HADOOP Cluster Setup

-

setting up a HADOOP cluster using Ambari and CentOS 7

By Henrik B, Sørensen @ Beech Grove
Updated November 7, 2016

Version 1.0

Table of Contents

What is Hadoop?

The short form :

Hadoop is an ecosystem for cluster computing. Cluster computing is the term used for connecting multiple computers together in a huge ”computer” in order to parallel process computational tasks. E.g. bank sometimes connect there computers in order to perform calculations. Hadoop and assistant frameworks offers a way of collecting, importing and processing data including failover.

The official version is :

> The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

Purpose

The purpose of this document is to describe the process of installing a Hadoop Cluster setup using Ambari. The software used in this process will only be Open Source and / or free of charge and / or own developed software / scripts.

About the author

Henrik B. Sørensen holds a Bachelors Degree in Electronics Engineering and has worked for the last 16 years in developing software including distributed computing systems for financial institutes.

His work in data measurement and processing has lead him over to Hadoop.

He can be contacted at info@beech-grove.eu for comments regarding this tutorial. This email is not to be used for asking for support of any kind.

Disclaimer

This document is provided as is. No guarantees of any kind is given. Also, it is highly recommended to consult documentation / best practices for e.g. handling security. The document is meant as a way of getting started with Hadoop on a cluster setup and not necessarily for production.

Prerequisites

In order to install Hadoop some things are necesarry.

First of all, you need somewhere to install the servers. It may be physical or virtual machines. In this document, I'll use a number of virtual machines located on 2 physical servers running Xen Server. Given the tutorial of the previous document ("HADOOP Cluster Setup - setting up a HADOOP cluster Ambari and CentOS 6") and a multitude of tutorials online, I will save space and not go through the creation of the virtual machines. One note, though. I use XenCenter to control my virtual servers. For some reason, the XenCenter and CentOS 7 graphical installer do not play well together. I found a workaround : Hit Tab when the installation begins and swap `quiet` with `inst.text`. This will give a console / text installation process.

Server Environment

My environment[1] is as follows :

Name	Address	Description
Utilities (utilities)	192.168.1.115	A file server holding a number of script files and demos
Ambari (ambari.cluster)	192.168.1.119	Server hosting the Ambari web site.
Master A (mastera.cluster)	192.168.1.120	Primary NameNode for the cluster
Master B (masterb.cluster)	192.168.1.130	Secondary NameNode for the cluster
Clients A 01-03 (clienta01-3.cluster)	192.168.1.121-123	3 clients in a group
Clients B 01-03 (clientb01-3.cluster)	192.168.1.131-133	3 clients in a group

The naming convention is based on which physical server, the virtual machines are located on.

Each of the machines need to have Java installed (I use the Java-1.8.0-openjdk) and have the JAVA_HOME environment variable set to the folder containing the Jave Runtime Engine.

```
/usr/lib/jvm/jre-1.8.0-openjdk
```

Logging on to the Ambari server, we need to setup a few things. In order to enable the EPEL repository, setup the SSH key and install the Parallel Distributed Shell (pdsh – for remote shell commands), we need to run the following script (SetupServerEnv.sh) :

```
#!/bin/bash
wget http://dl.fedoraproject.org/pub/epel/7/x86_64/e/epel-release-7-8.noarch.rpm
rpm -ivh epel-release-7-8.noarch.rpm
yum --enablerepo=epel -y install sshpass

ssh-keygen -t rsa

yum -y install pdsh-rcmd-ssh
```

The script also installs a sshpass utility which is used for passing the password to an ssh session.

1 Please note, that the names and addresses used in this document are dummies and not actual addresses / names.

Setting up the remote logins, I use a little script (CopyHostFiles.sh):

```
#!/bin/bash
cat $1 | while read line
do
  host=`echo $line | gawk  '{print $2}'`

  #Skip localhost
  if [ $host != "localhost" ]
  then
    #Skip utilities server
    if [ $host != "utilities" ]
    then
      if [ $host != $HOSTNAME ]
      then
        echo "Processing $host"
        ssh-keyscan  $host >> ~/.ssh/known_hosts
        sshpass -p $2 ssh-copy-id -i /root/.ssh/id_rsa.pub $host
        scp /etc/hosts root@$host:/etc/hosts
      fi
    fi
  fi

done
```

This script can off course be expanded to incorporate the Java installation and setup and / or to accommodate other requirements.

Calling this using the following syntax will run through the host file on the server (omitting localhost, current file and the utilities server), add the remote system to known hosts, enable passwordless login and copy the host file to the system using the given password[2].

```
./CopyHostFiles.sh /etc/hosts [Password]
```

2 In this case, the same root password is being used accross the entire cluster. This is NOT best practices, but for a demo purpose, this will be ok on a closed environment.

Hadoop overview

Normally, Hadoop is used in huge server systems consisting of many computers. A typical system could look like this :

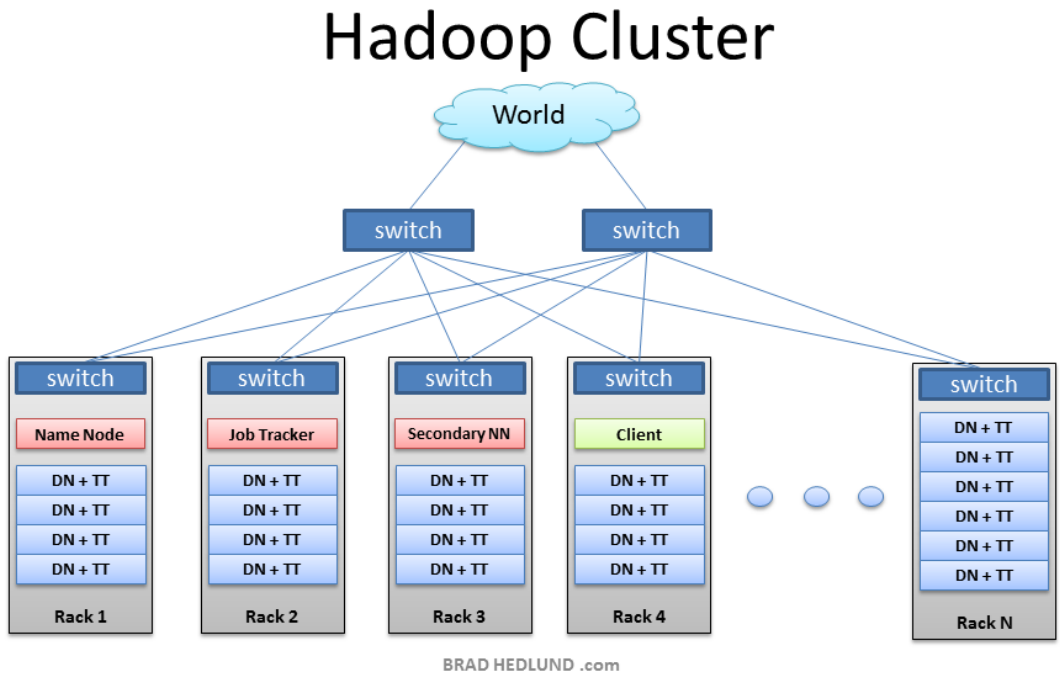

Illustration 1: Hadoop Cluster overview

Hadoop has its own filesystem called HDFS (Hadoop Distributed File System). It also is divided into 4 parts. The Master server contains 2 services called NameNode and JobTracker and is described in the official Hadoop Wiki as :

The **NameNode** is the centerpiece of an HDFS file system. It keeps the directory tree of all files in the file system, and tracks where across the cluster the file data is kept. It does not store the data of these files itself.

Client applications talk to the NameNode whenever they wish to locate a file, or when they want to add/copy/move/delete a file. The NameNode responds the successful requests by returning a list of relevant DataNode servers where the data lives.

The NameNode is a Single Point of Failure for the HDFS Cluster. HDFS is not currently a High Availability system. When the NameNode goes down, the file system goes offline. There is an optional SecondaryNameNode that can be hosted on a separate machine. It only creates checkpoints of the namespace by merging the edits file into the fsimage file and does not provide any real redundancy. Hadoop 0.21+ has a BackupNameNode that is part of a plan to have an HA name service, but it needs active contributions from the people who want it (i.e. you) to make it Highly Available.

Client applications talk to the NameNode whenever they wish to locate a file, or when they want to add/copy/move/delete a file. The NameNode responds the successful requests by returning a list of relevant DataNode servers where the data lives.

The NameNode is a Single Point of Failure for the HDFS Cluster. HDFS is not currently a High Availability system. When the NameNode goes down, the file system goes offline. There is an optional SecondaryNameNode that can be hosted on a separate machine. It only creates checkpoints of the namespace by merging the edits file into the fsimage file and does not provide any real redundancy. Hadoop 0.21+ has a BackupNameNode that is part of a plan to have an HA name service, but it needs active contributions from the people who want it (i.e. you) to make it Highly Available.

The **JobTracker** is the service within Hadoop that farms out MapReduce tasks to specific nodes in the cluster, ideally the nodes that have the data, or at least are in the same rack.

Client applications submit jobs to the Job tracker. The JobTracker talks to the NameNode to determine the location of the data. The JobTracker locates TaskTracker nodes with available slots at or near the data. The JobTracker submits the work to the chosen TaskTracker nodes. The TaskTracker nodes are monitored. If they do not submit heartbeat signals often enough, they are deemed to have failed and the work is scheduled on a different TaskTracker.

A TaskTracker will notify the JobTracker when a task fails. The JobTracker decides what to do then: it may resubmit the job elsewhere, it may mark that specific record as something to avoid, and it may may even blacklist the TaskTracker as unreliable. When the work is completed, the JobTracker updates its status.

Each of the Clients holds the DataNode and the TaskTracker :

A **DataNode** stores data in the HDFS. A functional filesystem has more than one DataNode, with data replicated across them.

On startup, a DataNode connects to the NameNode; spinning until that service comes up. It then responds to requests from the NameNode for filesystem operations.

Client applications can talk directly to a DataNode, once the NameNode has provided the location of the data. Similarly, MapReduce operations farmed out to TaskTracker instances near a DataNode, talk directly to the DataNode to access the files. TaskTracker instances can, indeed should, be deployed on the same servers that host DataNode instances, so that MapReduce operations are performed close to the data.

DataNode instances can talk to each other, which is what they do when they are replicating data.

A **TaskTracker** is a node in the cluster that accepts tasks - Map, Reduce and Shuffle operations - from a JobTracker.

Every TaskTracker is configured with a set of slots, these indicate the number of tasks that it can accept. When the JobTracker tries to find somewhere to schedule a task within the MapReduce operations, it first looks for an empty slot on the same server that hosts the DataNode containing the data, and if not, it looks for an empty slot on a machine in the same rack.

The TaskTracker spawns a separate JVM processes to do the actual work; this is to ensure that process failure does not take down the task tracker. The TaskTracker monitors these spawned processes, capturing the output and exit codes. When the process finishes, successfully or not, the tracker notifies the

JobTracker. The TaskTrackers also send out heartbeat messages to the JobTracker, usually every few minutes, to reassure the JobTracker that it is still alive. These message also inform the JobTracker of the number of available slots, so the JobTracker can stay up to date with where in the cluster work can be delegated.

The whole relationship between the various part can be illustrated this way :

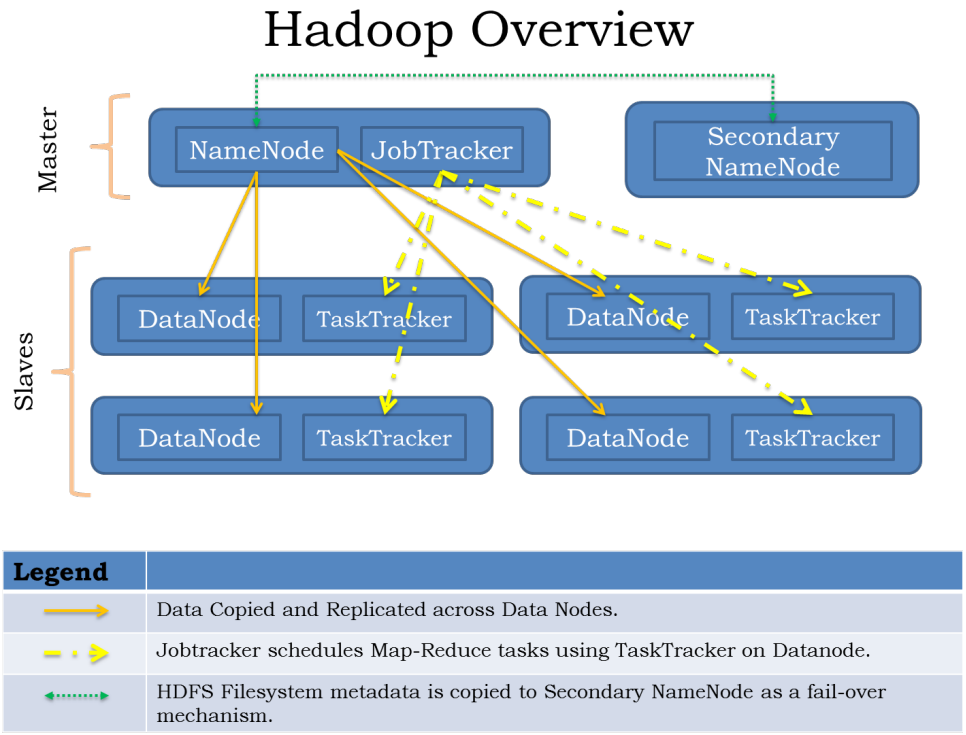

Illustration 2: Hadoop services relation

Normally, a failover strategy will be implemented having aSecondary NameNode, but we will not go into details about this.

