POC 1 :

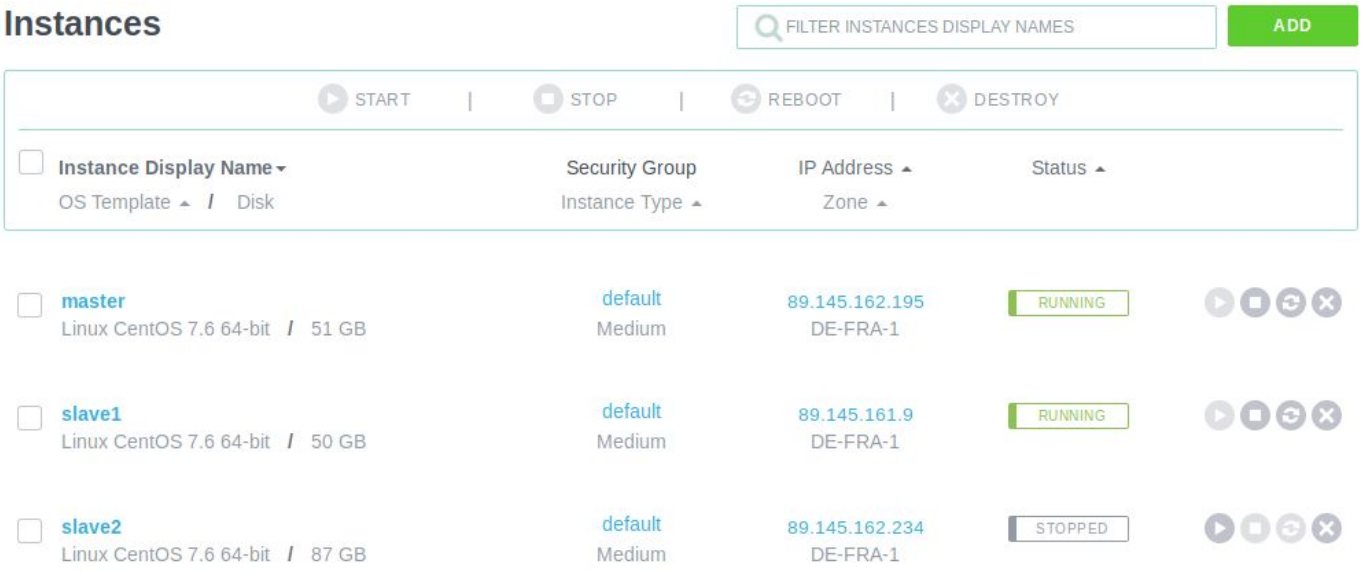
From data integration to Big data distributed processing and storage

