HIVE (HADOOP) - Jenkins CI/CD Integration

What is Hadoop ?

Apache Hadoop is an open source software framework used to develop data processing applications which are executed in a distributed computing environment.These are mainly useful for achieving greater computational power at low cost.It provides massive storage for any kind of data, enormous processing power and the ability to handle virtually limitless concurrent tasks or jobs. Apache Hadoop consists of components including:

- Hadoop Distributed File System (HDFS), the bottom layer component for storage. HDFS breaks up files into chunks and distributes them across the nodes of the cluster.
- Yarn for job scheduling and cluster resource management.
- MapReduce for parallel processing.
- Common libraries needed by the other Hadoop subsystems.

Hadoop is often used in conjunction with Apache Spark and NoSQL databases to provide the data storage and management for Spark-powered data pipelines. A modern implementation of Hadoop now features an ecosystem of related projects that provide a rich set of big data services:

Apache Spark
Spark is a general purpose, distributed processing engine that performs high performance, in-memory processing of large data sets.

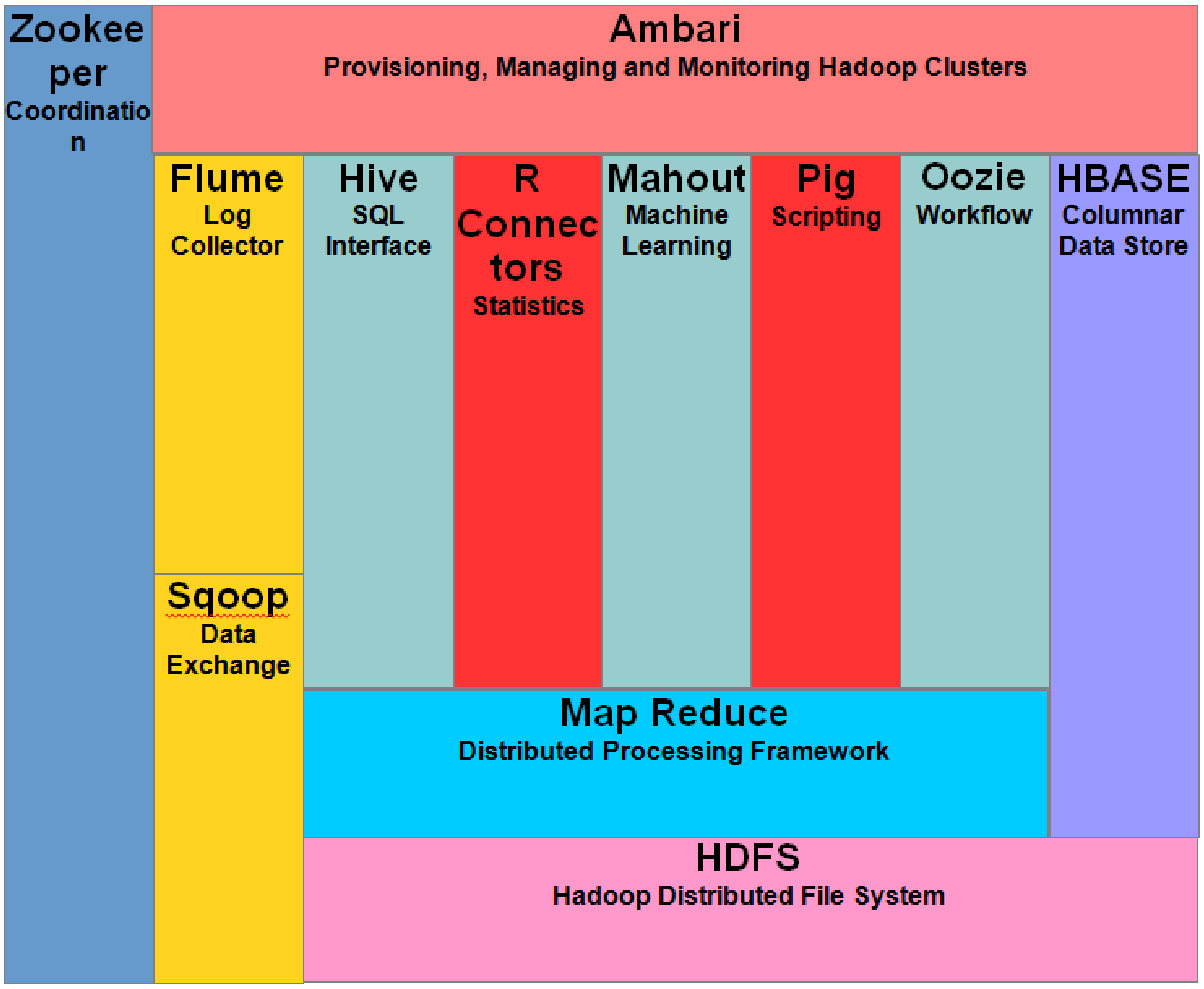
Apache Hive
Hive provides built-in data warehousing capabilities to the Hadoop system using a SQL-like access methods for querying data and analytics.