The Hadoop eco system includes according to official wiki :

Ambari™	A web-based tool for provisioning, managing, and monitoring Apache Hadoop clusters which includes support for Hadoop HDFS, Hadoop MapReduce, Hive, HCatalog, HBase, ZooKeeper, Oozie, Pig and Sqoop. Ambari also provides a dashboard for viewing cluster health such as heatmaps and ability to view MapReduce, Pig and Hive applications visually alongwith features to diagnose their performance characteristics in a user-friendly manner.
Avro™	A data serialization system.
Cassandra™	A scalable multi-master database with no single points of failure.
Chukwa™	A data collection system for managing large distributed systems.
Hbase™	A scalable, distributed database that supports structured data storage for large tables.
Hive™	A data warehouse infrastructure that provides data summarization and ad hoc querying.
Mahout™	A Scalable machine learning and data mining library.

Pig™	A high-level data-flow language and execution framework for parallel computation.
Spark™	A fast and general compute engine for Hadoop data. Spark provides a simple and expressive programming model that supports a wide range of applications, including ETL, machine learning, stream processing, and graph computation.
Tez™	A generalized data-flow programming framework, built on Hadoop YARN, which provides a powerful and flexible engine to execute an arbitrary DAG of tasks to process data for both batch and interactive use-cases. Tez is being adopted by Hive™, Pig™ and other frameworks in the Hadoop ecosystem, and also by other commercial software (e.g. ETL tools), to replace Hadoop™ MapReduce as the underlying execution engine.
ZooKeeper™	A high-performance coordination service for distributed applications.

For the setup, we'll use Ambari, Spark and ZooKeeper and setup 2 Master Nodes (1 primary and 1 secondary) and 6 Slave Nodes.

Ambari Installation

In our case, we only need the system for developing solutions for running on Hadoop clusters and also for doing some computations meaning that we will setup up a smaller version. So, we'll set up 2 Master Servers (Name Node) and 6 Workers (Data Nodes) plus a single machine for running the Ambari server.

Using the virtual machines, we now need to setup the systems. Provided we have the 9 virtual images as mentioned in the Server Environment section, we continue to install Ambari on them. PDSH enables us to run commands on other hosts. Hence, we run the InstallAmbari.sh script providing the host file as an argument :

```
#!/bin/bash
cat $1 | while read line

do
  host=`echo $line | gawk  '{print $2}'`

  #Skip localhost
  if [ $host != "localhost" ]
  then
    #Skip utilities server
    if [ $host != "utilities" ]
    then
      if [ $host != $HOSTNAME ]
      then
        echo "Processing $host"
             #Retrieve the Repository files
             pdsh -w $host "yum -y install wget"
             pdsh -w $host "wget http://public-repo-
1.hortonworks.com/ambari/centos7/2.x/updates/2.1.2.1/ambari.repo -O
/etc/yum.repos.d/ambari.repo" | sort
             #Install the Ambari Agent
             pdsh -w $host "yum -y install ambari-agent" | sort
             #Setup the Init file to point towards the ambari.cluster server
             pdsh -w $host "sed -i 's/hostname=localhost/hostname=ambari.cluster/g'
/etc/ambari-agent/conf/ambari-agent.ini"
             pdsh -w $host "sed -i 's/hostname=localhost/hostname=ambari.cluster/g'
/etc/ambari-agent/conf/ambari-agent.ini"
             pdsh -w $host "chkconfig ambari-agent on"
             pdsh -w $host "service ambari-agent restart"
             pdsh -w $host "systemctl stop firewalld"
             pdsh -w $host "systemctl disable firewalld"
      fi
    fi
  fi

done

wget http://public-repo-1.hortonworks.com/ambari/centos7/2.x/updates/2.1.2.1/ambari.repo
-O /etc/yum.repos.d/ambari.repo
yum -y install ambari-agent
yum -y install ambari-server
ambari-server setup -j /usr/lib/jvm/jre-1.8.0-openjdk
ambari-server start
systemctl stop firewalld
systemctl disable firewalld
```

Executing the script will take a while – a number of files must be downloaded.

Having waited for the commands to finish the Ambari website should now be available in a browser on port 8080 :

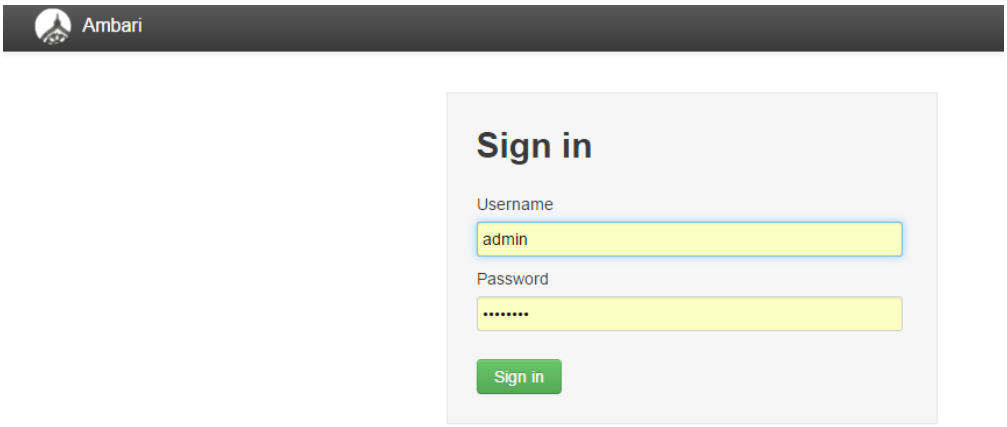

Illustration 3: Ambari, Initial login

Login using ”admin” as both password and username and change it before proceeding :

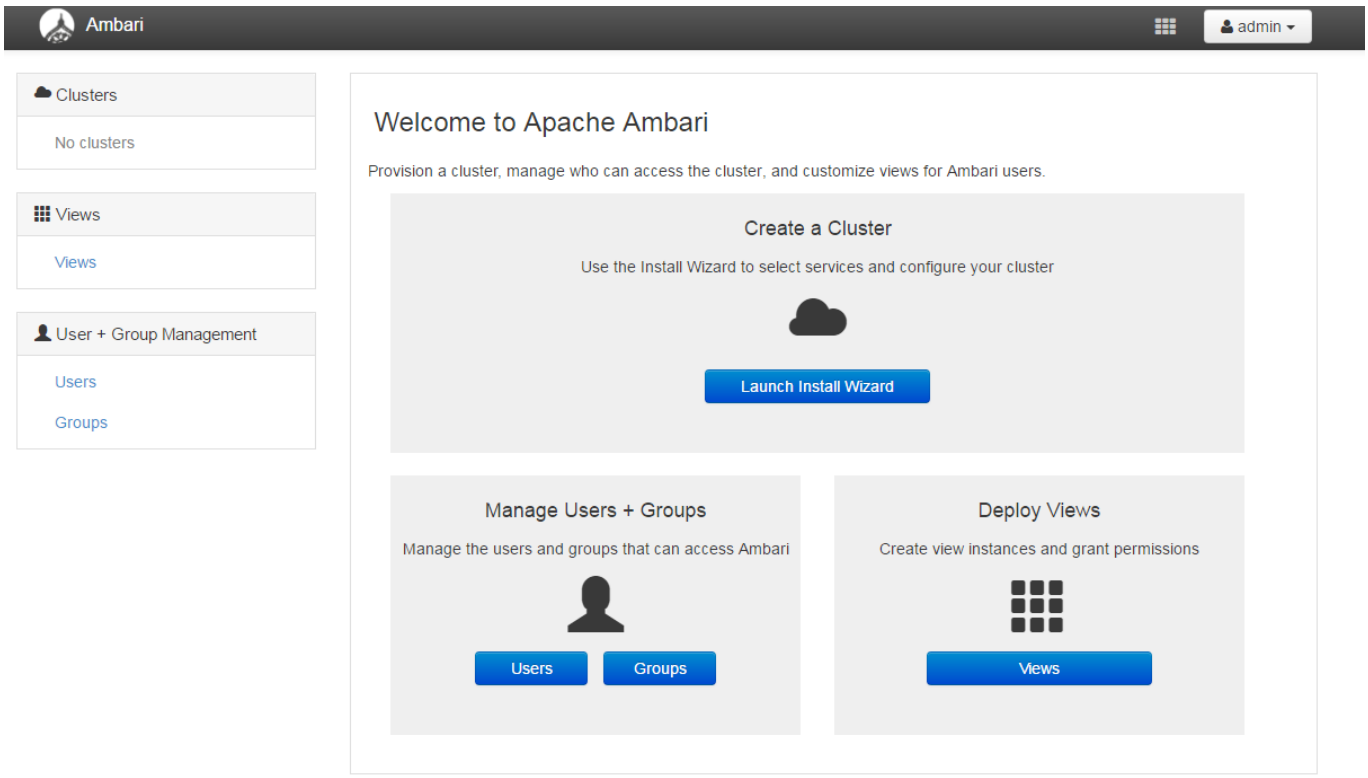

Illustration 4: Ambari, Initial welcome screen

Go to users and click the ”admin” user :

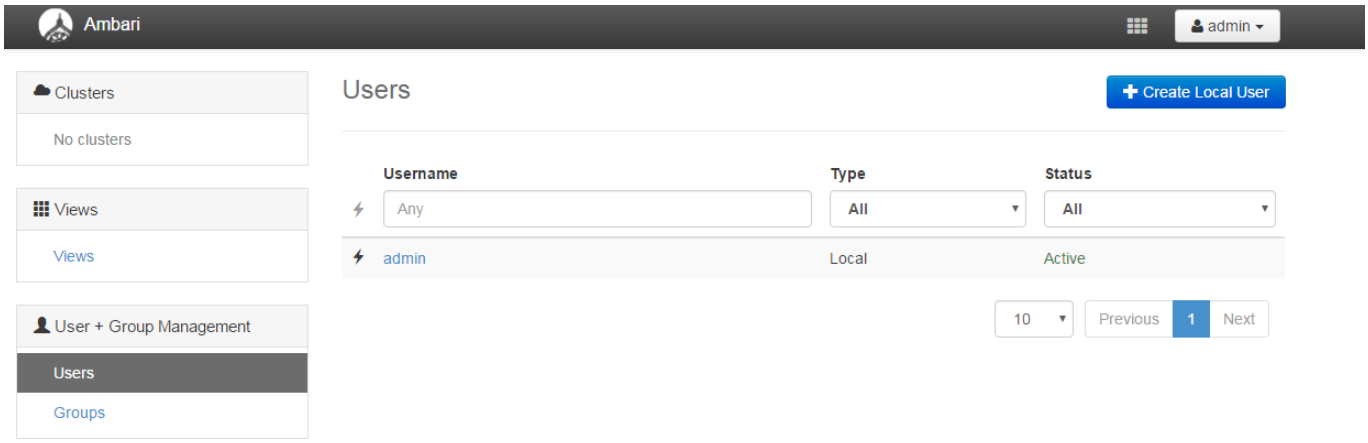

Illustration 5: Ambari, User management

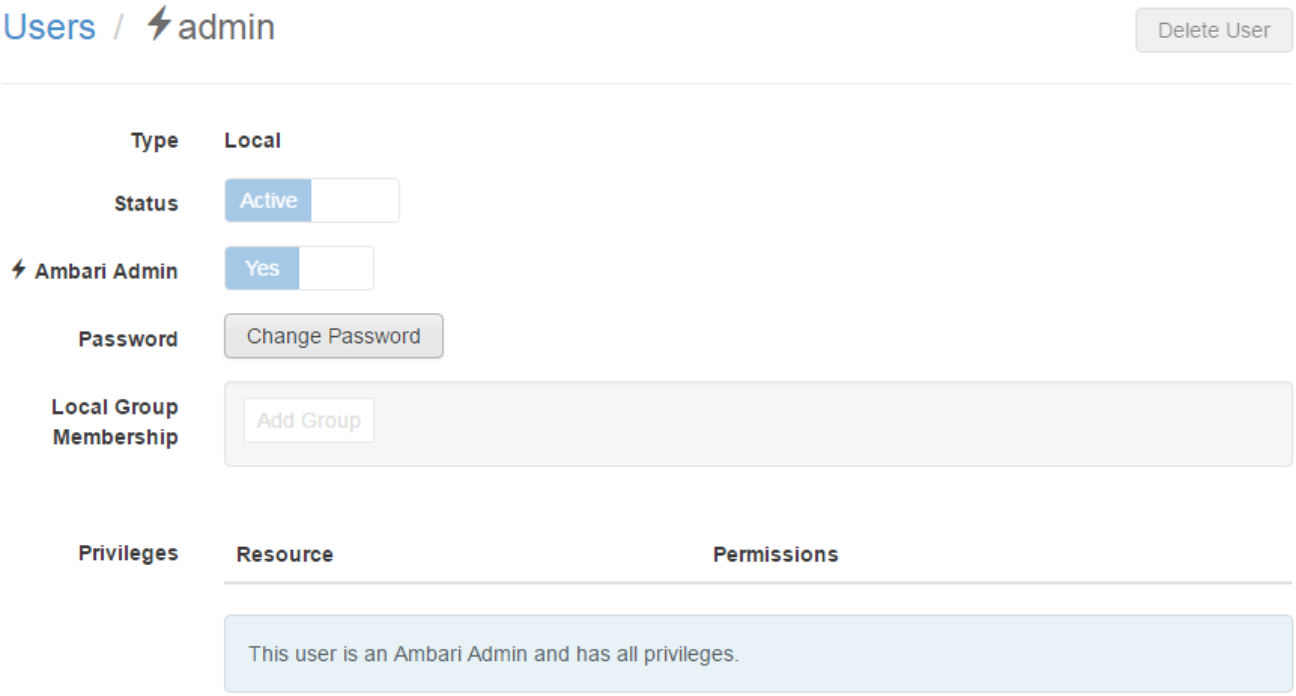

Illustration 6: Ambari, User editing

Go back to the frontpage by clicking the Ambari logo.