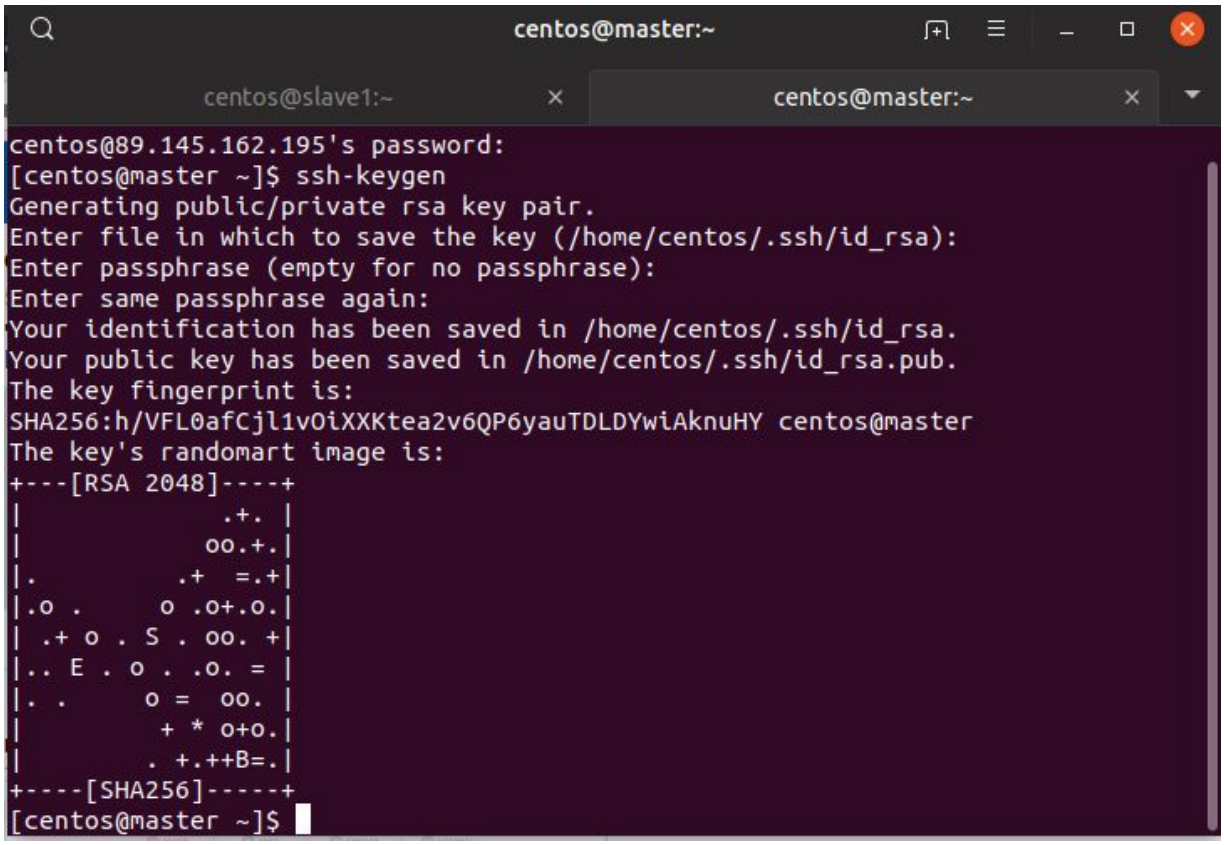
Rapport technique

Refka MEJRI
Wouroud GUEDDICH
Ibrahim ROUIS
Abdallah HAMZA

I- Setting up Ambari

1- Create instances



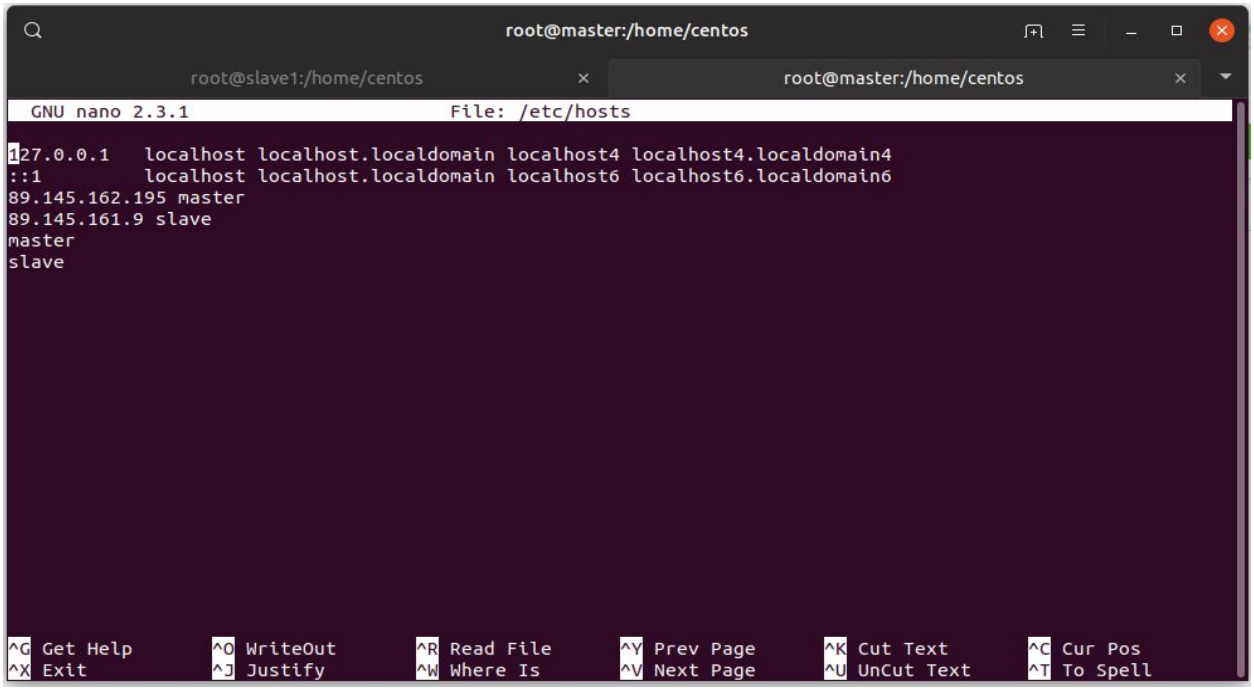


2- SSH key generation

$ ssh-keygen

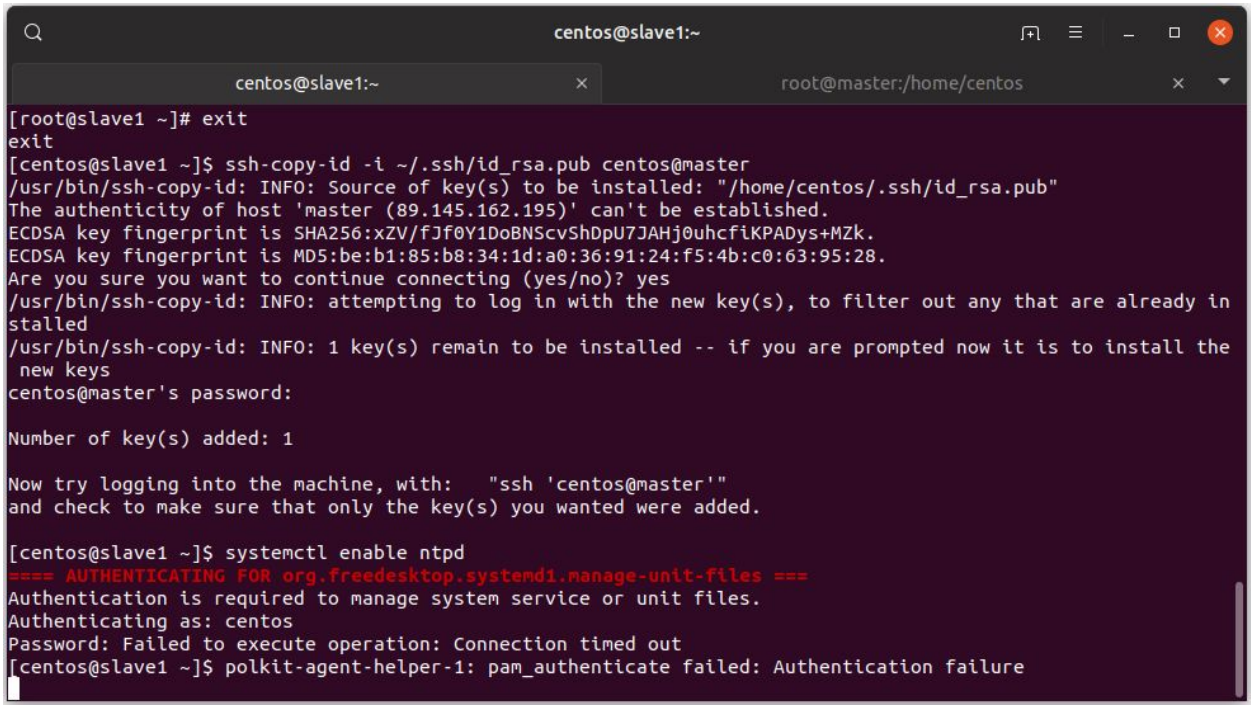
3- Edit Hosts file

$ nano /etc/hosts



4- Copy SSH key in different hosts

$ ssh-copy-id -i ~/.ssh/id_rsa.pub centos@master



5- Disable firewall and security configuration

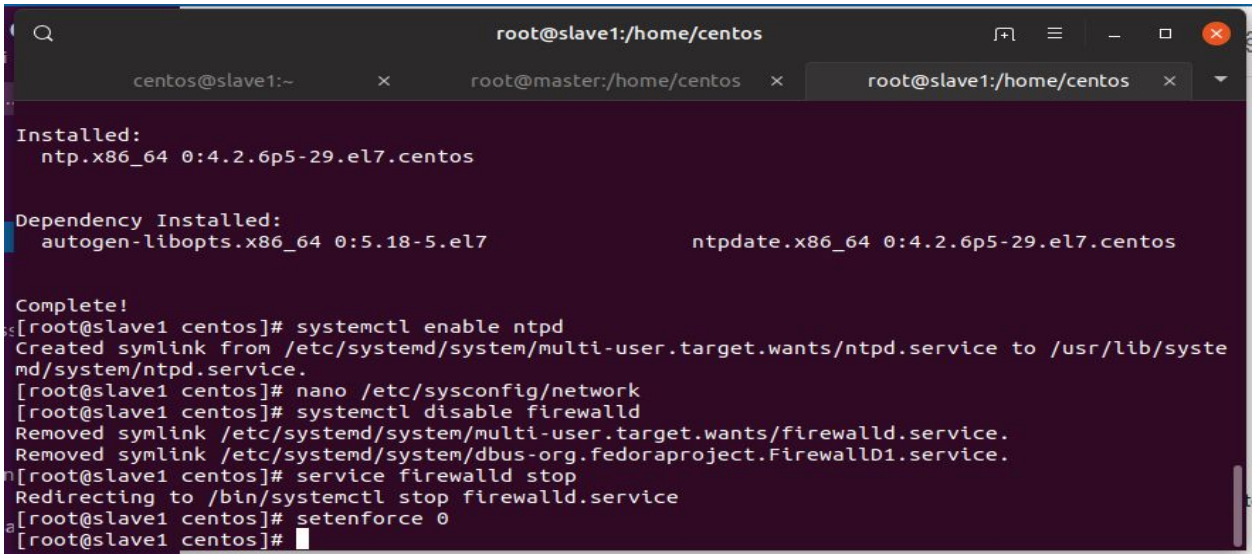
**** Configuring iptables

$ systemctl disable firewalld

$ service firewalld stop