Apache HBase
HBase is a scalable, distributed NoSQL wide column database built on top of HDFS.

Apache Zeppelin
Zeppelin is a web-based, multi-purpose notebook that enables interactive data processing including ingestion, exploration, visualization, and collaboration features for Hadoop and Spark.

Hadoop EcoSystem and Components:

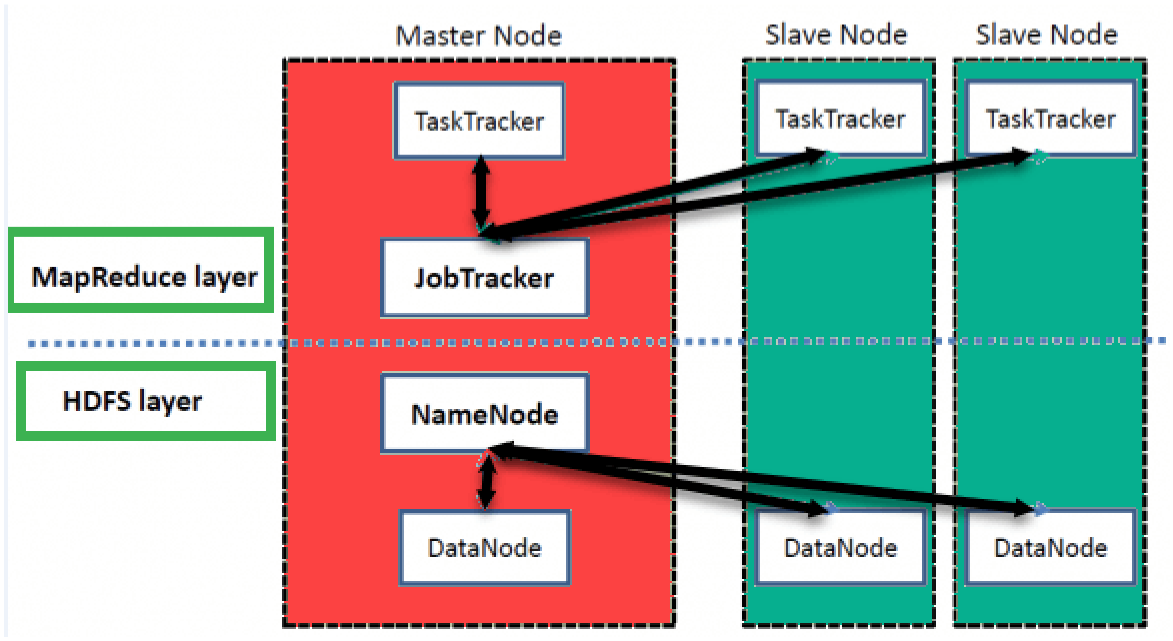
Below diagram shows various components in the Hadoop ecosystem-



Apache Hadoop consists of two sub-projects –

1. **Hadoop MapReduce:** MapReduce is a computational model and software framework for writing applications which are run on Hadoop. These MapReduce programs are capable of processing enormous data in parallel on large clusters of computation nodes.
2. **HDFS** (**Hadoop Distributed File System**): HDFS takes care of the storage part of Hadoop applications. MapReduce applications consume data from HDFS. HDFS creates multiple replicas of data blocks and distributes them on compute nodes in a cluster. This distribution enables reliable and extremely rapid computations.

Hadoop Architecture



Hadoop has a Master-Slave Architecture for data storage and distributed data processing using MapReduce and HDFS methods.

NameNode:

NameNode represented every files and directory which is used in the namespace

DataNode:

DataNode helps you to manage the state of an HDFS node and allows you to interacts with the blocks

MasterNode:

The master node allows you to conduct parallel processing of data using Hadoop MapReduce.

Slave node:

The slave nodes are the additional machines in the Hadoop cluster which allows you to store data to conduct complex calculations.

Hadoop Installation:

This set up is for creating Hadoop cluster with three Salve Nodes:

Some setup is common to all the nodes: NameNode and DataNodes. This is covered in this section.

ALL NODES: UPDATE THE INSTANCE

Let us update the OS with latest available software patches.

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

After the updates, the system might require a restart. Perform a Reboot from the EC2 Instances page.