Now, we need to create a cluster. We click the ”Launch Install Wizard” button in order to be guided through the cluster creation :

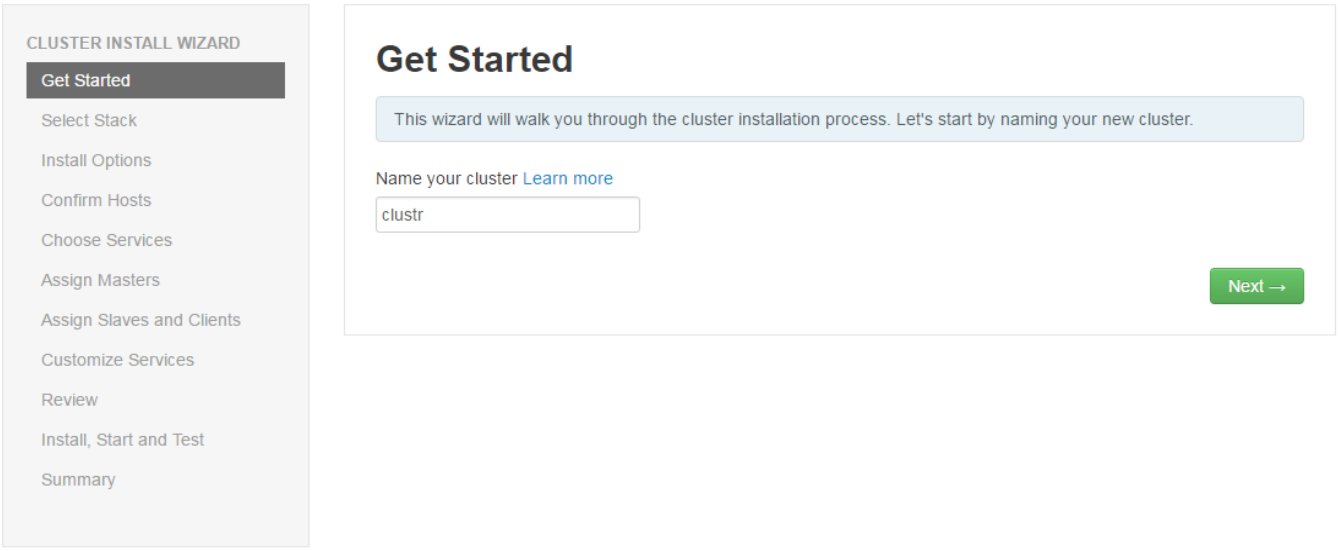

Illustration 7: Ambari Wizard, Name Cluster

First, we name the cluster (in this case ”clustr” - not very inspiring).

Next, we select the Hadoop stack (version) to use. The current installation supports up to Hadoop 2.2, so we’ll select that.

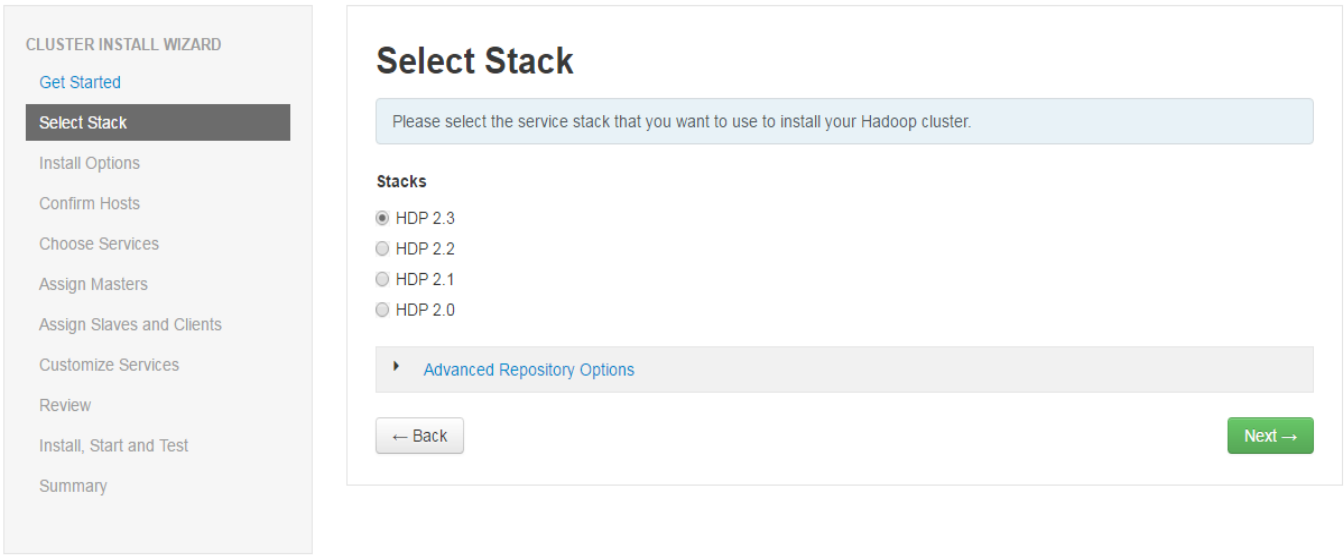

Illustration 8: Ambari Wizard, Stack selection.

In the Advanced Repository Options, I've disabled all but the RedHat7 repository, since we're using CentOS 7 as our operating system :

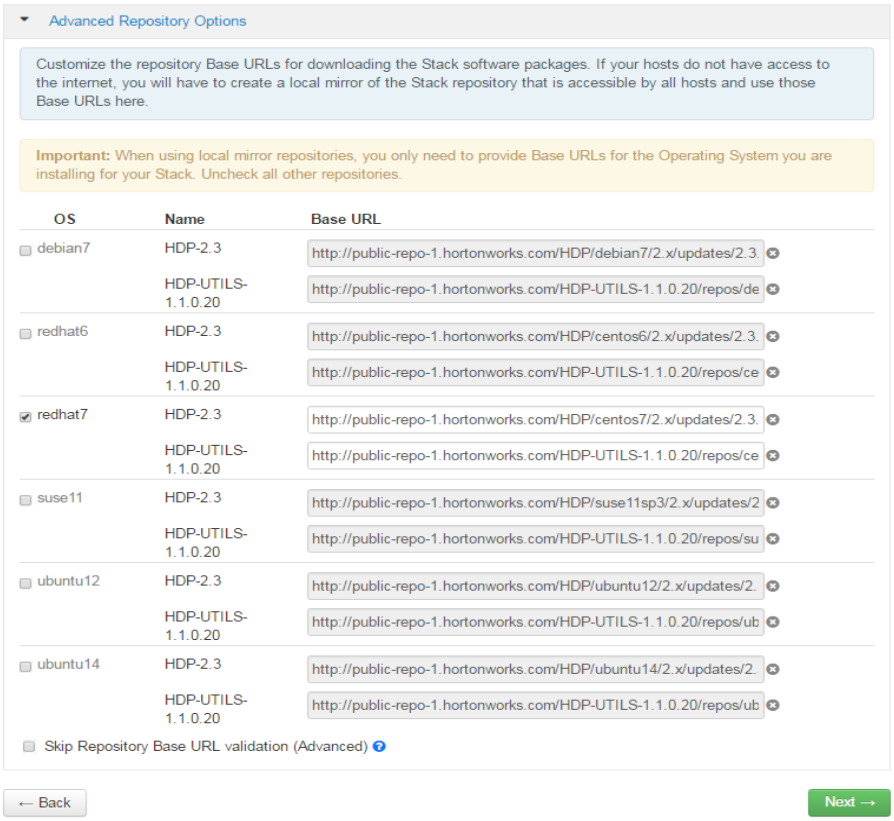

Illustration 9: Ambari Wizard, Repositories

Moving on, we have to select the hosts to install the services on.

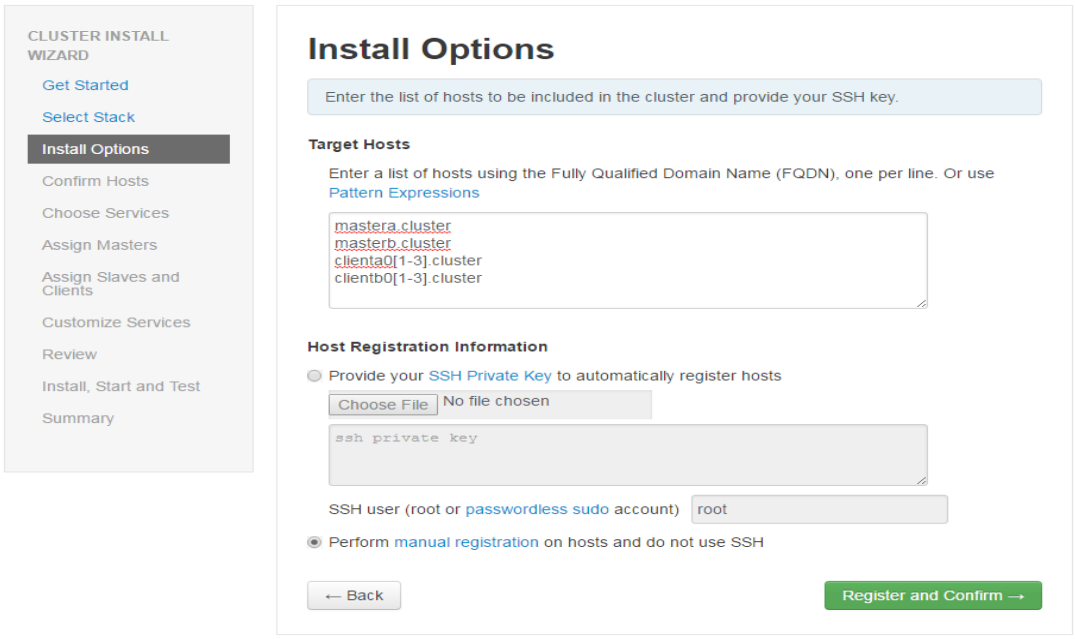

Illustration 10: Ambari Wizard, Install Options

We select all the machines running the agents (besides the one hosting the web site). Please note, that the Fully Qualified Domain Names are used. Due to our setup, we also select manual registration. A verification of the host names is shown in order to validate the patterns added :

Illustration 11: Ambari Wizard, Pattern validation

Accept the pattern and messages and proceed for the registration process :

