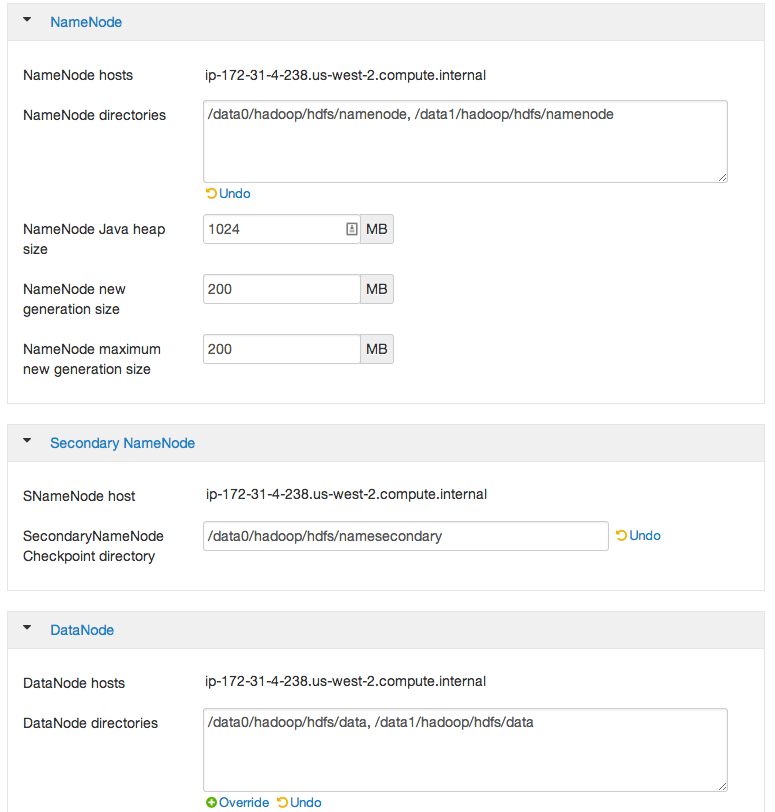
Once you have the distribution of services on the master nodes you want hit the “Next” button to setup the client services. Set up the remaining nodes to handle all data services. Although I only checked one node to be a client is ok to select them all to be clients.



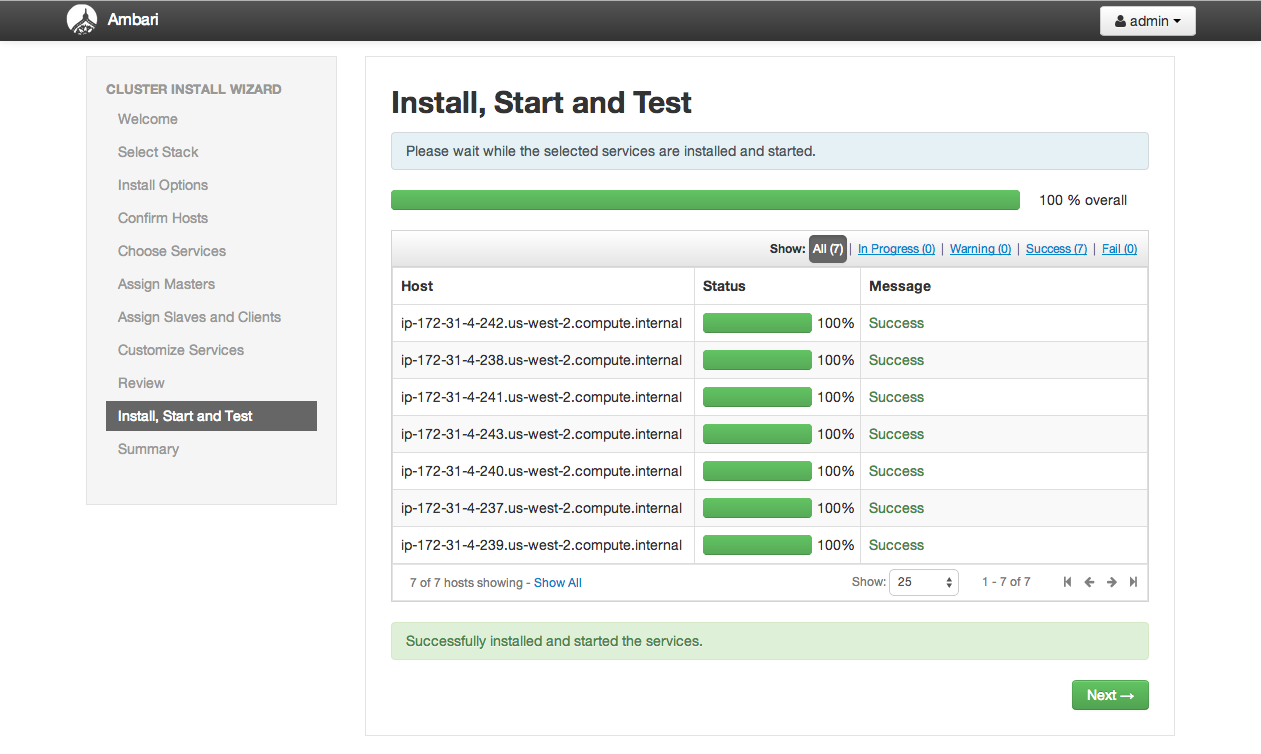
Ambari gives us the opportunity to customize our services upon install. You will notice the three red tabs below for Hive, Oozie, and Nagios. These are just asking for database username and passwords for these services. You can use whatever you want but just remember what they are. I use the following:

* Hive – hive:hive
* Oozie – oozie:oozie
* Nagios – nagios:nagios

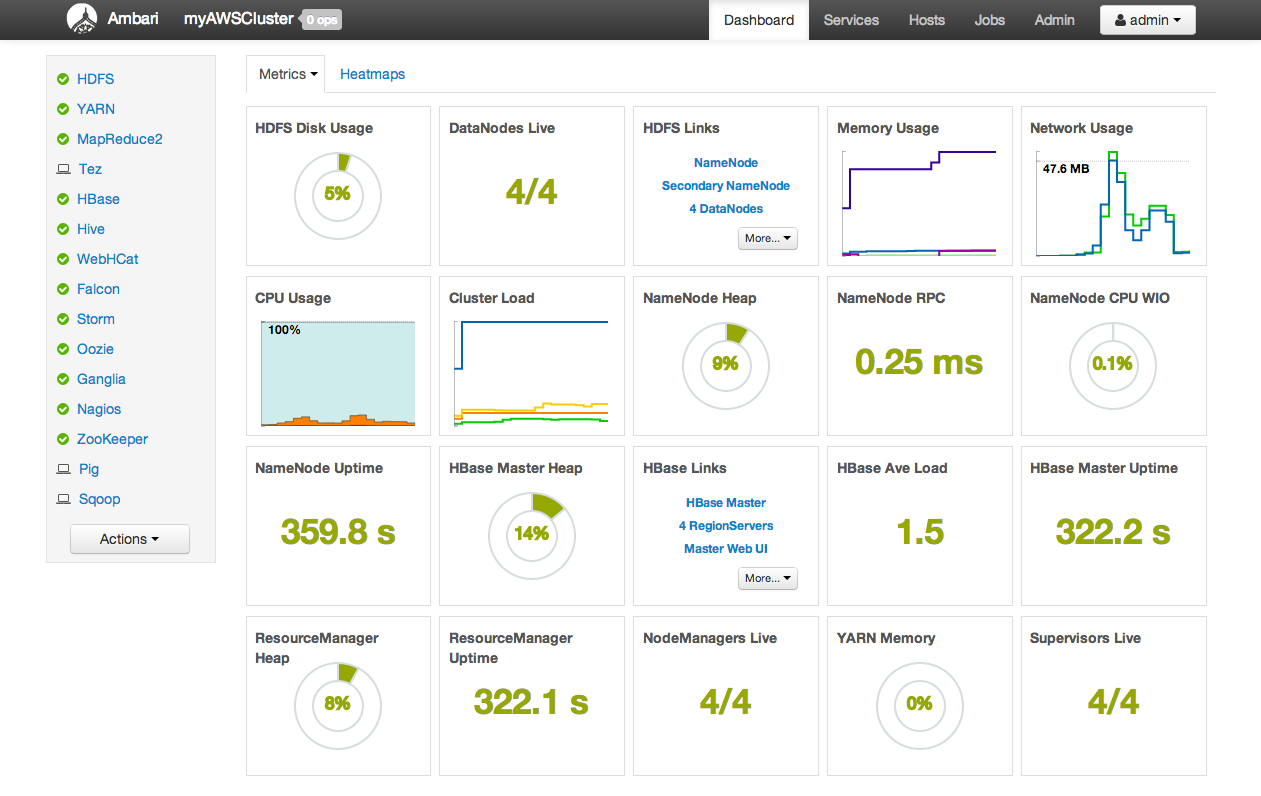
One other change we will need to make here are the NameNode Directories, Secondary NameNode Checkpoint Directory and DataNode directories we will be using for HDFS to take advantage of the giant partitions we have available to us on /data0 and /data1



Hit “Next “ to review our configuration. Double check everything to make sure it is ok and then hit the “Deploy” button. You should be rewarded with the following:



If the light is green the trap is clean. GO back to the Ambari homepage to see your cluster.



Congratulations you have now successfully installed a HDP2.1 cluster on Amazon EC2. Now start running jobs. I would suggest you take a look at the Hortonworks Tutorials at <http://hortonworks.com/tutorials/> and follow along. Good luck and Happy Hadooping!