All Nodes: Install Java

Let us now install Java. We install the package: openjdk-8-jdk-headless on all the nodes.

```
sudo apt-get -y install openjdk-8-jdk-headless
```

All Nodes: Install Apache Hadoop

Install Apache Hadoop 2.7.3 on all the instances. Obtain the link to download from the Apache website and run the following commands. We install Hadoop under a directory `server` in the home directory.

```
mkdir server
cd server
wget <Link to Hadoop 2.7.3>
tar xvzf hadoop-2.7.3.tar.gz
```

All Nodes: Setup JAVA_HOME

On each of the nodes, edit `~/server/hadoop-2.7.3/etc/hadoop/hadoop-env.sh`.

Replace this line:

```
export JAVA_HOME=${JAVA_HOME}
```

With the following line:

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

All Nodes: Update `core_site.xml`

On each node, edit `~/server/hadoop-2.7.3/etc/hadoop/core-site.xml` and replace the following lines:

```
<configuration>
</configuration>
```

with these (as mentioned above, replace <nnode> with NameNode’s public DNS):

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

All Nodes: Create Data Dir

HDFS needs the data directory to be present on each node: 1 name node and 3 data nodes. Create this directory as shown and change ownership to user ubuntu.

```
sudo mkdir -p /usr/local/hadoop/hdfs/data
sudo chown -R ubuntu:ubuntu /usr/local/hadoop/hdfs/data
```

Configuring NameNode

After performing configuration common to all nodes, let us now setup the NameNode.

Namenode: Password Less SSH

As mentioned before, we need password-less SSH between the name nodes and the data nodes. Let us create a public-private key pair for this purpose on the namenode.

```
namenode> ssh-keygen
```

Use the default (`/home/ubuntu/.ssh/id_rsa`) for the key location and hit enter for an empty passphrase.

Datanodes: Setup Public Key

The public key is saved in `/home/ubuntu/.ssh/id_rsa.pub`. We need to copy this file from the namenode to each data node and append the contents to `/home/ubuntu/.ssh/authorized_keys` on each data node.

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

Namenode: Setup SSH Config

SSH uses a configuration file located at `~/.ssh/config` for various parameters. Set it up as shown below. Again, substitute each node's Public DNS for the HostName parameter (for example, replace <nnode> with EC2 Public DNS for NameNode).

```
Host nnode
  HostName <nnode>
  User ubuntu
  IdentityFile ~/.ssh/id_rsa

Host dnode1
  HostName <dnode1>
  User ubuntu
  IdentityFile ~/.ssh/id_rsa

Host dnode2
  HostName <dnode2>
  User ubuntu
  IdentityFile ~/.ssh/id_rsa

Host dnode3
  HostName <dnode3>
  User ubuntu
  IdentityFile ~/.ssh/id_rsa
```

At this point, verify that password-less operation works on each node as follows (the first time, you will get a warning that the host is unknown and whether you want to connect to it. Type *yes* and hit enter. This step is needed once only):

```
namenode> ssh nnode
namenode> ssh dnode1
namenode> ssh dnode2
namenode> ssh dnode3
```

Namenode: Setup HDFS Properties

On the NameNode, edit the following file:`~/server/hadoop-2.7.3/etc/hadoop/hdfs-site.xml`

Replace:

```
<configuration>
</configuration>
```

With:

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>3</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:///usr/local/hadoop/hdfs/data</value>
  </property>
</configuration>
```

Namenode: Setup MapReduce Properties

On the NameNode, copy the file (`~/server/hadoop-2.7.3/etc/hadoop/mapred-site.xml.template`) to (`~/server/hadoop-2.7.3/etc/hadoop/mapred-site.xml`). Replace:

```
<configuration>
</configuration>
```

With this (as above replace <nnode> with NameNode's public DNS):

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

Namenode: Setup YARN Properties

Next we need to set up `~/server/hadoop-2.7.3/etc/hadoop/yarn-site.xml` on the NameNode. Replace the following:

```
<configuration>

<!-- Site specific YARN configuration properties -->

</configuration>
```

With (as before, replace <nnode> with NameNode's public DNS):

```
<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><nnode></value>
  </property>

</configuration>
```

Namenode: Setup Master and Slaves

On the NameNode, create `~/server/hadoop-2.7.3/etc/hadoop/masters` with the following (replace <nnode> with the NameNode's public DNS):

```
<nnode>
```

Also replace all content in `~/server/hadoop-2.7.3/etc/hadoop/slaves` with (replace each of <dnode1>, etc with the appropriate DateNode's public DNS):

```
<dnode1>
<dnode2>
<dnode3>
```

Configuring Data Nodes

After covering configuration common to both NameNode and DataNodes, we have a little bit of configuring specific to DataNodes. On each data node, edit the file `~/server/hadoop-2.7.3/etc/hadoop/hdfs-site.xml` and replace the following:

```
<configuration>
</configuration>
```

With:

```
<property>
    <name>dfs.replication</name>
    <value>3</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:///usr/local/hadoop/hdfs/data</value>
  </property>
```

Starting the Hadoop Cluster

After all that configuration, it is now time to test drive the cluster. First, format the HDFS file system on the NameNode:

```
namenode> cd ~/server
namenode> ./hadoop-2.7.3/bin/hdfs namenode -format
```

Finally, startup the Hadoop Cluster. After this step you should have all the daemons running on the NameNode and the DataNodes.