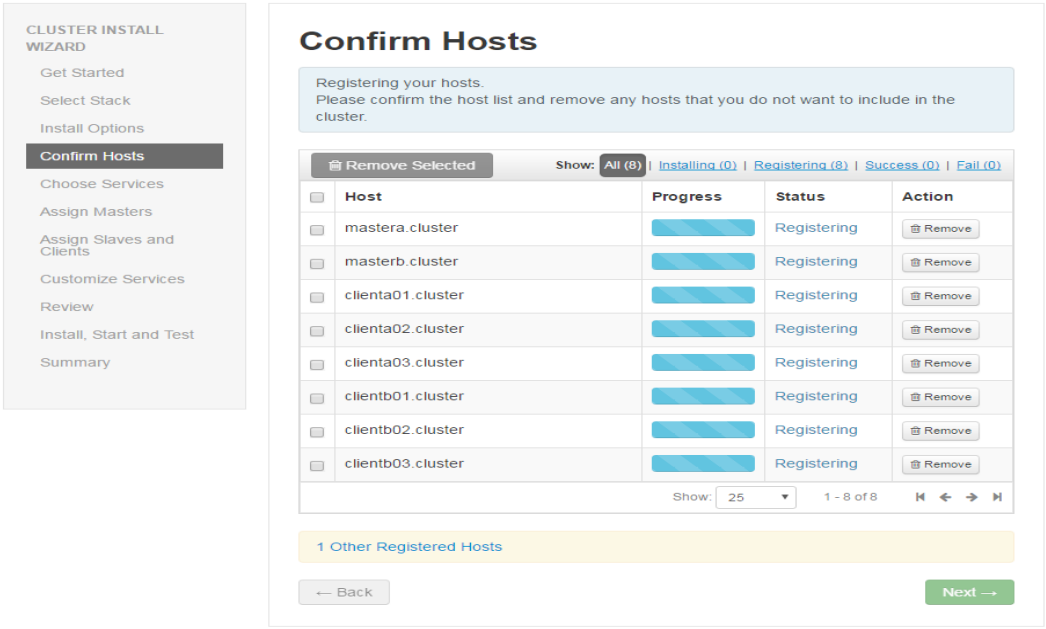

Illustration 12: Ambari Wizard, Confirm Hosts

Having performed the registration, we proceed to add services

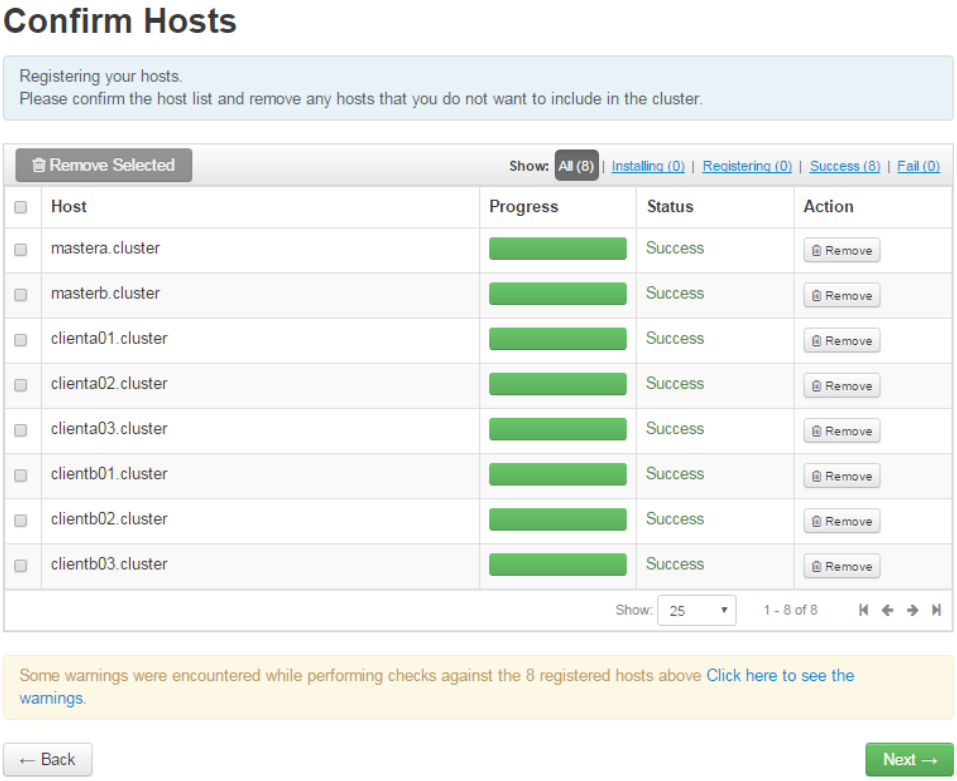

Illustration 13: Ambari Wizard, Registration Completed.

Clicking on the link will display the details. Since, we're using the CentOS 7, which uses the ntpdate service for synchronizing time, we ignore the warning :

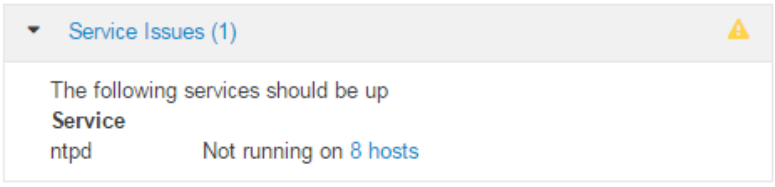

Illustration 14: Service Issues

Proceed to select the services to run on the cluster. In our case, we wish to install a bare minimum Hadoop cluster, also to prevent the installation to timeout. Later, we will go back and add additional services to run on the cluster.

So for now, we will stick to :

- HDFS
- YARN + MapReduce 2
- ZooKeeper
- Ambari Metrics

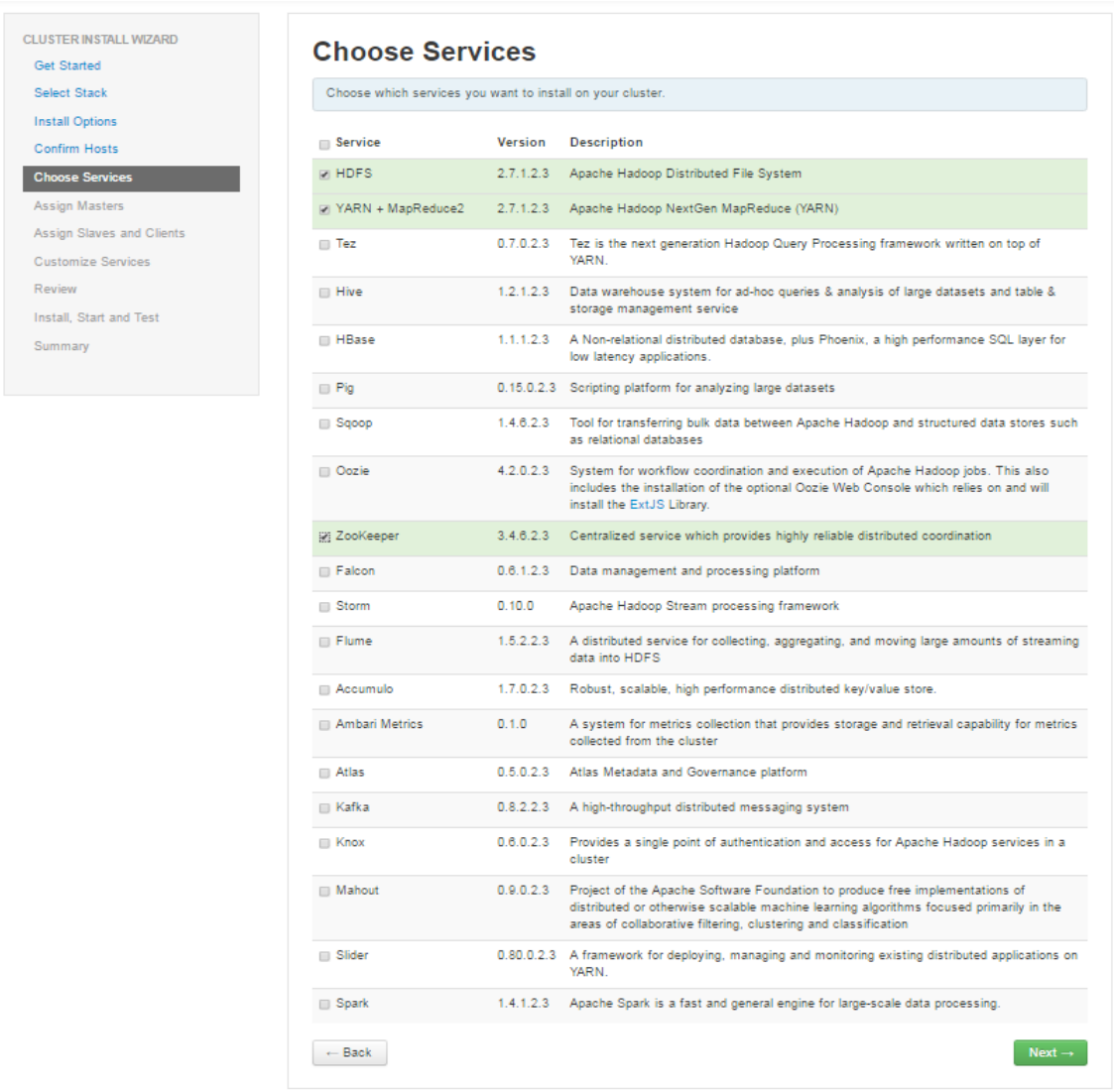

Illustration 15: Initial Service Selection

Assigning the Masters of the system. We will attempt to put the load evenly on the 2 masters (mastera and masterb).

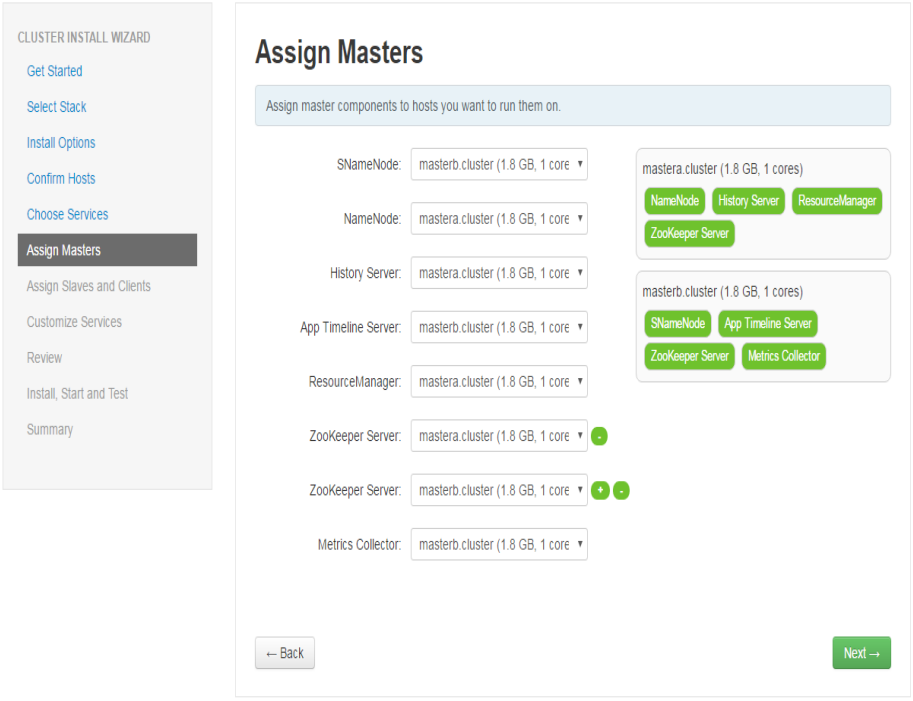

Illustration 16: Ambari Wizard, Assign Masters

Next, we setup the clients :

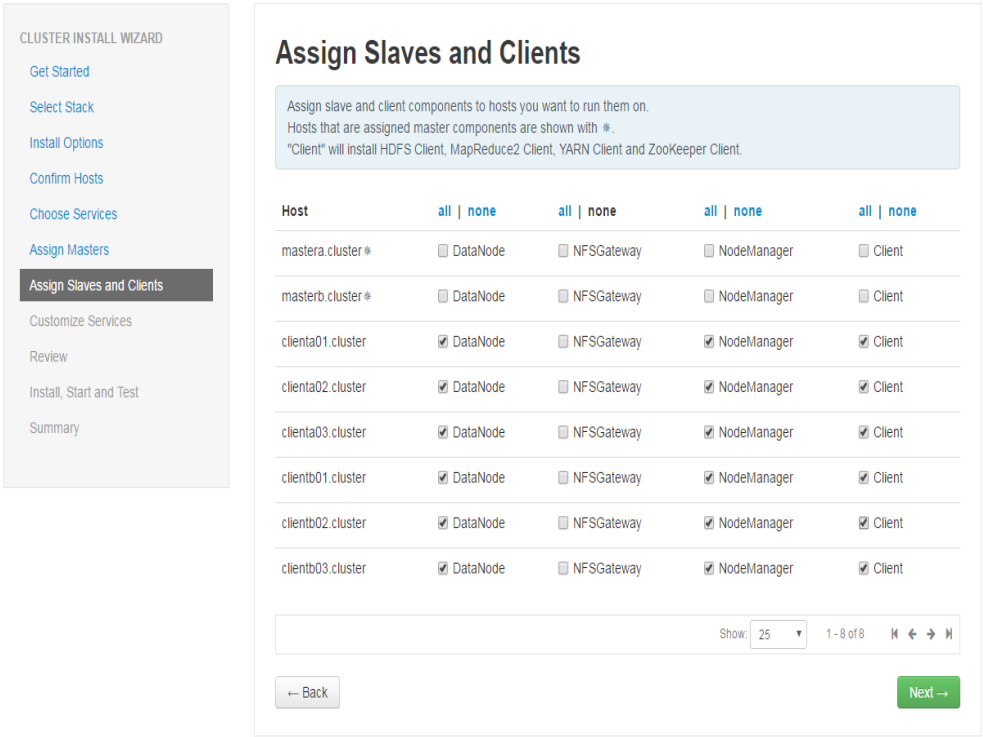

Illustration 17: Ambari Wizard, Assign Slaves and clients

We select all other machines, leaving out the NFSGateway, since we currently do not use it.