***** Disable SELinux

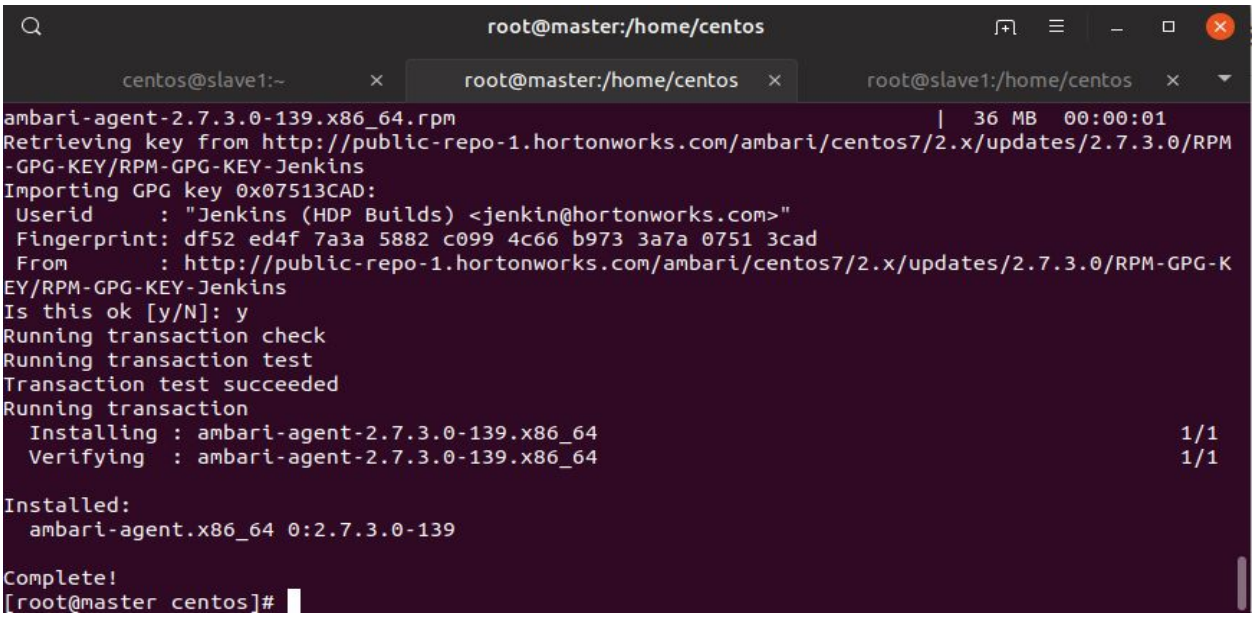
$ setenforce 0



6- Install ambari agent in all the hosts

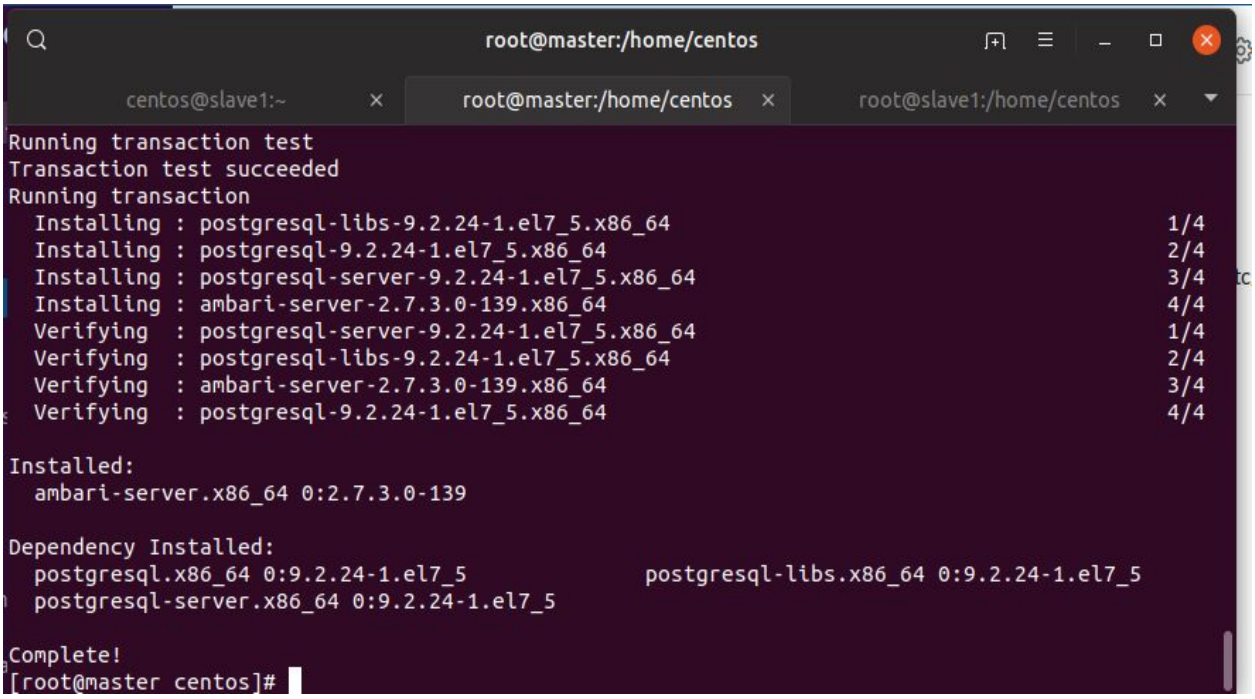
$ wget -nv http://public-repo-1.hortonworks.com/ambari/centos7/2.x/updates/2.7.3.0/ambari.repo -O /etc/yum.repos.d/ambari.repo

$ yum install ambari-agent



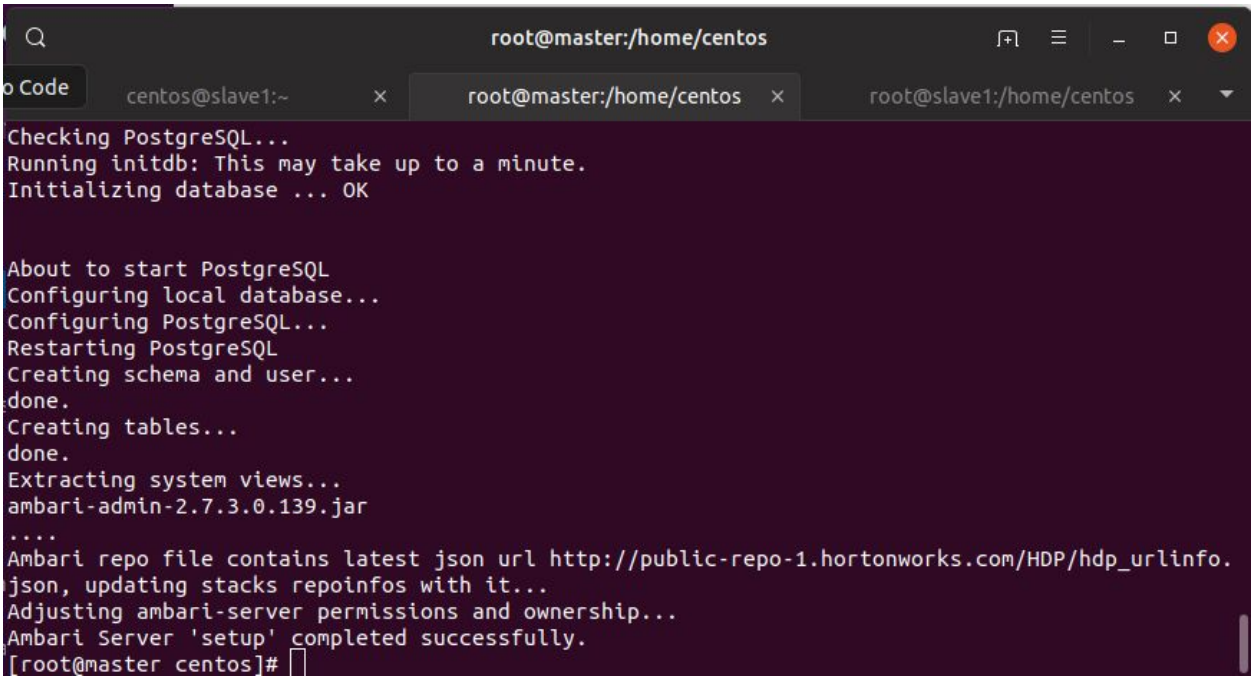
7- Install Ambari server

$ yum install ambari-server



8- Set up Ambari server

$ ambari-server setup



9- Edit ambari-agent config file

$ nano /etc/ambari-agent/conf/ambari-agent.ini

****

[server]