```
namenode> ./hadoop-2.7.3/sbin/start-dfs.sh
namenode> ./hadoop-2.7.3/sbin/start-yarn.sh
namenode> ./hadoop-2.7.3/sbin/mr-jobhistory-daemon.sh start
historyserver
```

Check the console carefully for any error messages. If everything looks OK, check for the daemons using `jps`.

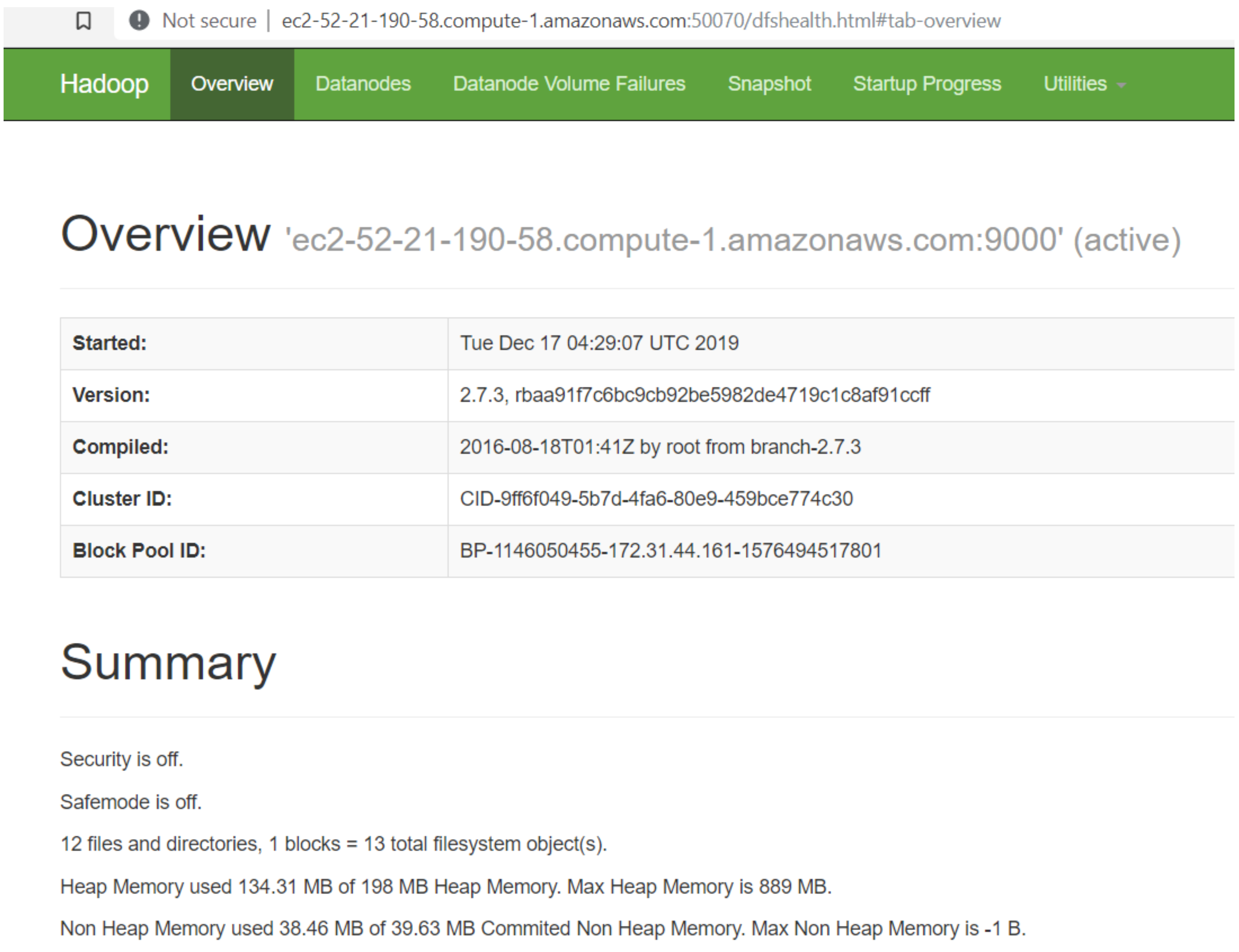
```
namenode> jps
```

You can also check on the datanodes for java processes.