Now, we move on to customization of the services. According to the UI, all configurations have been addressed :

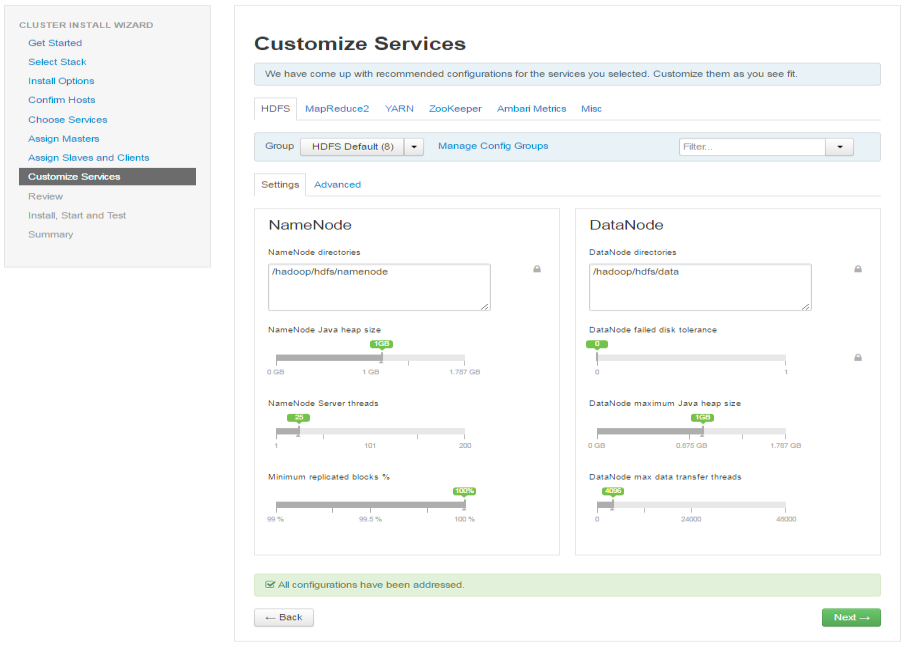

Illustration 18: Ambari Wizard

We can now review our selections and proceed, if we're OK with the settings:

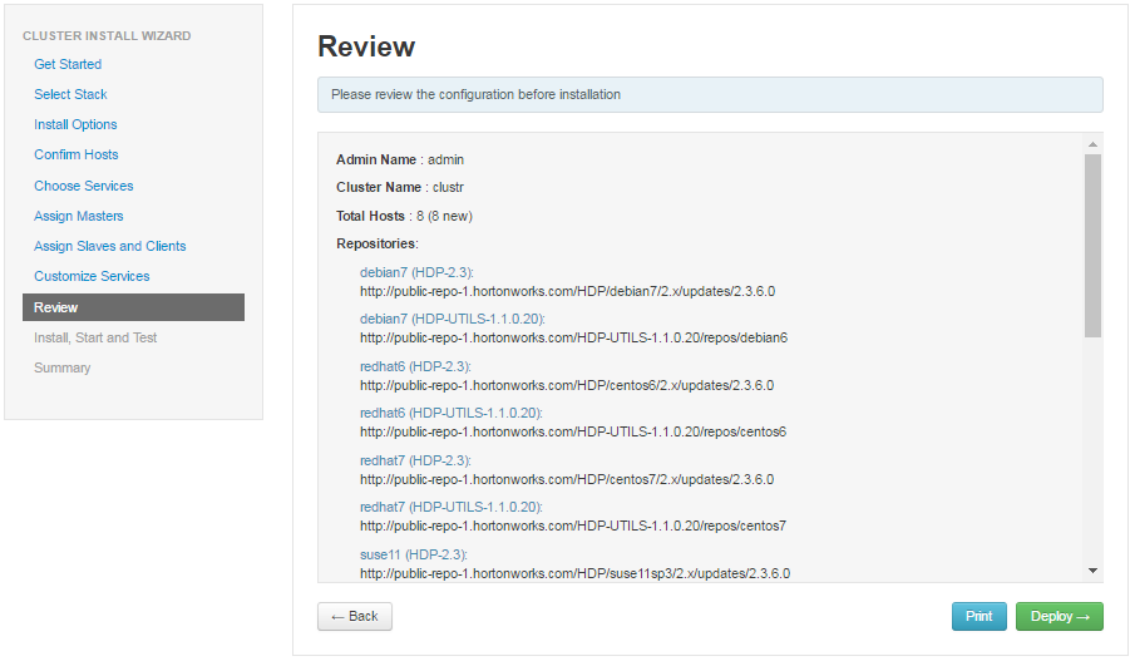

Illustration 19: Installation Review

Upon reviewing the settings, we proceed to start installing the cluster.

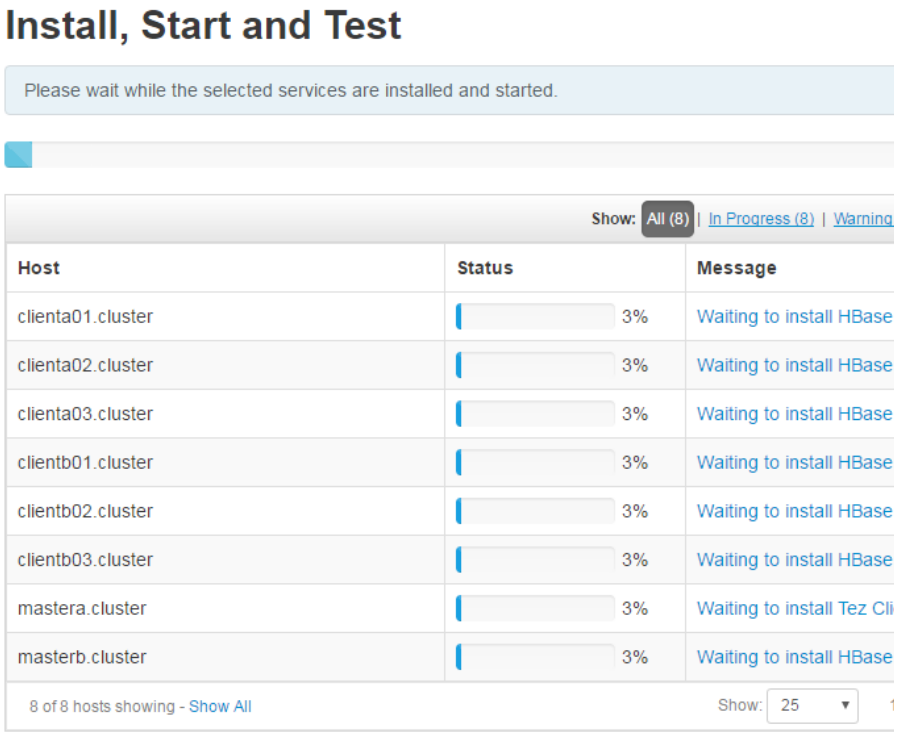

Illustration 20: Ambari Wizard, Installation

This will take some time based on the internet connection, hardware etc.
Having ended the installation and starting of the services (this may fail due to timeouts. But don't worry. Just restart a couple of times until all services startup).

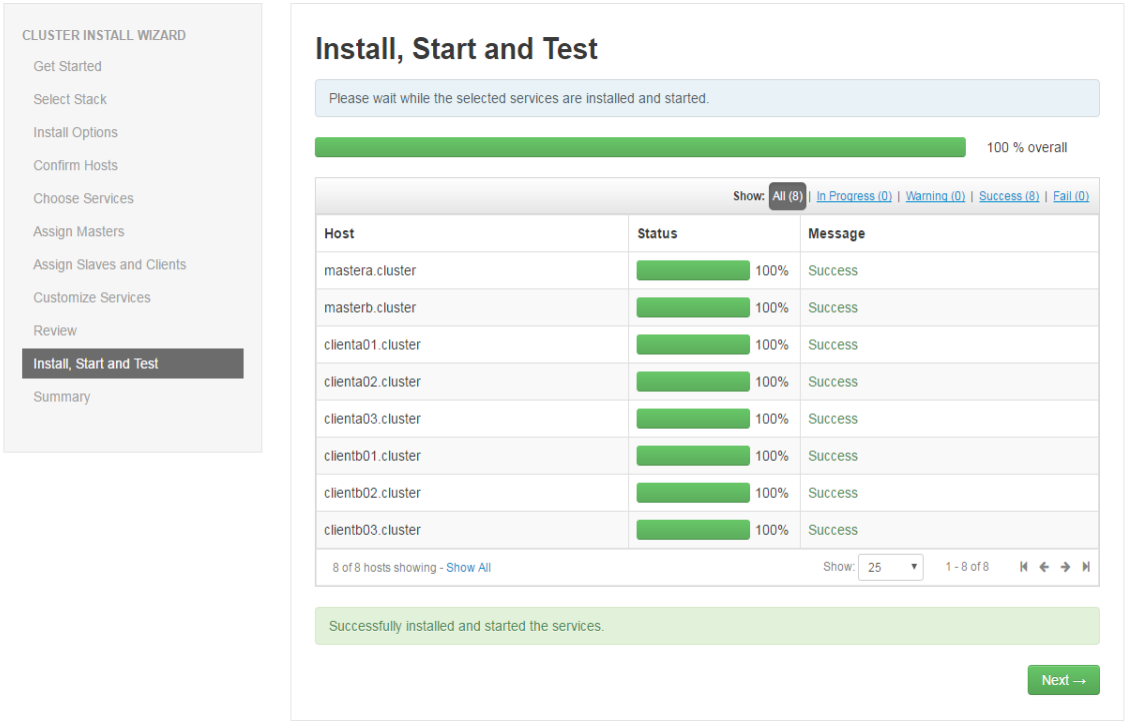

Illustration 21: End of Service installation

After the installation process has ended, go on to get the summary of the installation :

Reading the summary, we are informed, that everything worked out fine.

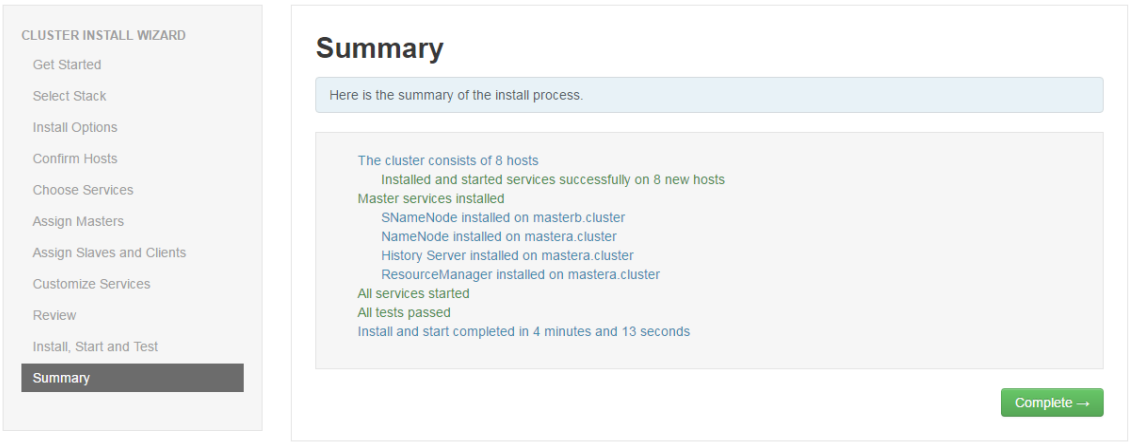

Illustration 22: Installation Summary

Finally, we get an updated frontpage showing us the status of the cluster :