hostname=<ambari.server.hostname: master>

url_port=8440

secured_url_port=8441

****

10- start ambari server and agent

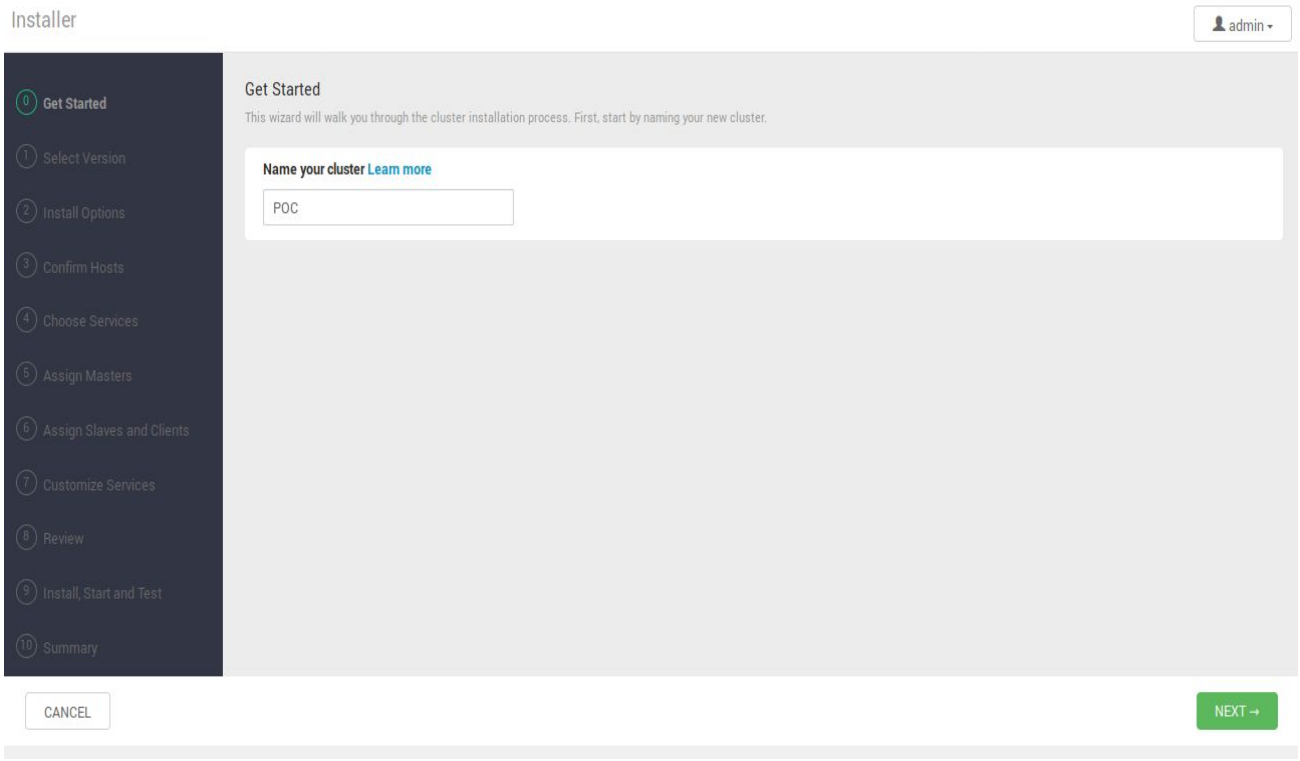
$ ambari-agent start

$ ambari-server start

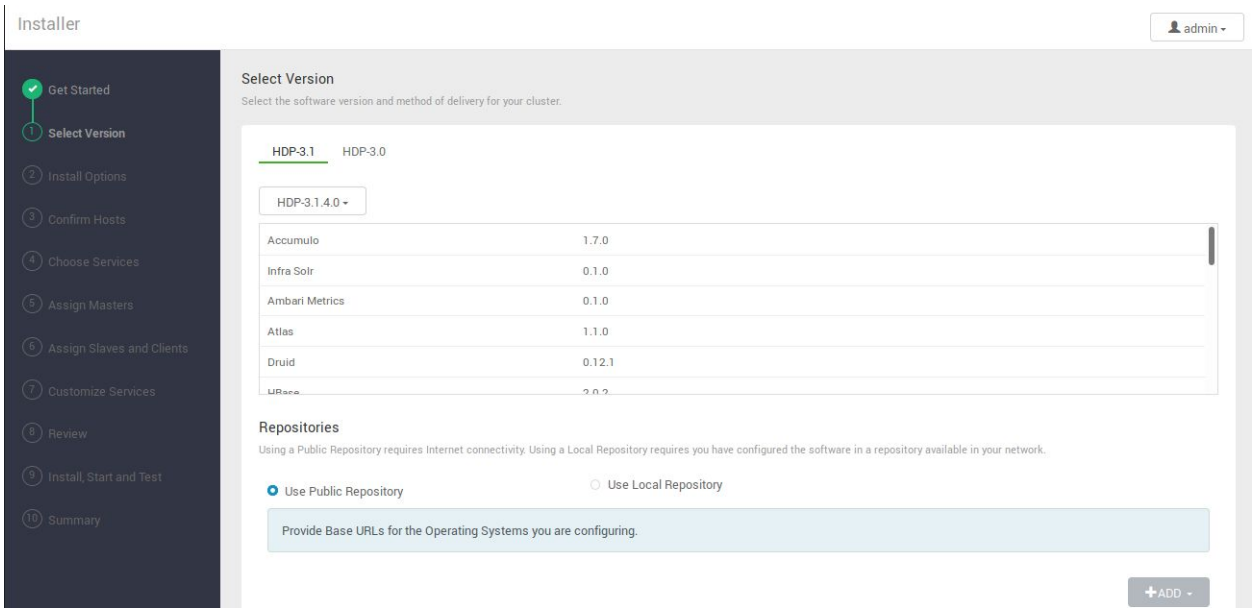
11- Enter to web ambari server web interface