# Appendix

## Scripting It Out

### PrepareNode.sh

#!/bin/bash

# A Bash script to prepare an EC2 node for HDP installation

# v1.1

# 27JUN14

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

echo "Starting Prepare Host"

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

#set umask

echo -e "\nSetting Umask to 022 in .bashrc"

umask 022

echo "umask 022" ? ~/.bashrc

#disable SELinux

echo -e "\nDisabling SELinux"

sudo sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/selinux/config

#Turn on NTPD

echo "Setting up NTPD and syncing time"

#Need to add a check to see if NTPD is installed. If not install it

chkconfig ntpd on

ntpd -q

service ntpd start

# Turn off autostart of iptables and ip6tables

echo -e "\nChecking ipTables and ip6table are off"

service iptables stop

sudo chkconfig iptables off

service ip6tables stop

sudo chkconfig ip6tables off

#Make filesystems on partitions

echo -e "\nMaking Filesystems"

echo -e "\nPartition: xvdb"

sudo mkfs -t ext4 /dev/xvdb

echo -e "\nPartition: xvdc"

sudo mkfs -t ext4 /dev/xvdc

echo -e "\nPartition: xvdf"

sudo mkfs -t ext4 /dev/xvdf

echo -e "\nPartition: xvdg"

sudo mkfs -t ext4 /dev/xvdg

#Make Needed Directories

echo -e "\nBacking up /var/log to /var/log-original"

mv /var/log /var/log-original

echo -e "\nMaking /var/log, /data0 and /data1"

mkdir /var/log /data0 data1

#Add entries to /etc/fstab

echo -e "\nBacking up fstab and adding entries to /etc/fstab"

cp /etc/fstab /etc/fstab.bak

sudo sed -i '$ a/dev/xvdc /var/log ext4 defaults 0 0' /etc/fstab

sudo sed -i '$ a/dev/xvdf /data0 ext4 defaults,noatime 0 0' /etc/fstab

sudo sed -i '$ a/dev/xvdg /data1 ext4 defaults,noatime 0 0' /etc/fstab

#Set Swapiness

echo -e "\nSetting Swapiness to 0"

echo 0 | sudo tee /proc/sys/vm/swappiness

echo vm.swappiness = 0 | sudo tee -a /etc/sysctl.conf

#Turn on NSCD

echo -e "\nTurning on NSCD"

chkconfig --level 345 ncsd on

ncsd -g

#Set File Handle Limits

echo -e "\nSetting File Handle Limits"

echo hdfs – nofile 32768 >> /etc/security/limits.conf

echo mapred – nofile 32768 >> /etc/security/limits.conf

echo hbase – nofile 32768 >> /etc/security/limits.conf

echo hdfs – nproc 32768 >> /etc/security/limits.conf

echo mapred – nofile 32768 >> /etc/security/limits.conf

echo hbase – nofile 32768 >> /etc/security/limits.conf

#Tune HDFS Disks

echo -e "\nTuning HDFS disk partitions"

tune2fs –m 0 /dev/xvdf

tune2fs –m 0 /dev/xvdg

echo -e "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

echo "Prepare Nodes COMPLETE!"

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

### Parallelizing Node Setup

The script above will run on all nodes. However doing this serially with a bash for loop will take some time. It is strongly recommended that you parallelize this operation whenever possible. The best way to do this is to install parallel-ssh form Google Code at <https://code.google.com/p/parallel-ssh/> . Once that is doen we can use the following script to launch the node setup process in parallel.

#!/bin/bash

echo "Copying prepareNodes.sh to nodes"

pscp -v -l root -h cluster-hosts.txt -x " -oStrictHostKeyChecking=no -i keys/id\_rsa" prepareNode.sh /root/prepareNodes.sh

echo "Starting prepareNodes on each node"

pssh -v -t 0 -l root -h cluster-hosts.txt -x "-t -t -oStrictHostKeyChecking=no -i keys/id\_rsa" 'chmod u+x prepareNodes.sh && ./prepareNodes.sh >> prepareNode.log && sudo reboot'

## Why Hortonworks for Hadoop?

Founded in 2011 by 24 engineers from the original Yahoo! Hadoop development and operations team, Hortonworks has amassed more Hadoop experience under one roof than any other organization. Our team members are active participants and leaders in Hadoop development, with proven expertise in designing, building and testing the core of the Hadoop platform. We have years of experience in Hadoop operations and are best suited to support your mission-critical Hadoop project.

For an independent analysis of Hortonworks Data Platform, download the report entitled *Forrester Wave™: Big Data Hadoop Solutions, Q1 2014* from Forrester Research.

## About Hortonworks

Hortonworks develops, distributes and supports a completely open Apache™ Hadoop® data platform. Our team comprises the largest contingent of builders and architects within the Hadoop ecosystem who represent and lead the broader enterprise requirements within these communities. The Hortonworks Data Platform provides an open platform that deeply integrates with existing IT investments upon which enterprises can build and deploy Hadoop-based applications. Hortonworks has deep relationships with the key strategic data center partners that enable our customers to unlock the broadest opportunities from Hadoop. For more information, visit [www.hortonworks.com](http://www.hortonworks.com).