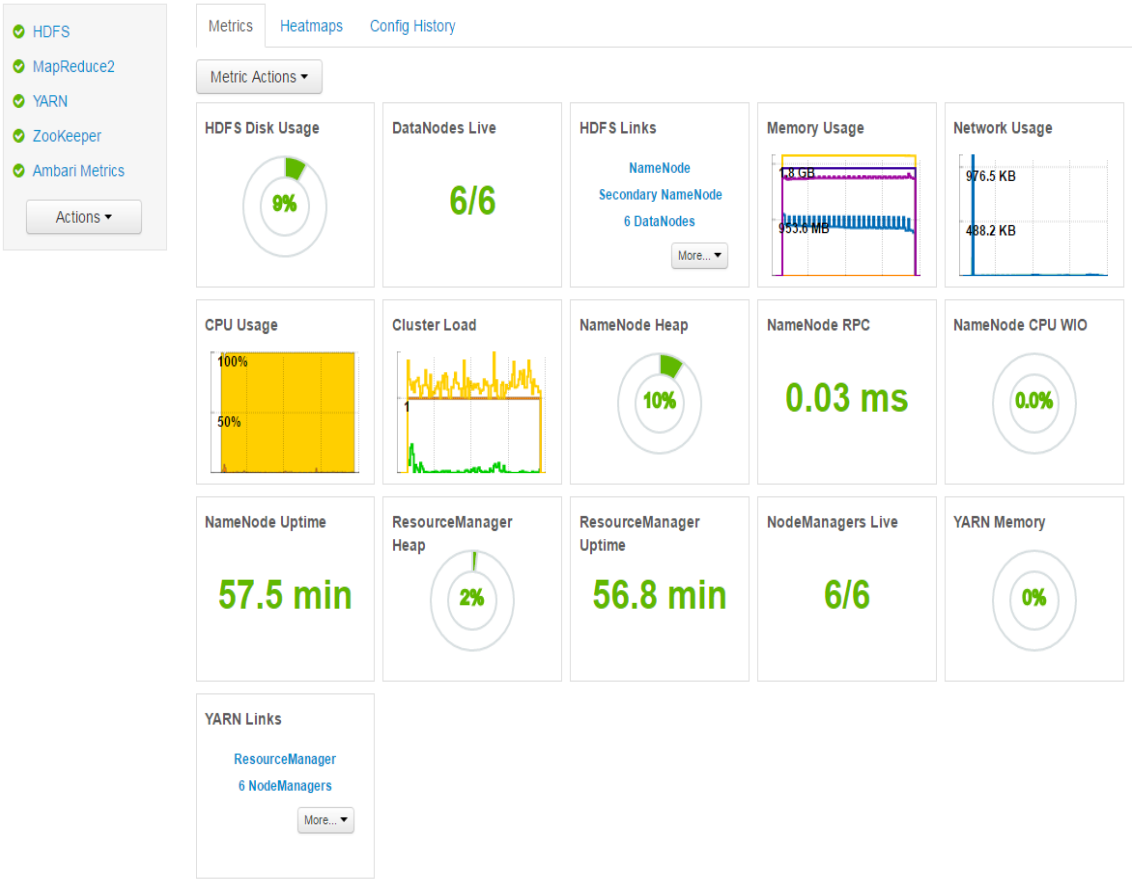

Illustration 23: Cluster control Panel

Now, we should start adding more services to the cluster.

Adding Spark

When installing the cluster, we did not install Spark right away. This would have been the easiest. However, given the internet connection it would result in a time out due to the long time downloading packages[3].

On the frontpage, select Add Service in the Actions menu

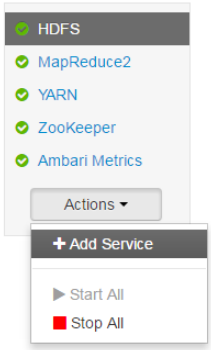

Illustration 24: Add Service

Scroll down and select Spark :

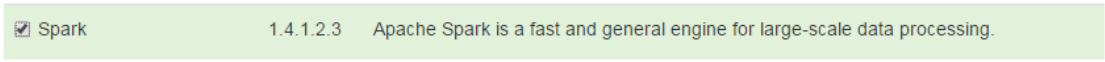

Illustration 25: Select Spark

We assign the mastera to run the Spark History server :

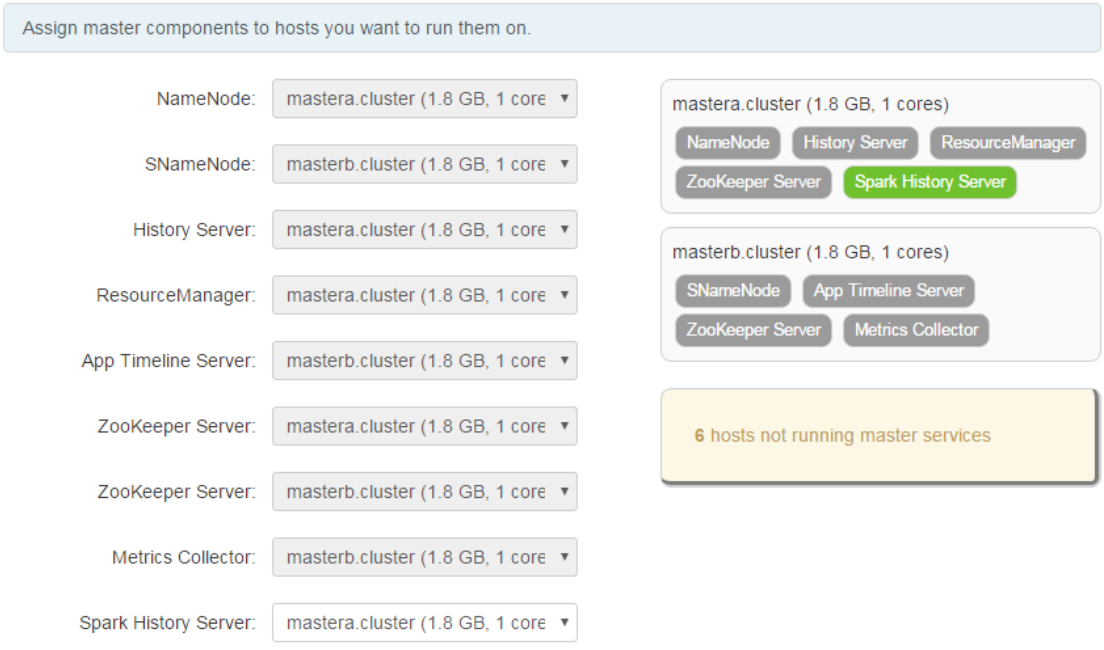

Illustration 26: Select Spark History server

3 There are ways around this. E.g. creating a local repository of packages. This, however, is beyond the scope of this document.

Select all (except masters) as clients :

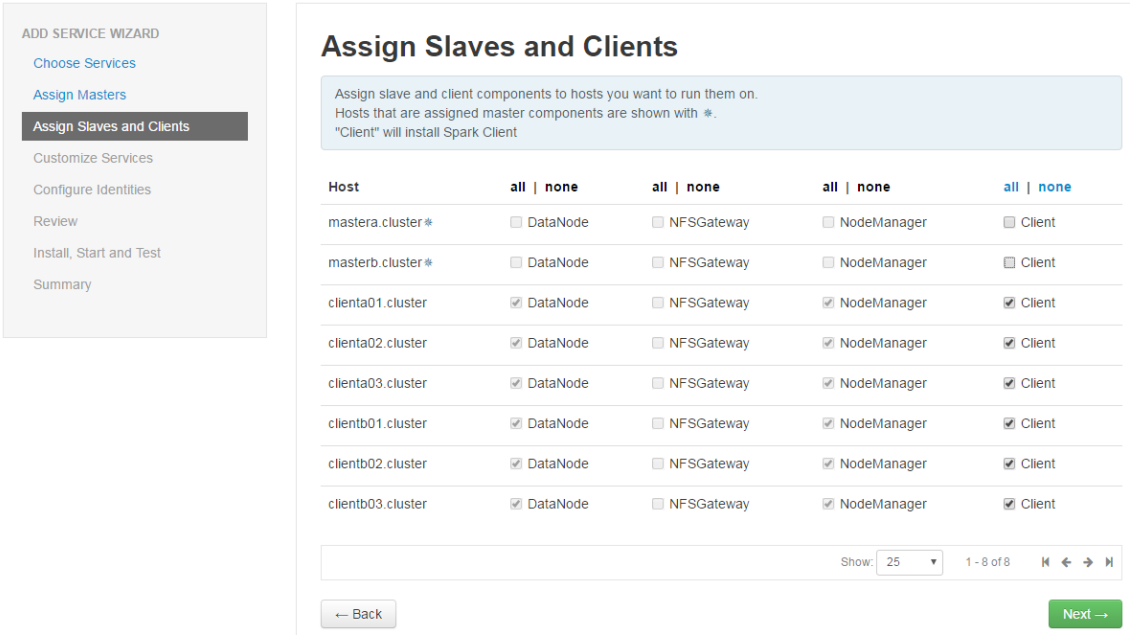

Illustration 27: Assign Spark Clients

I'll go for the default configuration :

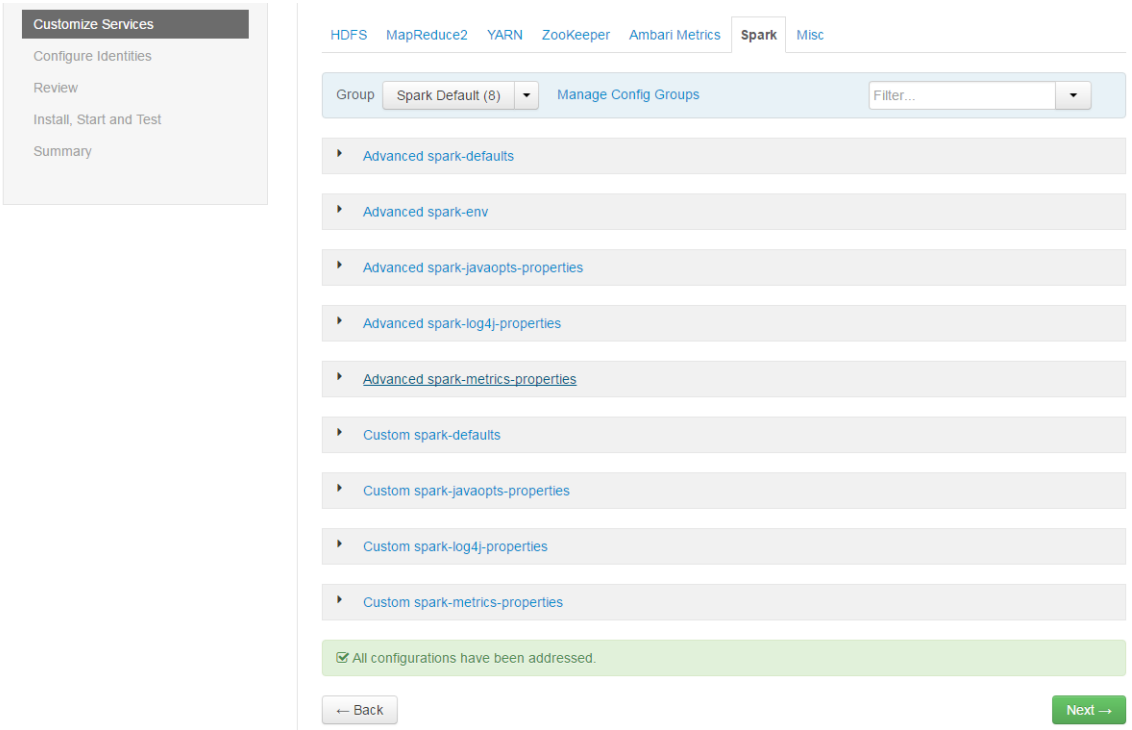

Illustration 28: Customize Spark Configuration

Review the configuration and start the deployment process :

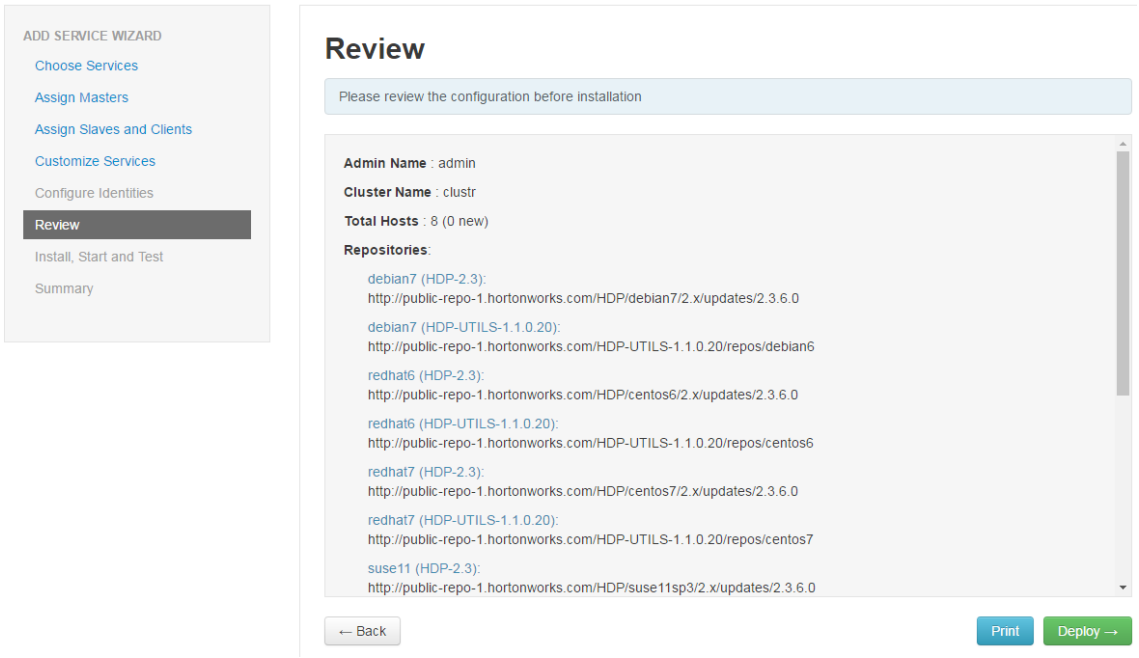

Illustration 29: Review Spark Configuration

Wait for the deployment to end :