Master:8080

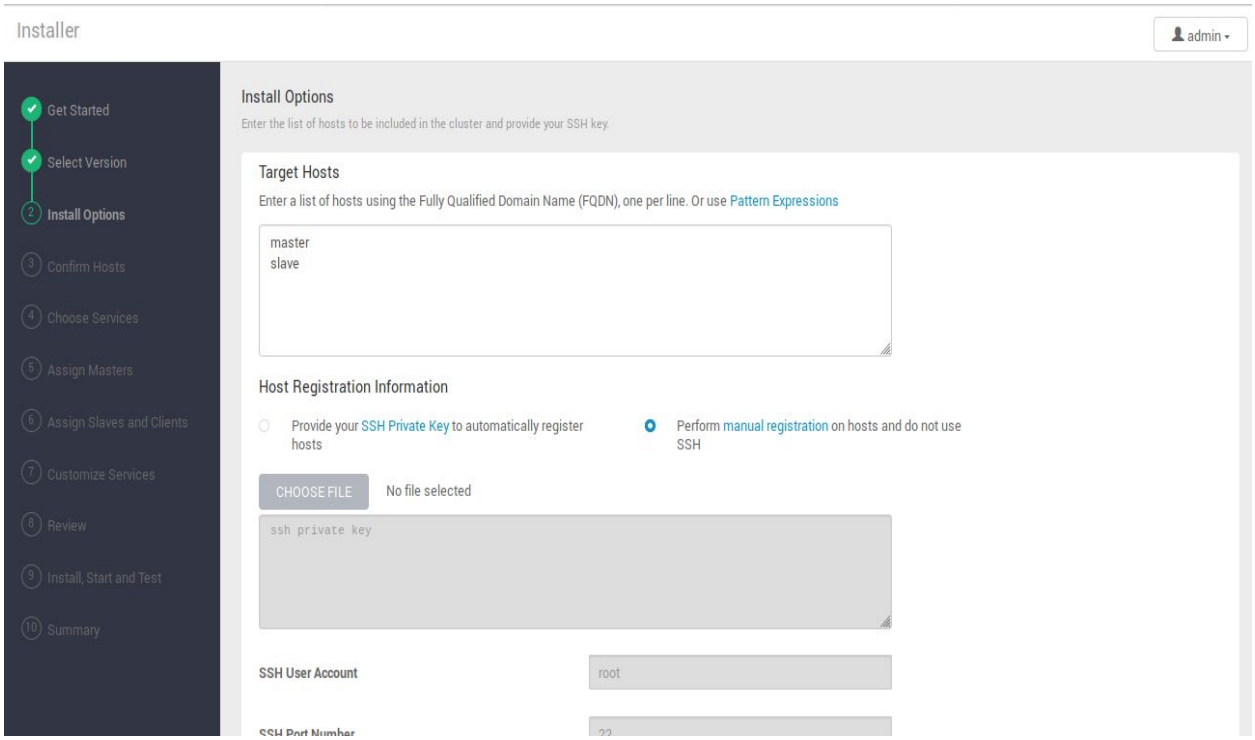
Set cluster name



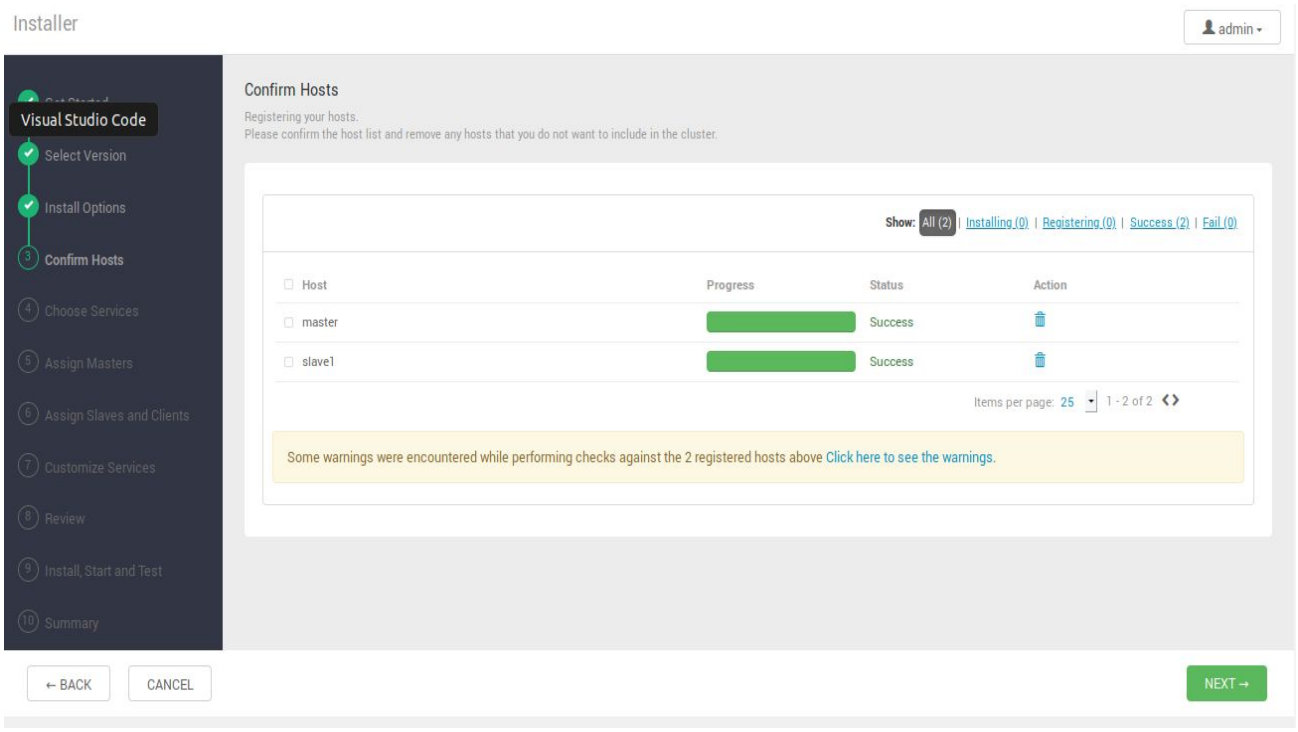
12- Install Ambari: select version



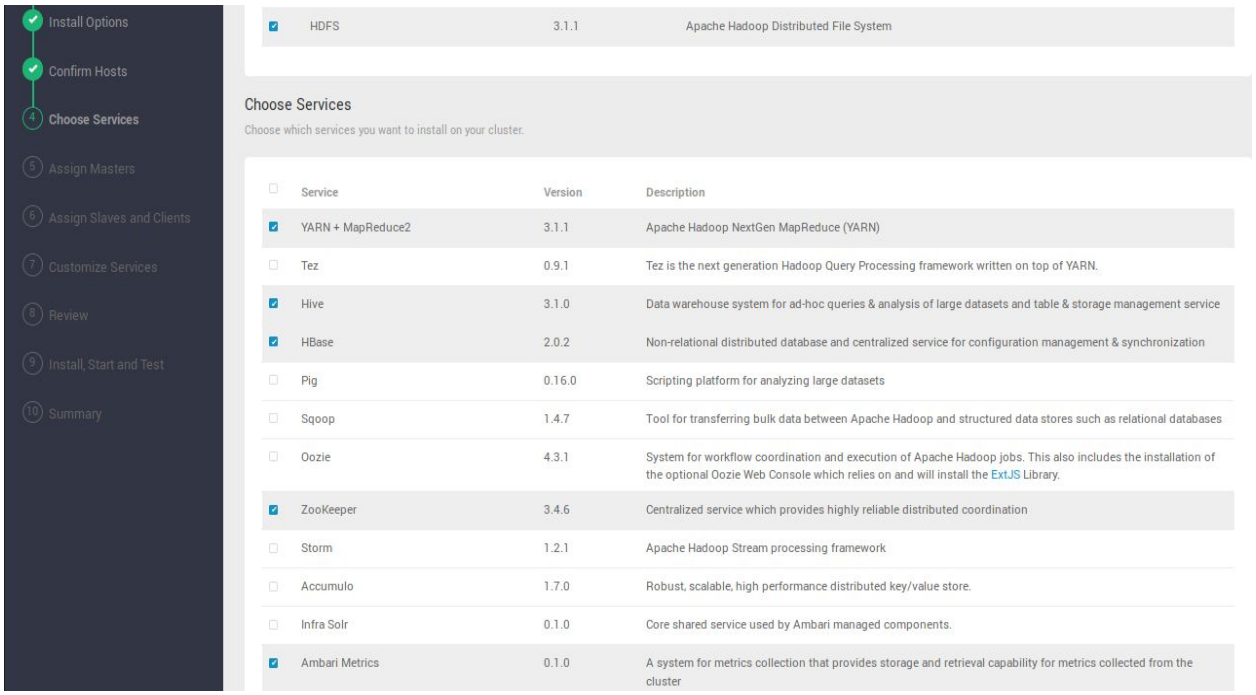
13- Set ambari hosts



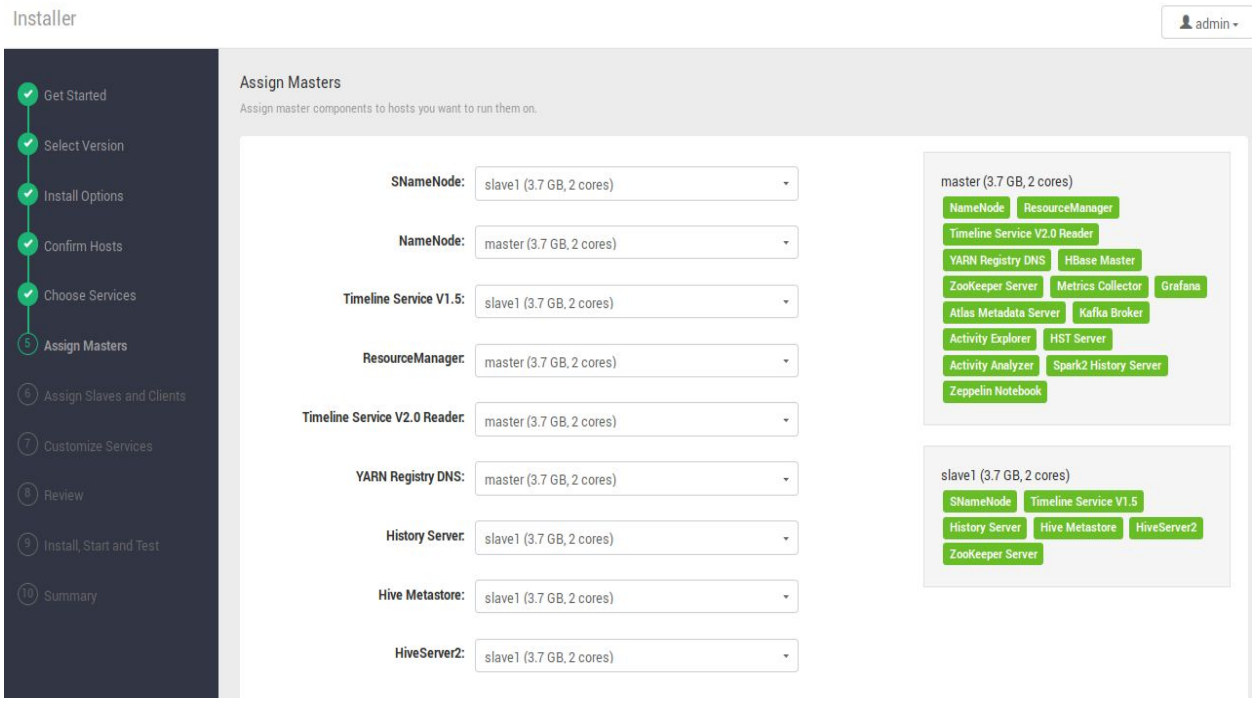
14- register hosts



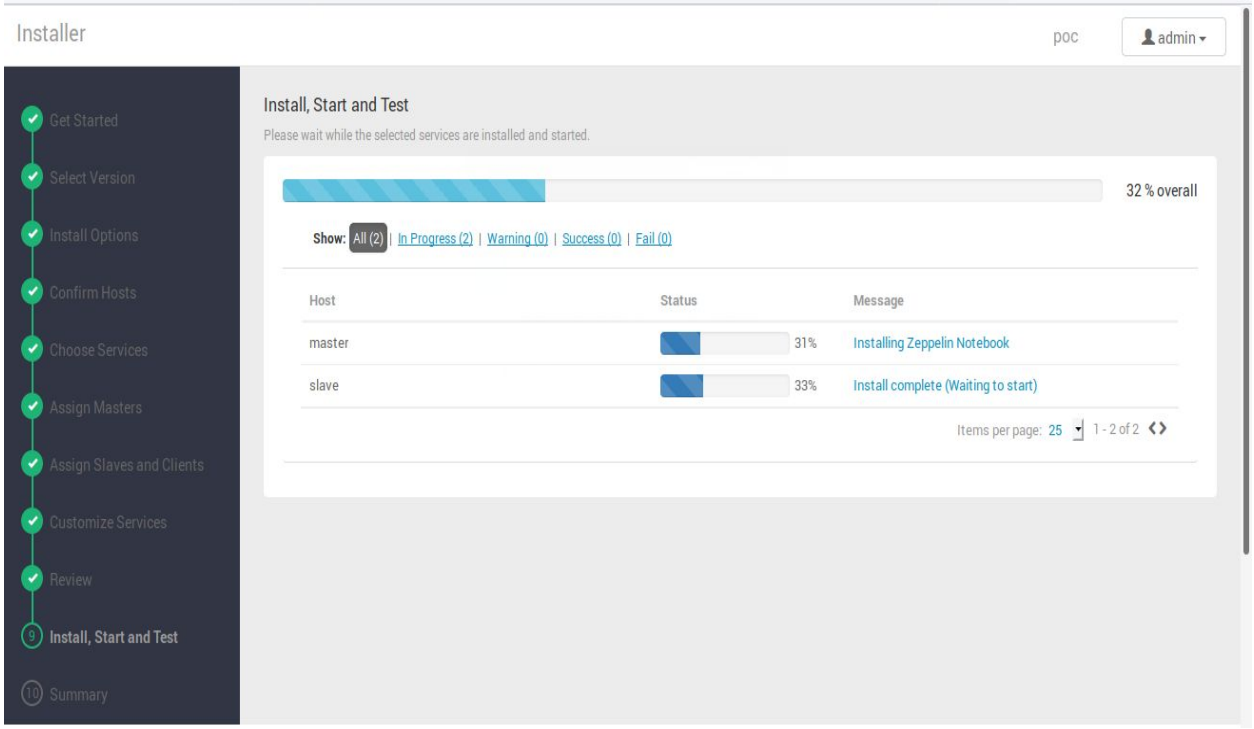
15- select services to install



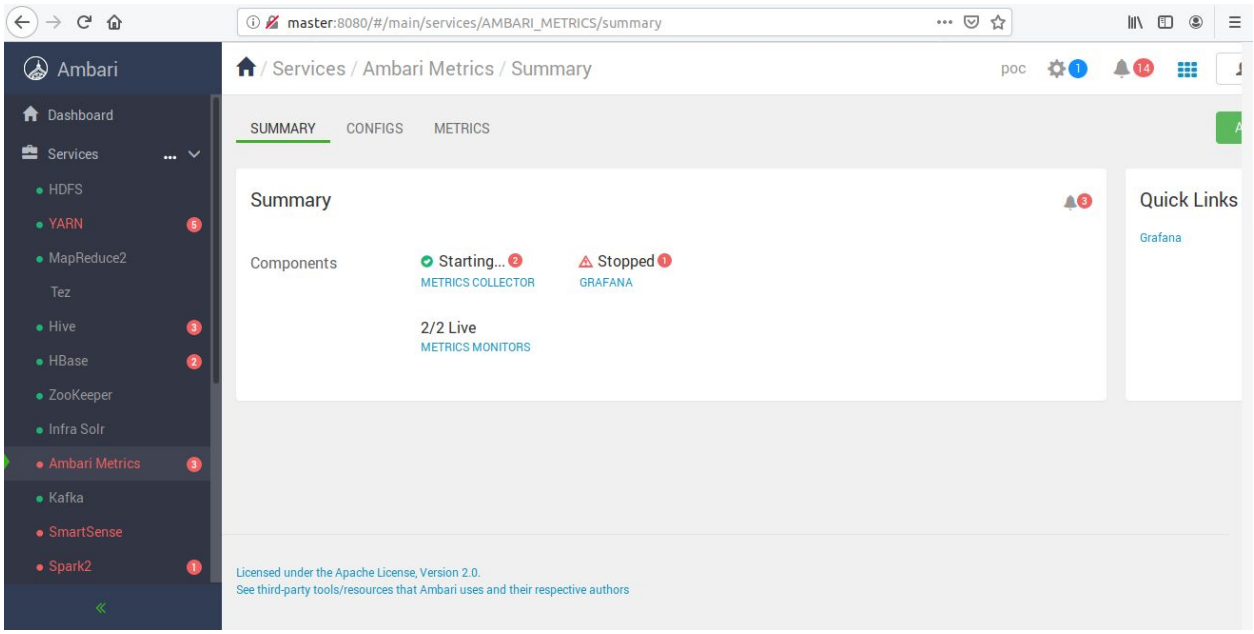
16- assign masters

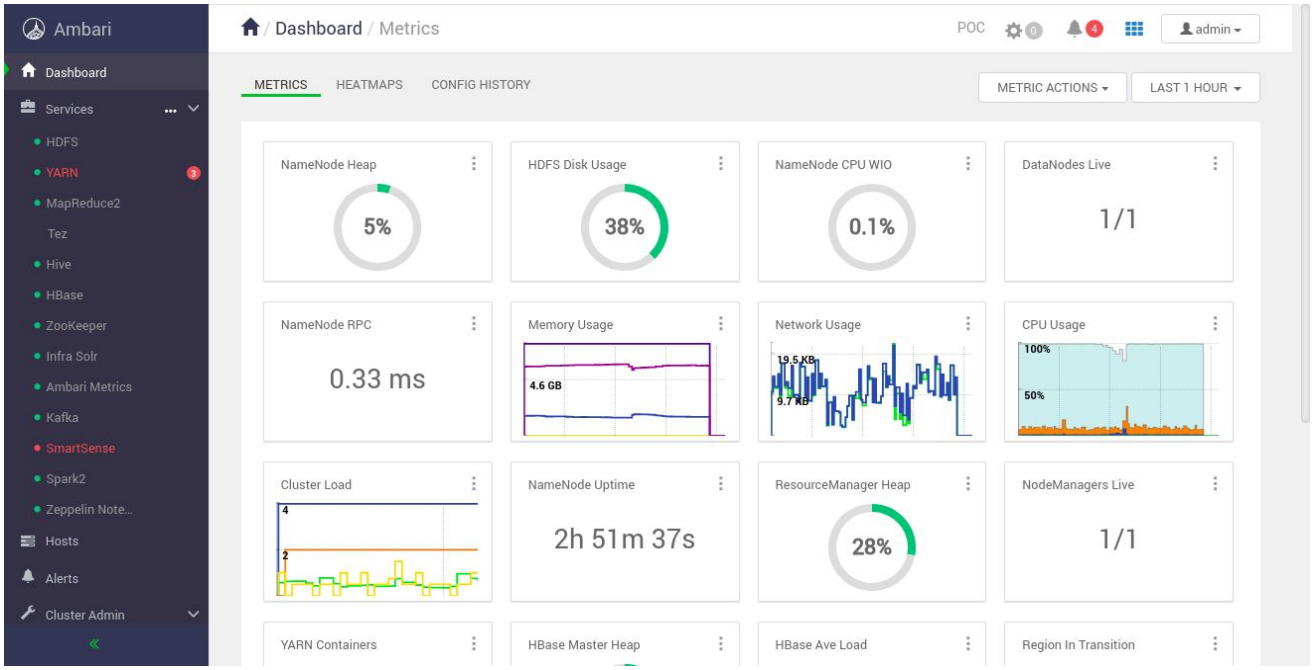


17- installing services



18- Ambri Dashboard: Hadoop, Spark et compagnies à partir de Ambari (Hortonworks)





II- Setting up Cassandra

Installing Cassandra steps 3.11.4 on CentOs 7.6

- Check the compatibility of the version of Cassanda with your java version
- Installation from Debian packages :

echo "deb http://www.apache.org/dist/cassandra/debian 311x main" | sudo tee -a /etc/apt/sources.list.d/cassandra.sources.list

- Update the repositories :
  sudo apt-get update
- install Cassandra :
  sudo apt-get install cassandra
- Commands for starting and stopping Cassandra :
  service cassandra start
- Command to check Cassandra status :
  service cassandra status

Create a Simple Cassandra Cluster with 3 Nodes

- Cassandra is installed on the 3 V.M (nodes)
- Each node has open communication between the other nodes.
- (disable the firewall on Cassandra)
- The IP addresses of each node are known.
- No data is stored on the 3 Cassandra instances.