```
datanode1> jps
```

Check the Web UI

Once the cluster is running, we can check the web UI for the status of the data nodes. Go to **http://<nnode>:50070** for the Web UI and verify that the 3 data nodes added are online.


Not secure | ec2-52-21-190-58.compute-1.amazonaws.com:50070/dfshealth.html#tab-overview
Hadoop
Overview
Datanodes
Datanode Volume Failures
Snapshot
Startup Progress
Utilities

Overview 'ec2-52-21-190-58.compute-1.amazonaws.com:9000' (active)

Started:	Tue Dec 17 04:29:07 UTC 2019
Version:	2.7.3, rbaa91f7c6bc9cb92be5982de4719c1c8af91ccff
Compiled:	2016-08-18T01:41Z by root from branch-2.7.3
Cluster ID:	CID-9ff6f049-5b7d-4fa6-80e9-459bce774c30
Block Pool ID:	BP-1146050455-172.31.44.161-1576494517801

Summary

Security is off.

Safemode is off.

12 files and directories, 1 blocks = 13 total filesystem object(s).

Heap Memory used 134.31 MB of 198 MB Heap Memory. Max Heap Memory is 889 MB.

Non Heap Memory used 38.46 MB of 39.63 MB Commited Non Heap Memory. Max Non Heap Memory is -1 B.

What is HIVE ?

Apache Hive is an open source data warehouse system built on top of **Hadoop** Haused for querying and analyzing large datasets stored in Hadoop files. It process structured and semi-structured data in Hadoop.

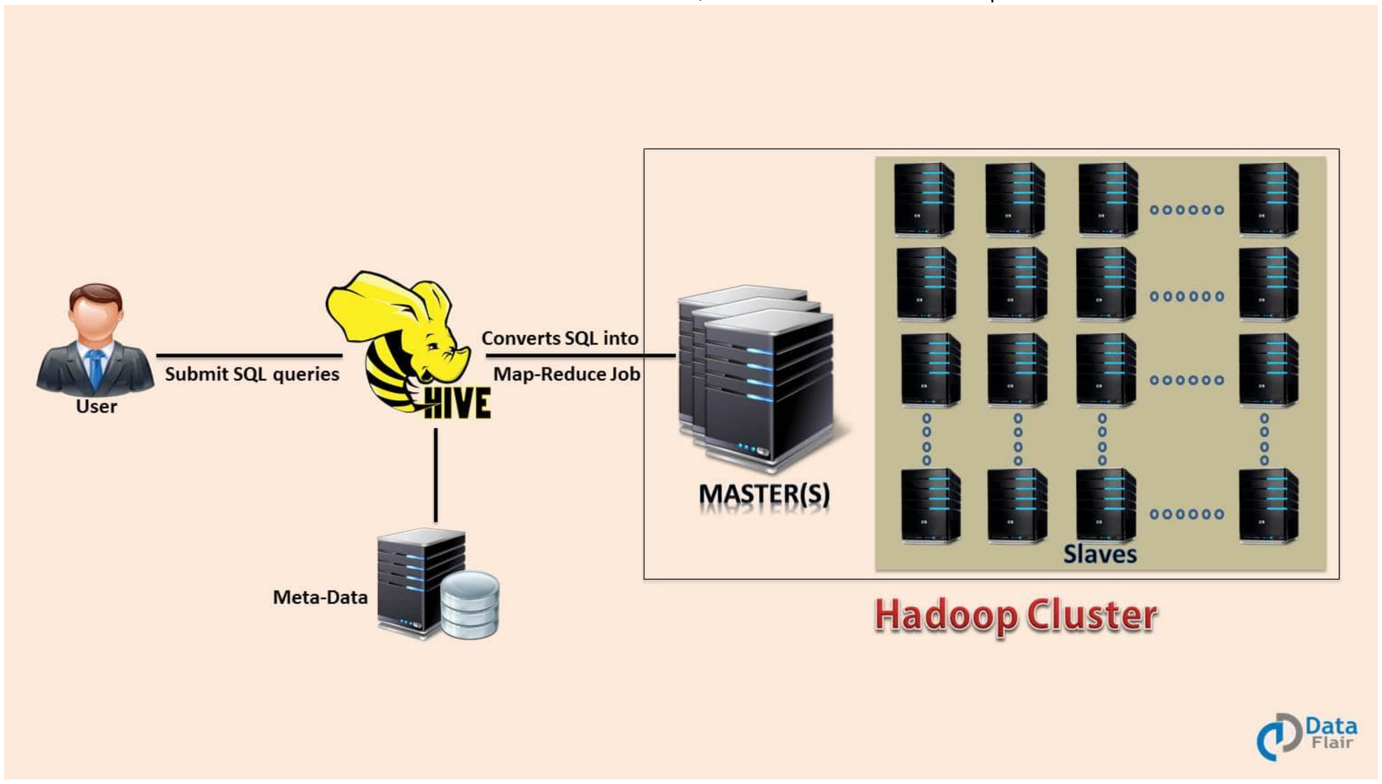
It converts SQL-like queries into MapReduce jobs for easy execution and processing of extremely large volumes of data.

Hive Architecture

The Apache Hive components are-

- **Metastore –** It stores metadata for each of the tables like their schema and location. Hive also includes the partition metadata. This helps the driver to track the progress of various data sets distributed over the cluster. It stores the data in a traditional RDBMS format. Hive metadata helps the driver to keep a track of the data and it is highly crucial. Backup server regularly replicates the data which it can retrieve in case of data loss.
- **Driver –** It acts like a controller which receives the HiveQL statements. The driver starts the execution of the statement by creating sessions. It monitors the life cycle and progress of the execution. Driver stores the necessary metadata generated during the execution of a HiveQL statement. It also acts as a collection point of data or query result obtained after the Reduce operation.
- **Compiler –** It performs the compilation of the HiveQL query. This converts the query to an execution plan. The plan contains the tasks. It also contains steps needed to be performed by the MapReduce to get the output as translated by the query. The compiler in Hive converts the query to an **Abstract Syntax Tree (AST)**. First, check for compatibility and compile-time errors, then converts the AST to a **Directed Acyclic Graph (DAG).**
- **Optimizer –** It performs various transformations on the execution plan to provide optimized DAG. It aggregates the transformations together, such as converting a pipeline of joins to a single join, for better performance. The optimizer can also split the tasks, such as applying a transformation on data before a reduce operation, to provide better performance.
- **Executor –** Once compilation and optimization complete, the executor executes the tasks. Executor takes care of pipelining the tasks.
- **CLI, UI, and Thrift Server –** CLI (command-line interface) provides a user interface for an external user to interact with Hive. Thrift server

in Hive allows external clients to interact with Hive over a network, similar to the JDBC or ODBC protocols.



Installing Hive

The following steps are required for installing Hive on your system.

$ wget http://archive.apache.org/dist/hive/hive-2.1.0/apache-hive-2.1.0-bin.tar.gz

$ `tar zxvf apache-hive-0.14.0-bin.tar.gz`

Setting up the Environment variable in ~/.bashrc.