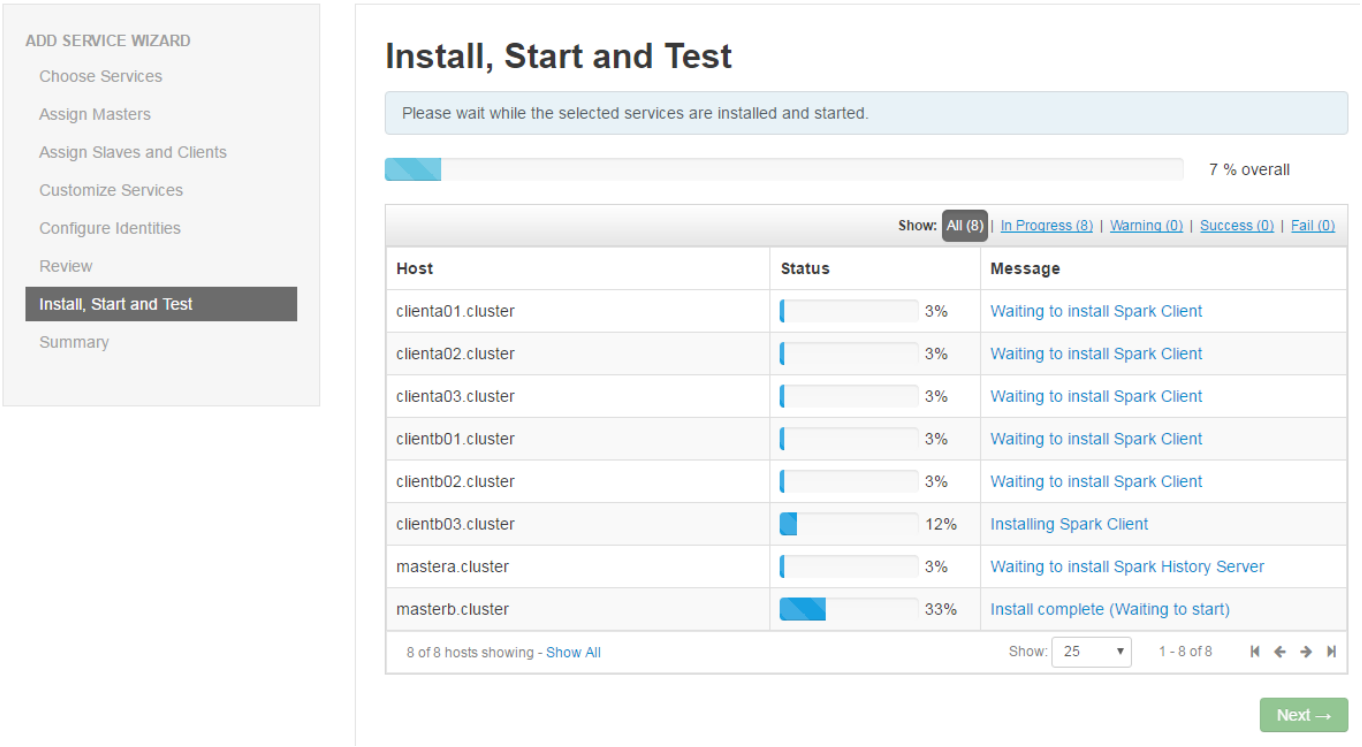

Illustration 30: Spark Installation and Initialization

This will take a while.

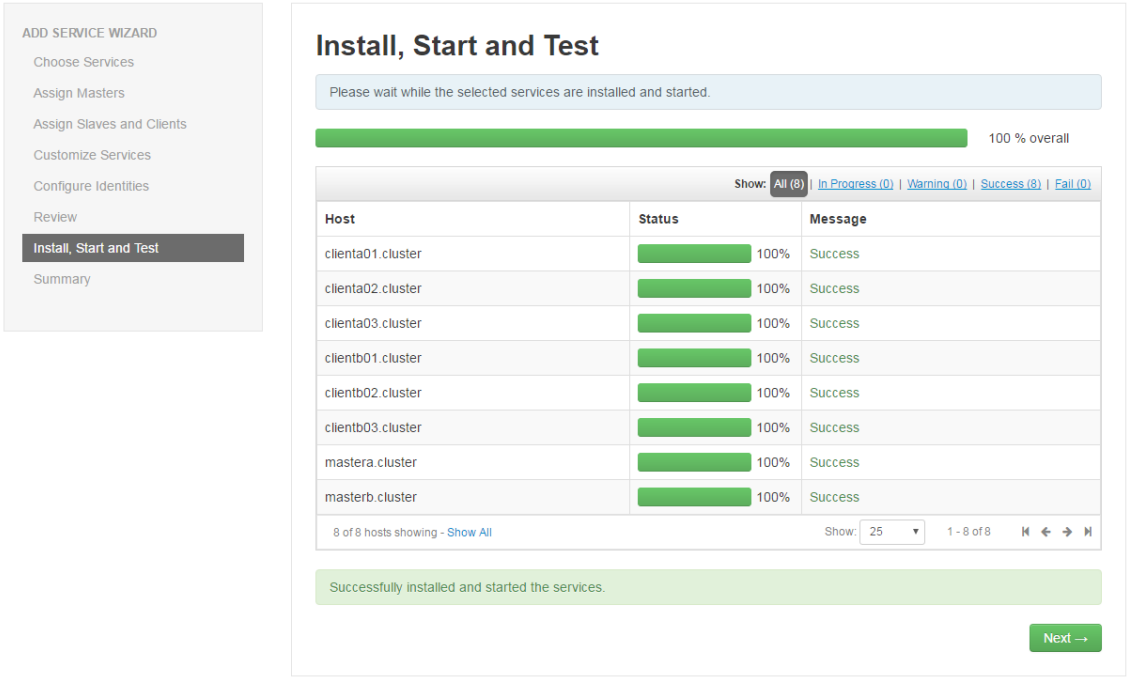

Illustration 31: End of installation

Having completed the installation, the Summary informs us, that certain services need to be restarted

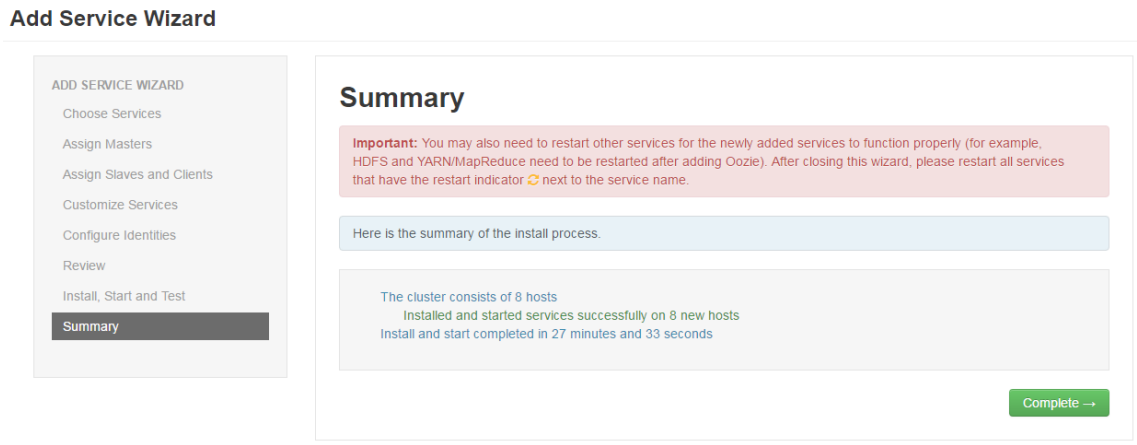

Illustration 32: Spark Installation Summary

Having installed the Spark service, we are once more presented with the frontpage. The only change is now another service

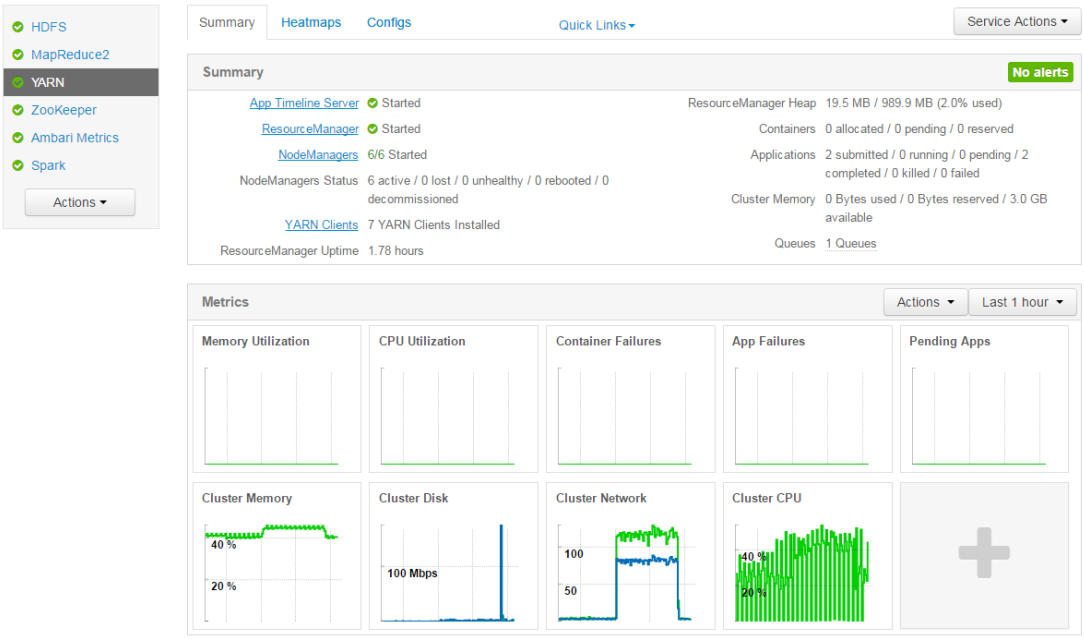

Illustration 33: Spark installation Frontpage

Test

In order to test our Spark installation, we use the demo ”Spark Pi” application.

Login into mastera and run the following code :

```
cd /usr/hdp/current/spark-client

su spark

./bin/spark-submit --class org.apache.spark.examples.SparkPi --master yarn-client --num-executors 1 --driver-memory 512m --executor-memory 512m --executor-cores 1 lib/spark-examples*.jar 10
```