  (If you've already started your Cassandra instance you'll need to stop it and remove the data it contains. The main reason for this is because the cluster_name needs to be the same on all nodes)

  ```
  service cassandra stop

  rm -rf /var/lib/cassandra/data/system/*
  ```

- Cassandra is configured using various files in the /etc/cassandra/conf directory.

  ```
  /etc/cassandra/conf/cassandra.yaml
  ```

  -cluster_name : 'poc'

  -seeds : are the IP addresses of the clusters seed servers

  -listen_address : the server IP

  -rpc_address : the server IP

  -endpoint_snitch :`GossipingPropertyFileSnitch`

- Edit the cassandra-rackdc.properties

  ```
  /etc/cassandra/conf/cassandra-rackdc.properties
  ```

- Troubleshooting operating timeouts :

  ```
  Cassandra imposes timeouts on read and write operations to prevent a
  given operation from negatively impacting the performance of the
  cluster. If you encounter an operation timeout, you can take the
  following actions to promote operation success.
  ```

```
sudo nano /etc/cassandra/cassandra.yaml

# How long the coordinator should wait for read operations to complete
read_request_timeout_in_ms: 50000
# How long the coordinator should wait for seq or index scans to complete
range_request_timeout_in_ms: 100000
# How long the coordinator should wait for writes to complete
write_request_timeout_in_ms: 20000
# How long the coordinator should wait for counter writes to complete
counter_write_request_timeout_in_ms: 50000
# How long a coordinator should continue to retry a CAS operation
# that contends with other proposals for the same row
cas_contention_timeout_in_ms: 10000
# How long the coordinator should wait for truncates to complete
# (This can be much longer, because unless auto_snapshot is disabled
# we need to flush first so we can snapshot before removing the data.)
truncate_request_timeout_in_ms: 600000
# The default timeout for other, miscellaneous operations
request_timeout_in_ms: 100000

# How long before a node logs slow queries. Select queries that take longer than
# this timeout to execute, will generate an aggregated log message, so that slow quer
# can be identified. Set this value to zero to disable slow query logging.
slow_query_log_timeout_in_ms: 5000
```

Importing the csv file into the Cassandra Cluster

- Create a keyspace :

```
tion refused")})
[root@slave1 default.conf]# cqlsh slave1
Connected to poc Cluster at slave1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.4 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> CREATE KEYSPACE weather WITH REPLICATION = { 'class' : 'SinpleStrategy', 'replication_factor': 1};
cqlsh> describe keyspaces

system_schema  system_auth  system  weather  system_distributed  systen_traces

cqlsh> use weather
   ... ;
```

- Create a table (a column family)

```
cqlsh:weather> CREATE table isd_history (USAF varchar, WBAN varchar,station_name varchar, country varchar, state varchar,ICAO varchar,LAT floa
t,LON float, ELEV float, begin varchar,end varchar, PRIMARY KEY (USAF, WBAN) ) ;
SyntaxException: line 1:152 mismatched input 'begin' expecting ')' (...LON float, ELEV float, [begin]...)
cqlsh:weather> CREATE table gsod (STN varchar, WBAN varchar,YEARMODA varchar, TEMP float, DEWP float, SLP float, STP float, VISIB float, WDSP
float, MXSPD float, GUST float, MAX float, MIN float, PRCP float, SNDP float, F varchar,R varchar,S varchar,H varchar,TH varchar,TOO varchar,P
RIMARY KEY ((STN, WBAN), YEARMODA));
cqlsh:weather> COPY gsod (STN, WBAN,YEARMODA, TEMP , DEWP , SLP , STP , VISIB , WDSP , MXSPD , GUST , MAX , MIN , PRCP , SNDP , F, R, S, H, TH
, TOO ) from '/home/centos/gsod.csv' with DELIMITER = ';';
Using 1 child processes

Starting copy of weather.gsod with columns [stn, wban, yearmoda, temp, dewp, slp, stp, visib, wdsp, mxspd, gust, max, min, prcp, sndp, f, r, s
, h, th, too].
Processed: 35000 rows; Rate:   12184 rows/s; Avg. rate:    9915 rows/s
```

- Copy the data on the table :

```
SyntaxException: line 1:5 no viable alternative at input 'keyscpace' ([drop] keyscpace...)
cqlsh> drop keyspace weather ;
cqlsh> CREATE KEYSPACE IF NOT EXISTS resto_NY WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor': 1};
cqlsh> CREATE KEYSPACE weather WITH REPLICATION = { 'class' : 'NetworkTopologyStrategy','dc1' : 1 ,'datacenter1' : 1};
cqlsh> use weather
   ... ;
cqlsh:weather> CREATE table gsod (STN varchar, WBAN varchar,YEARMODA varchar, TEMP float, DEWP float, SLP float, STP float, VISIB float, WDSP
float, MXSPD float, GUST float, MAX float, MIN float, PRCP float, SNDP float, F varchar,R varchar,S varchar,H varchar,TH varchar,TOO varchar,
 PRIMARY KEY ((STN, WBAN), YEARMODA));
cqlsh:weather> COPY gsod (STN, WBAN,YEARMODA, TEMP , DEWP , SLP , STP , VISIB , WDSP , MXSPD , GUST , MAX , MIN , PRCP , SNDP , F, R, S, H, TH
, TOO ) from '/home/centos/gsod.csv' with DELIMITER = ';';
Using 1 child processes

Starting copy of weather.gsod with columns [stn, wban, yearmoda, temp, dewp, slp, stp, visib, wdsp, mxspd, gust, max, min, prcp, sndp, f, r, s
, h, th, too].
Processed: 35000 rows; Rate:   11061 rows/s; Avg. rate:    9206 rows/s
```

- Result after importing data

```
Failed to import 20 rows: Unavailable - Error from server: code=1000 [Unavailable exception] message="Cannot achieve consistency level ONE" in
fo={'required_replicas': 1, 'alive_replicas': 0, 'consistency': 'ONE'},  given up after 5 attempts
Failed to import 20 rows: Unavailable - Error from server: code=1000 [Unavailable exception] message="Cannot achieve consistency level ONE" in
fo={'required_replicas': 1, 'alive_replicas': 0, 'consistency': 'ONE'},  given up after 5 attempts
Failed to import 20 rows: Unavailable - Error from server: code=1000 [Unavailable exception] message="Cannot achieve consistency level ONE" in
fo={'required_replicas': 1, 'alive_replicas': 0, 'consistency': 'ONE'},  given up after 5 attempts
Failed to import 20 rows: Unavailable - Error from server: code=1000 [Unavailable exception] message="Cannot achieve consistency level ONE" in
fo={'required_replicas': 1, 'alive_replicas': 0, 'consistency': 'ONE'},  given up after 5 attempts
Failed to import 20 rows: Unavailable - Error from server: code=1000 [Unavailable exception] message="Cannot achieve consistency level ONE" in
fo={'required_replicas': 1, 'alive_replicas': 0, 'consistency': 'ONE'},  given up after 5 attempts
Failed to import 20 rows: Unavailable - Error from server: code=1000 [Unavailable exception] message="Cannot achieve consistency level ONE" in
fo={'required_replicas': 1, 'alive_replicas': 0, 'consistency': 'ONE'},  given up after 5 attempts
Failed to import 20 rows: Unavailable - Error from server: code=1000 [Unavailable exception] message="Cannot achieve consistency level ONE" in
fo={'required_replicas': 1, 'alive_replicas': 0, 'consistency': 'ONE'},  given up after 5 attempts
Failed to import 5 rows: Unavailable - Error from server: code=1000 [Unavailable exception] message="Cannot achieve consistency level ONE" inf
o={'required_replicas': 1, 'alive_replicas': 0, 'consistency': 'ONE'},  given up after 5 attempts
Failed to import 5 rows: Unavailable - Error from server: code=1000 [Unavailable exception] message="Cannot achieve consistency level ONE" inf
o={'required_replicas': 1, 'alive_replicas': 0, 'consistency': 'ONE'},  given up after 5 attempts
Failed to import 20 rows: Unavailable - Error from server: code=1000 [Unavailable exception] message="Cannot achieve consistency level ONE" in
fo={'required_replicas': 1, 'alive_replicas': 0, 'consistency': 'ONE'},  given up after 5 attempts
Failed to import 20 rows: Unavailable - Error from server: code=1000 [Unavailable exception] message="Cannot achieve consistency level ONE" in
fo={'required_replicas': 1, 'alive_replicas': 0, 'consistency': 'ONE'},  given up after 5 attempts
Failed to import 20 rows: Unavailable - Error from server: code=1000 [Unavailable exception] message="Cannot achieve consistency level ONE" in
fo={'required_replicas': 1, 'alive_replicas': 0, 'consistency': 'ONE'},  given up after 5 attempts
Failed to import 20 rows: Unavailable - Error from server: code=1000 [Unavailable exception] message="Cannot achieve consistency level ONE" in
fo={'required_replicas': 1, 'alive_replicas': 0, 'consistency': 'ONE'},  given up after 5 attempts
Failed to import 20 rows: Unavailable - Error from server: code=1000 [Unavailable exception] message="Cannot achieve consistency level ONE" in
fo={'required_replicas': 1, 'alive_replicas': 0, 'consistency': 'ONE'},  given up after 5 attempts
Failed to import 20 rows: Unavailable - Error from server: code=1000 [Unavailable exception] message="Cannot achieve consistency level ONE" in
fo={'required_replicas': 1, 'alive_replicas': 0, 'consistency': 'ONE'},  given up after 5 attempts
Exceeded maximum number of insert errors 1000
Failed to process 1007 rows; failed rows written to import_weather_gsod.err
Exceeded maximum number of insert errors 1000
Processed: 97580000 rows; Rate:    6203 rows/s; Avg. rate:   11775 rows/s
97580000 rows imported from 0 files in 2 hours, 18 minutes, and 7.289 seconds (0 skipped).
cqlsh:weather> Connection to slave1 closed by remote host.
Connection to slave1 closed.
```