```
export HIVE_HOME=/home/ubuntu/server/apache-hive-2.1.0-bin
export PATH=$PATH:$HIVE_HOME/bin
export CLASSPATH=$CLASSPATH:/home/ubuntu/server/hadoop-2.7.3/lib/:.
export PATH=$PATH:/home/ubuntu/server/apache-hive-3.1.2-bin/lib:.
export HADOOP_HOME=/home/ubuntu/server/hadoop-2.7.3
```

The following command is used to execute ~/.bashrc file.

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

Configuring Hive

To configure Hive with Hadoop, you need to edit the **hive-env.sh** file, which is placed in the **$HIVE_HOME/conf** directory. The following commands redirect to Hive **config** folder and copy the template file:

```
$ cd $HIVE_HOME/conf
$ cp hive-env.sh.template hive-env.sh
```

Edit the **hive-env.sh** file by appending the following line:

```
export HADOOP_HOME=/home/ubuntu/server/hadoop-2.7.3
```

Installing Apache Derby

Follow the steps given below to download and install Apache Derby:

Downloading Apache Derby

The following command is used to download Apache Derby. It takes some time to download.

```
$ cd ~
$ wget
http://archive.apache.org/dist/db/derby/db-derby-10.4.2.0/db-derby-10.4.
2.0-bin.tar.gz
```

Extracting and verifying Derby archive

The following commands are used for extracting and verifying the Derby archive:

```
$ tar zxvf db-derby-10.4.2.0-bin.tar.gz
```

Setting up environment for Derby

You can set up the Derby environment by appending the following lines to **~/.bashrc** file:

```
export DERBY_HOME=/home/ubuntu/server/db-derby-10.4.2.0-bin
export PATH=$PATH:$DERBY_HOME/bin
```

The following command is used to execute **~/.bashrc** file:

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

Create a directory to store Metastore

Create a directory named data in $DERBY_HOME directory to store Metastore data.