This will soon fill the whole screen with a lot of log messages.

```
16/11/07 00:30:52 INFO YarnClientSchedulerBackend: Application application_1478474972019_0001 has started running.
16/11/07 00:30:52 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 38176.
16/11/07 00:30:52 INFO NettyBlockTransferService: Server created on 38176
16/11/07 00:30:52 INFO BlockManagerMaster: Trying to register BlockManager
16/11/07 00:30:52 INFO BlockManagerMasterEndpoint: Registering block manager 192.168.1.120:38176 with 267.3 MB RAM, BlockManagerId(driver, 192.168.1.120, 38176)
16/11/07 00:30:52 INFO BlockManagerMaster: Registered BlockManager
16/11/07 00:30:52 INFO YarnClientSchedulerBackend: SchedulerBackend is ready for scheduling beginning after waiting maxRegisteredResourcesWaitingTime: 30000(ms)
16/11/07 00:30:52 INFO YarnHistoryService: Application started: SparkListenerApplicationStart(Spark Pi,Some(application_1478474972019_0001),1478474998567,spark,None,None)
16/11/07 00:30:52 INFO YarnHistoryService: About to POST entity application_1478474972019_0001 with 3 events to timeline service http://masterb.cluster:8188/ws/v1/timeline/
16/11/07 00:30:52 INFO SparkContext: Starting job: reduce at SparkPi.scala:36
16/11/07 00:30:52 INFO DAGScheduler: Got job 0 (reduce at SparkPi.scala:36) with 10 output partitions
16/11/07 00:30:52 INFO DAGScheduler: Final stage: ResultStage 0(reduce at SparkPi.scala:36)
16/11/07 00:30:52 INFO DAGScheduler: Parents of final stage: List()
16/11/07 00:30:52 INFO DAGScheduler: Missing parents: List()
16/11/07 00:30:52 INFO DAGScheduler: Submitting ResultStage 0 (MapPartitionsRDD[1] at map at SparkPi.scala:32), which has no missing parents
16/11/07 00:30:53 INFO MemoryStore: ensureFreeSpace(1888) called with curMem=0, maxMem=280248975
16/11/07 00:30:53 INFO MemoryStore: Block broadcast_0 stored as values in memory (estimated size 1888.0 B, free 267.3 MB)
16/11/07 00:30:53 INFO MemoryStore: ensureFreeSpace(1202) called with curMem=1888, maxMem=280248975
16/11/07 00:30:53 INFO MemoryStore: Block broadcast_0_piece0 stored as bytes in memory (estimated size 1202.0 B, free 267.3 MB)
16/11/07 00:30:53 INFO BlockManagerInfo: Added broadcast_0_piece0 in memory on 192.168.1.120:38176 (size: 1202.0 B, free: 267.3 MB)
16/11/07 00:30:53 INFO SparkContext: Created broadcast 0 from broadcast at DAGScheduler.scala:861
16/11/07 00:30:53 INFO DAGScheduler: Submitting 10 missing tasks from ResultStage 0 (MapPartitionsRDD[1] at map at SparkPi.scala:32)
16/11/07 00:30:53 INFO YarnScheduler: Adding task set 0.0 with 10 tasks
16/11/07 00:30:57 INFO YarnClientSchedulerBackend: Registered executor: AkkaRpcEndpointRef(Actor[akka.tcp://sparkExecutor@clientb02.cluster:42895/user/Executor#1883927924]) with ID 1
16/11/07 00:30:57 INFO TaskSetManager: Starting task 0.0 in stage 0.0 (TID 0, clientb02.cluster, PROCESS_LOCAL, 2189 bytes)
16/11/07 00:30:58 INFO BlockManagerMasterEndpoint: Registering block manager clientb02.cluster:37918 with 267.3 MB RAM, BlockManagerId(1, clientb02.cluster, 37918)
16/11/07 00:31:07 INFO BlockManagerInfo: Added broadcast_0_piece0 in memory on clientb02.cluster:37918 (size: 1202.0 B, free: 267.3 MB)
16/11/07 00:31:08 INFO TaskSetManager: Starting task 1.0 in stage 0.0 (TID 1, clientb02.cluster, PROCESS_LOCAL, 2189 bytes)
16/11/07 00:31:08 INFO TaskSetManager: Starting task 2.0 in stage 0.0 (TID 2, clientb02.cluster, PROCESS_LOCAL, 2189 bytes)
16/11/07 00:31:08 INFO TaskSetManager: Starting task 3.0 in stage 0.0 (TID 3, clientb02.cluster, PROCESS_LOCAL, 2189 bytes)
16/11/07 00:31:08 INFO TaskSetManager: Starting task 4.0 in stage 0.0 (TID 4, clientb02.cluster, PROCESS_LOCAL, 2189 bytes)
16/11/07 00:31:08 INFO TaskSetManager: Starting task 5.0 in stage 0.0 (TID 5, clientb02.cluster, PROCESS_LOCAL, 2189 bytes)
16/11/07 00:31:08 INFO TaskSetManager: Starting task 6.0 in stage 0.0 (TID 6, clientb02.cluster, PROCESS_LOCAL, 2189 bytes)
16/11/07 00:31:08 INFO TaskSetManager: Starting task 7.0 in stage 0.0 (TID 7, clientb02.cluster, PROCESS_LOCAL, 2189 bytes)
16/11/07 00:31:08 INFO TaskSetManager: Starting task 8.0 in stage 0.0 (TID 8, clientb02.cluster, PROCESS_LOCAL, 2189 bytes)
16/11/07 00:31:08 INFO TaskSetManager: Starting task 9.0 in stage 0.0 (TID 9, clientb02.cluster, PROCESS_LOCAL, 2189 bytes)
16/11/07 00:31:08 INFO YarnHistoryService: About to POST entity application_1478474972019_0001 with 10 events to timeline service http://masterb.cluster:8188/ws/v1/timeline/
16/11/07 00:31:08 INFO TaskSetManager: Finished task 2.0 in stage 0.0 (TID 2) in 206 ms on clientb02.cluster (1/10)
16/11/07 00:31:08 INFO TaskSetManager: Finished task 4.0 in stage 0.0 (TID 4) in 178 ms on clientb02.cluster (2/10)
16/11/07 00:31:08 INFO TaskSetManager: Finished task 5.0 in stage 0.0 (TID 5) in 158 ms on clientb02.cluster (3/10)
16/11/07 00:31:08 INFO TaskSetManager: Finished task 6.0 in stage 0.0 (TID 6) in 141 ms on clientb02.cluster (4/10)
16/11/07 00:31:08 INFO TaskSetManager: Finished task 7.0 in stage 0.0 (TID 7) in 125 ms on clientb02.cluster (5/10)
16/11/07 00:31:08 INFO TaskSetManager: Finished task 8.0 in stage 0.0 (TID 8) in 102 ms on clientb02.cluster (6/10)
16/11/07 00:31:08 INFO TaskSetManager: Finished task 9.0 in stage 0.0 (TID 9) in 84 ms on clientb02.cluster (7/10)
16/11/07 00:31:08 INFO TaskSetManager: Finished task 1.0 in stage 0.0 (TID 1) in 249 ms on clientb02.cluster (8/10)
16/11/07 00:31:08 INFO TaskSetManager: Finished task 3.0 in stage 0.0 (TID 3) in 199 ms on clientb02.cluster (9/10)
16/11/07 00:31:08 INFO TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 10777 ms on clientb02.cluster (10/10)
16/11/07 00:31:08 INFO DAGScheduler: ResultStage 0 (reduce at SparkPi.scala:36) finished in 15.531 s
16/11/07 00:31:08 INFO YarnScheduler: Removed TaskSet 0.0, whose tasks have all completed, from pool
16/11/07 00:31:08 INFO DAGScheduler: Job 0 finished: reduce at SparkPi.scala:36, took 16.003066 s
```

Illustration 34: Raw output

Going to the Quick Links menu and ResourceManager UI, we can navigate to the overview of running Jobs.

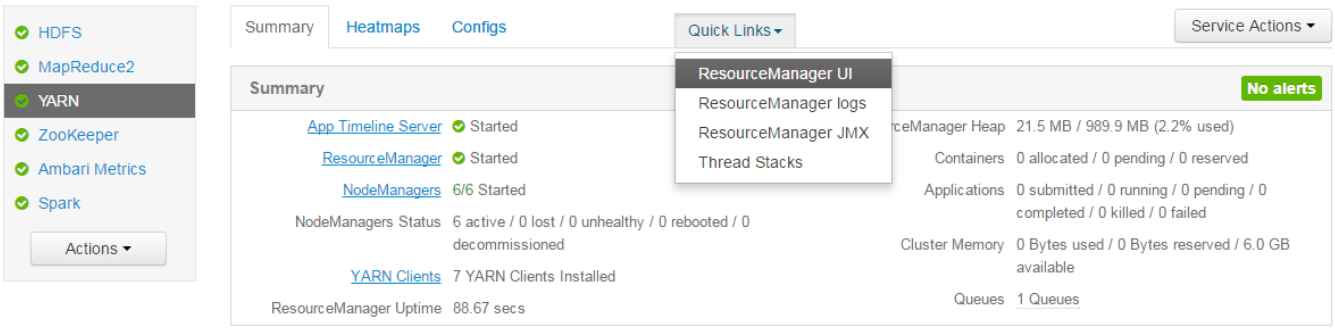

Illustration 35: Find Navigation Links

The application overview shows a list of applications

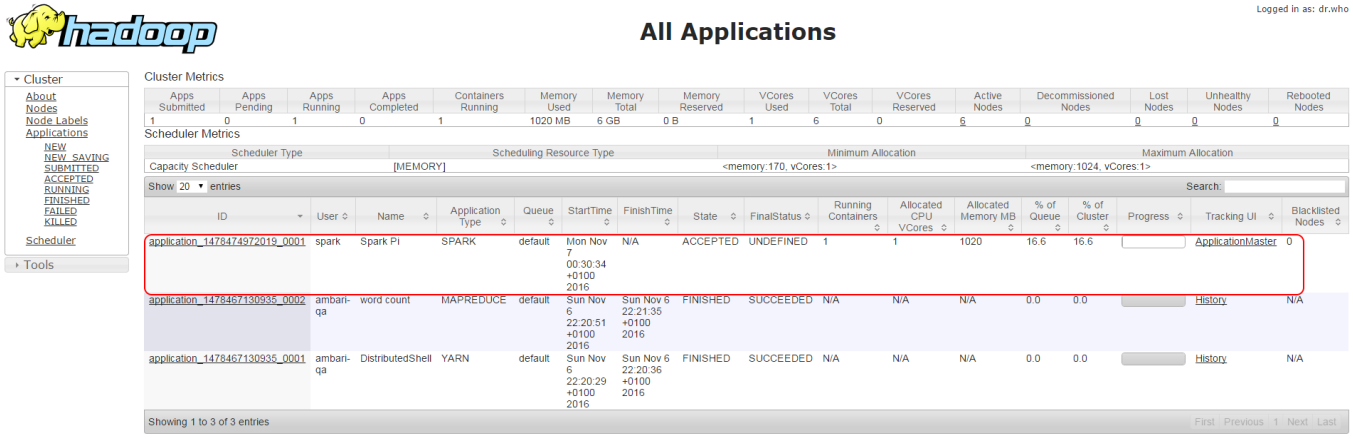

Illustration 36: Spark application overview

When the application has run to the end, the result is presented on the screen:

```
16/11/07 00:31:08 INFO DAGScheduler: ResultStage 0 (reduce at SparkPi.scala:36) finished in 15.531 s
16/11/07 00:31:08 INFO YarnScheduler: Removed TaskSet 0.0, whose tasks have all completed, from pool
16/11/07 00:31:08 INFO DAGScheduler: Job 0 finished: reduce at SparkPi.scala:36, took 16.003066 s
Pi is roughly 3.141908
```

The example is not especially usable. However, this is just a test of the installation.

Where to go from here?

Having a Hadoop cluster without using it is no fun. So, what should we use it for?

A Hadoop cluster is normally used for processing data from various sources, compressing and processing it into a simple result. The whole process is shown in the following illustration :

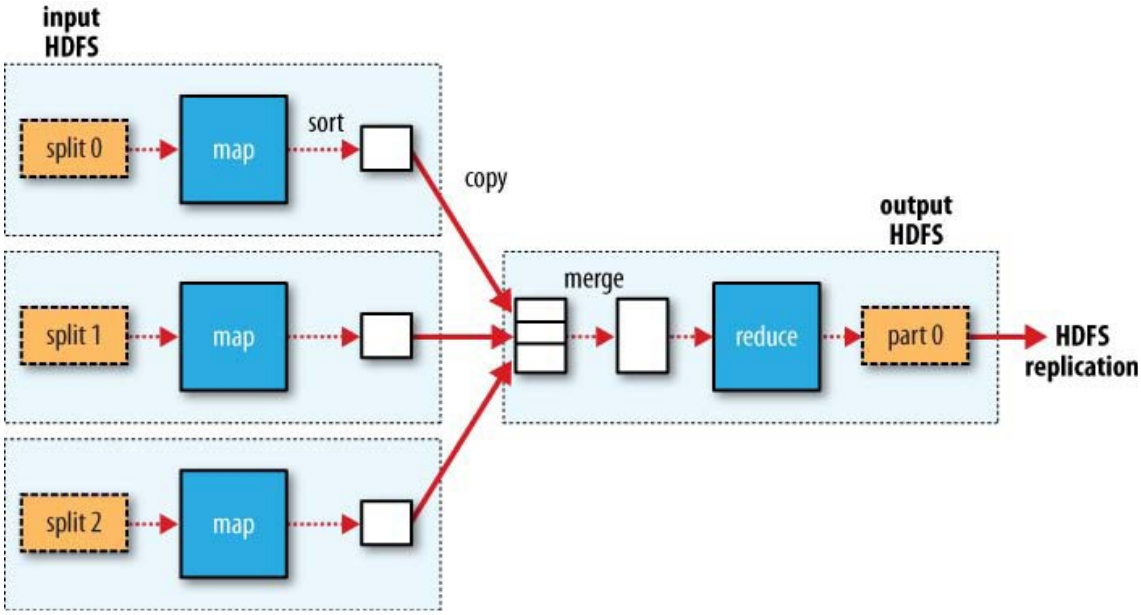

Illustration 37: Hadoop data flow

Having run the various samples on the cluster, verifying the state of the of the cluster, the example really not serve any practical uses.

That being said, the cluster now is ready to start working. In order to get acquainted wich Hadoop and Spark, I suggest you download a large dataset. A listing of these can be found at :

http://hadoopilluminated.com/hadoop_illuminated/Public_Bigdata_Sets.html

The data can be fed into the cluster and processed using various tools such as Spark, R etc.

Also, experiments with other types of services can be performed using the cluster.

So, happy Hadooping, and please feel free to keep an eye out for more intros on Hadoop.

/Henrik