III- Download and clean data

To download and clean the data we used a bash script.

```
#!/bin/bash
for year in {1930..1950}; do
# Create a directory for the year
   if [ ! -d "$year" ]; then
   echo Creating a directory.
       mkdir -p $year
   fi
# Download the data (if we don't already have it)
   if [ ! -f "$year/gsod_$year.tar" ]; then
   wget ftp://ftp.ncdc.noaa.gov/pub/data/gsod/$year/gsod_$year.tar -O
$year/gsod_$year.tar
   fi

# Unzip the data
   if [ -f "$year/gsod_$year.tar" ]; then
   echo Unzipping the downloaded data.
       tar -xvf $year/gsod_$year.tar -C $year/
       rm $year/gsod_$year.tar
       for filename in `ls $year/*.gz`; do
           gunzip $filename
       done
   fi
done
for year in {1930..1950}; do
# Strip the first line of each of the .op files
   for filename in `ls $year/*.op`; do
       tail -n +2 $filename > $filename.header_stripped
```

```
        mv $filename.header_stripped $filename
    done
# Stack the .op files
    cat $year/*.op > $year.op
    rm -rf $year
done

#converting to csv
if [ -f "gsod.csv" ]; then
    rm gsod.csv
fi
for year in {1930..1950}; do
    echo $year
    awk '{printf
"%s;%s;%s;%.1f;%.1f;%.1f;%.1f;%.1f;%.1f;%.1f;%.1f;%.1f;%.2f;%.1f;%s;%s
;%s;%s;%s;%s\n",$1,$2,$3,$4,$6,$8,$10,$12,$14,$16,$17,substr($0,103,6),subs
tr($0,111,6),substr($0,119,5),$21,substr($22,1,1),substr($22,2,1),substr($2
2,3,1),substr($22,4,1),substr($22,5,1),substr($22,6,1)}' OFS=';' $year.op
>> gsod2.csv;
    rm $year.op
```

IV- Data processing with Pyspark:

```
import pandas as pd
import pyspark
from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession
from functools import reduce
from past.builtins import xrange
from datetime import datetime
from pyspark.sql.functions import col, udf
from pyspark.sql.types import DateType
from pyspark.sql.functions import *

sc = SparkContext("local","gsod")
spark = SparkSession(sc)
#import gsod data into dataframe
df =
spark.read.format("csv").option("delimiter",";").load("hdfs:///poc/data/gso
d.csv")
#import station history data into dataframe
```

```
hi =
spark.read.format("csv").option("header","true").load("hdfs:///poc/data/isd
-history.csv")
#import country list data into dataframe
cn =
spark.read.format("csv").option("delimiter",";").option("header","true").lo
ad("hdfs:///poc/data/country-list.csv")
#import continent list into dataframe
co =
spark.read.format("csv").option("delimiter",";").option("header","true").lo
ad("hdfs:///poc/data/continent.csv")

# Add header
oldColumns = df.schema.names
newColumns = ["STN", "WBAN","YEARMODA", "TEMP" , "DEWP" , "SLP" , "STP" ,
"VISIB" , "WDSP" , "MXSPD" , "GUST" , "MAX" , "MIN" , "PRCP" , "SNDP" ,
"F", "R", "S", "H", "TH", "TOO"]
df = reduce(lambda df, idx: df.withColumnRenamed(oldColumns[idx],
newColumns[idx]), xrange(len(oldColumns)), df)
# set column types
types=["string","string","string","float","float","float","float","float","
float","float","float","float","float","float","float","string","string","S
tring","string","string","String"]
df = reduce(lambda df, idx: df.withColumn(newColumns[idx],
df[newColumns[idx]].cast(types[idx])),xrange(len(newColumns)), df)
# set date type column
func = udf(lambda x: datetime.strptime(x,'%Y%m%d'), DateType())
df = df.withColumn('YEARMODA', func(col('YEARMODA')))
# change temp from F to C
tocColumns=[ "TEMP" , "DEWP" , "MAX" , "MIN" ]
df= reduce(lambda df, idx:
df.withColumn(tocColumns[idx],when(df[tocColumns[idx]]
<9999.9,round((df[tocColumns[idx]]-32)*5/9,1))),xrange(len(tocColumns)),
df)
#join continent table to country
cn=cn.join(co,on='ID',how='left')
# join country table to station history
df3 = hi.join(cn, hi.CTRY == cn.ID,how='left').drop(cn.ID)
# join all data
df4 = df.join(df3, (df.STN == df3.USAF) & (df.WBAN ==
df3.WBAN),how='left').drop(df3.USAF).drop(df3.WBAN)

# basic data by year
df7=df4.select('STN','WBAN','YEARMODA','COUNTRY').groupby(year('YEARMODA').
alias('Year')).agg(countDistinct('STN','WBAN').alias('No of
station'),count('STN').alias('No of
lines'),countDistinct('COUNTRY').alias('No of countries')).orderBy('Year')
```

```
df7.toPandas().to_csv('year.csv',index=False)

# mean temp by decade
df5=df4.groupby(floor(year('YEARMODA')/10)
%10).agg(mean('TEMP').alias("Mean
Temp"),count('TEMP')).withColumnRenamed('(FLOOR((year(YEARMODA) / 10)) %
10)', 'decade')
df5.toPandas().to_csv('mean.csv',index=False)
#mean temp by decade per continent
df5c=df4.groupby(floor(year('YEARMODA')/10)
%10,'CONTINENT').agg(mean('TEMP').alias("Mean
Temp"),count('TEMP')).withColumnRenamed('(FLOOR((year(YEARMODA) / 10)) %
10)', 'decade').orderBy('decade')
df5c.toPandas().to_csv('mean_cont.csv',index=False)

# station by year
df8=df4.select('STN','WBAN',year('YEARMODA').alias('year'),'STATION
NAME','COUNTRY','LAT','LON').distinct()
df8.toPandas().to_csv('station.csv',index=False)

# Temp with decade and continent
dfbb=df4.where(year('YEARMODA')>1969)
dfB=dfbb.select((floor(year('YEARMODA')/10)
%10).alias('decade'),'TEMP','CONTINENT')
dfB.write.csv("hdfs:///poc/data/decade_cont.csv")
#dfB.toPandas().to_csv('decade_cont.csv',index=False)

# Min and Max temp every year
df21=df4.select(year('YEARMODA').alias('Year'),'STATION
NAME','COUNTRY','MIN').groupBy('Year').agg(min('MIN').alias('MIN')).orderBy
('Year')
df22=df4.select(year('YEARMODA').alias('Year'),'STATION
NAME','COUNTRY','MAX').groupBy('Year').agg(max('MAX').alias('MAX')).orderBy
('Year')
df11=df4.select(year('YEARMODA').alias('Year'),'YEARMODA','STATION
NAME','COUNTRY','MIN')
df12=df4.select(year('YEARMODA').alias('Year'),'YEARMODA','STATION
NAME','COUNTRY','MAX')
df13=df4.select(year('YEARMODA').alias('Year'),'STATION
NAME','COUNTRY','TEMP')
df31=df21.join(df11,(df21.MIN==df11.MIN)&(df21.Year==df11.Year),how='left')
.drop(df11.Year).drop(df11.MIN).orderBy('Year')
df32=df22.join(df12,(df22.MAX==df12.MAX)&(df22.Year==df12.Year),how='left')
.drop(df12.Year).drop(df12.MAX).orderBy('Year')
df31.toPandas().to_csv('min.csv',index=False)
df32.toPandas().to_csv('max.csv',index=False)
```

```
# rainy days per year
dfr=df4.select(year('YEARMODA').alias('Year'),'R').crosstab('Year','R').ord
erBy('Year_R')
dfr.toPandas().to_csv('rain.csv',index=False)

# snowy days per year
dfs=df4.select(year('YEARMODA').alias('Year'),'S').crosstab('Year','S').ord
erBy('Year_S')
dfs.toPandas().to_csv('snow.csv',index=False)
```