```
$ mkdir $DERBY_HOME/data
```

Derby installation and environmental setup is now complete.

Configuring Metastore of Hive

Configuring Metastore means specifying to Hive where the database is stored. You can do this by editing the hive-site.xml file, which is in the $HIVE_HOME/conf directory. First of all, copy the template file using the following command:

```
$ cd $HIVE_HOME/conf
$ cp hive-default.xml.template hive-site.xml
```

Edit **hive-site.xml** and append the following lines between the <configuration> and </configuration> tags:

```
<property>
   <name>javax.jdo.option.ConnectionURL</name>
   <value>jdbc:derby://localhost:1527/metastore_db;create=true </value>
   <description>JDBC connect string for a JDBC metastore </description>
</property>
```

Create a file named jpox.properties and add the following lines into it:

```
javax.jdo.PersistenceManagerFactoryClass =

org.jpox.PersistenceManagerFactoryImpl
org.jpox.autoCreateSchema = false
org.jpox.validateTables = false
org.jpox.validateColumns = false
org.jpox.validateConstraints = false
org.jpox.storeManagerType = rdbms
org.jpox.autoCreateSchema = true
org.jpox.autoStartMechanismMode = checked
org.jpox.transactionIsolation = read_committed
javax.jdo.option.DetachAllOnCommit = true
javax.jdo.option.NontransactionalRead = true
javax.jdo.option.ConnectionDriverName =
org.apache.derby.jdbc.ClientDriver
javax.jdo.option.ConnectionURL =
jdbc:derby://hadoop1:1527/metastore_db;create = true
javax.jdo.option.ConnectionUserName = APP
javax.jdo.option.ConnectionPassword = mine
```

Step 8: Verifying Hive Installation

Before running Hive, you need to create the **/tmp** folder and a separate Hive folder in HDFS. Here, we use the **/user/hive/warehouse** folder. You need to set write permission for these newly created folders as shown below:

```
chmod g+w
```

Now set them in HDFS before verifying Hive. Use the following commands:

```
$ $HADOOP_HOME/bin/hadoop fs -mkdir /tmp
$ $HADOOP_HOME/bin/hadoop fs -mkdir /user/hive/warehouse
$ $HADOOP_HOME/bin/hadoop fs -chmod g+w /tmp
$ $HADOOP_HOME/bin/hadoop fs -chmod g+w /user/hive/warehouse
```

The following commands are used to verify Hive installation:

```
$ cd $HIVE_HOME
$ bin/hive
```

On successful installation of Hive, you get to see the following response:

```
Logging initialized using configuration in
jar:file:/home/hadoop/hive-0.9.0/lib/hive-common-0.9.0.jar!/hive-log4j.p
roperties
Hive history
file=/tmp/hadoop/hive_job_log_hadoop_201312121621_1494929084.txt
…………………….
hive>
```

The following sample command is executed to display all the tables:

```
hive> show tables;
OK
Time taken: 2.798 seconds
hive>
```

HIVE Integration with Jenkins

To configure Jenkins instance with Hive you need to upload pem key in credentials section of Jenkins. Create new item or Pipeline which will consist the shell or groovy code to run the pipeline.

Pipeline/Shell code will consist all .hql files of Hive to fetch or update data in DB.

Sample code for Jenkins integration:

Jenkinsfile

```
node {
  def remote = [:]
  remote.name = 'ip-172-31-44-161'
  remote.host = '52.21.190.58'
  remote.user = 'ubuntu'
  remote.identityFile = '/var/jenkins_home/Key.pem'
  remote.allowAnyHosts = true

   stage('Remote SSH to Hadoop ') {
   sh "pwd"
   sh "ls -ltr"
    sshCommand remote: remote, command: "ls"
      }
  stage('Hive Queries') {
 sshScript remote: remote, script: "/var/jenkins_home/db.sh"
 }
 }
```

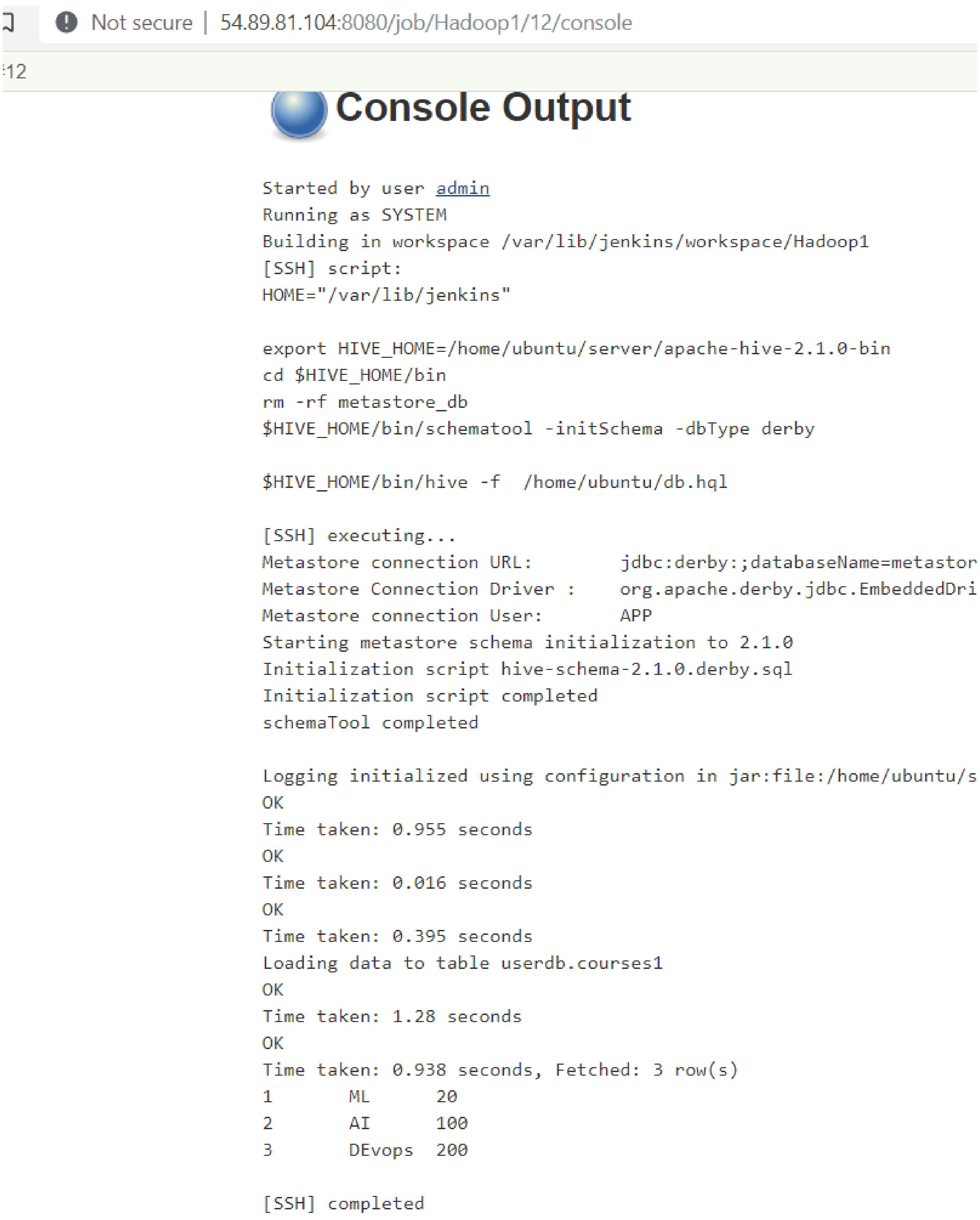
db.sh

```
export HIVE_HOME="/home/ubuntu/server/apache-hive-2.1.0-bin"
cd ${HIVE_HOME}/
rm -rf metastore_db
${HIVE_HOME}/bin/schematool -initSchema -dbType derby
${HIVE_HOME}/bin/hive  -f  /home/ubuntu/db.hql
```

db.hql

```
-- Create Database
!echo "Creating Database..."
CREATE DATABASE userdb;
!echo "Creating Table..."
USE userdb;
create table courses1(course_id int,course_name string,students_enrolled
int) ROW FORMAT DELIMITED FIELDS TERMINATED BY ' ';
LOAD DATA LOCAL INPATH '/home/ubuntu/test2.txt' OVERWRITE INTO TABLE
courses1;
!echo "Displaying content of Table from Hive;
select * from courses1;
```

Output:



Not secure | 54.89.81.104:8080/job/Hadoop1/12/console

:12

Console Output

```
Started by user admin
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/Hadoop1
[SSH] script:
HOME="/var/lib/jenkins"

export HIVE_HOME=/home/ubuntu/server/apache-hive-2.1.0-bin
cd $HIVE_HOME/bin
rm -rf metastore_db
$HIVE_HOME/bin/schematool -initSchema -dbType derby

$HIVE_HOME/bin/hive -f  /home/ubuntu/db.hql

[SSH] executing...
Metastore connection URL:        jdbc:derby:;databaseName=metastor
Metastore Connection Driver :    org.apache.derby.jdbc.EmbeddedDri
Metastore connection User:       APP
Starting metastore schema initialization to 2.1.0
Initialization script hive-schema-2.1.0.derby.sql
Initialization script completed
schemaTool completed

Logging initialized using configuration in jar:file:/home/ubuntu/s
OK
Time taken: 0.955 seconds
OK
Time taken: 0.016 seconds
OK
Time taken: 0.395 seconds
Loading data to table userdb.courses1
OK
Time taken: 1.28 seconds
OK
Time taken: 0.938 seconds, Fetched: 3 row(s)
1       ML      20
2       AI      100
3       DEvops  200

[SSH] completed
```