***2014.11.16***

1. Spin up Ubuntu server VM

I called it TachyonCluster\_Master

2. Install AWS EC2 CLI tools

Had to install Java

sudo apt-get install openjdk-7-jre

I had to create an IAM user and add the Administrator User Policy:

the user's name is nieltown

saved the credential in credential.csv

I added ec2-api-tools.sh to set up environment variables:

put it in /etc/profile.d/ec2-api-tools.sh

^^^ you don't have to make these executable; they'll just run every time you start a bash session

had to set up 4 different environment variables for the keys

3. Set Up Spark & Tachyon on EC2

Following this guide:

<https://github.com/amplab/tachyon/wiki/Running-Tachyon-on-EC2>

I did all of this from TachyonCluster\_Master, specifying

-a ami-5bb18832

^^^ (as per

<https://github.com/mesos/spark-ec2/blob/v4/ami-list/us-east-1/pvm> )

-k DriveStoredKey

-i ~/keys/DriveStoredKey.pem

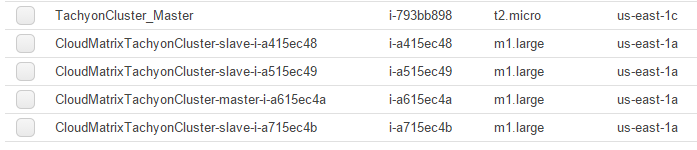
-s 3

^^^ that's three (3) slaves

-w 600

^^^ wait 600 seconds (read that somewhere in a forum post; not sure if it's actually doing anything for me)

That in turn created 4 EC2 instances:



Took a long time (I did say to wait 600 seconds after all), but it did a lot of setup on the servers it created (software installation, etc)

I noticed installing these things:

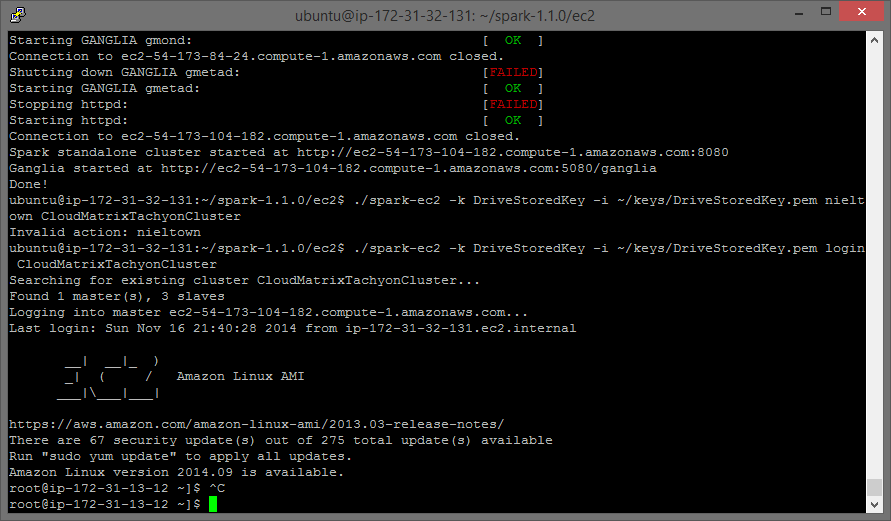
* Scala
* Spark/Shark
* Hadoop (1.0.4)
* Tachyon (0.4.1)
* Ganglia

After it finished, I ran this command:

./spark-ec2 -k DriveStoredKey -i ~/keys/DriveStoredKey.pem \

login CloudMatrixTachyonCluster

And it logged me into the Cluster



I ran ‘sudo yum update’ and updated all my shit (446 M)

4. Setting up Git on my local VM

This is the local VM on my own desktop. I set up Git on it in the same directory as the Eclipse project, i.e. /home/nieltown/cloudmatrix

This is what I did to do that:

cd cloudmatrix/

touch README.md

git init

git add README.md

git commit -m "first commit"

git remote add origin [git@github.com:nieltown/CloudMatrix.git](mailto:git@github.com:nieltown/CloudMatrix.git)

git push -u origin master

But then there was an error! I had to fix my public key. So to do that, I took the following steps:

ssh-keygen -t rsa -C [mr.john.nielson@gmail.com](mailto:mr.john.nielson@gmail.com)

ssh-agent –s

eval $(ssh-agent)

ssh-add ~/.ssh/id\_rsa

Then I copied the key to my clipboard and added it to GitHub via the web interface:

***[REDACTED]***

So then I went back to try to push to GitHub again:

cd ~/cloudmatrix

git push -u origin master

And everything worked fine.

Next I added the contents of the project to GitHub:

git add \*

git commit -m "initial load of project files and flask app"

git push –u origin master

Note that this will NOT add empty directories. I also added this document to the repo – wow!!

5. Set Up Spark & Tachyon on EC2 – Part II

Remember in 3. I logged into the Spark cluster using this command:

./spark-ec2 -k DriveStoredKey -i ~/keys/DriveStoredKey.pem \

login CloudMatrixTachyonCluster

Then I updated all my yum packages. Well, I’ve gone back to the Spark cluster now.

Time to config Tachyon on the Spark cluster.

cd /root/tachyon/conf

cp tachyon-env.sh.template tachyon-env.sh # creating the config script

Added the following to tachyon-env.sh:

export TACHYON\_HDFS\_ADDRESS=hdfs://HDFS\_HOSTNAME:HDFS\_PORT

Edit Spark /root/spark/conf/spark-env.sh by adding to top:

export SPARK\_CLASSPATH+=/root/tachyon/target/tachyon-0.4.1-SNAPSHOT-jar-with-dependencies.jar

SPARK\_JAVA\_OPTS+=" -Dtachyon.hdfs.address=hdfs://HDFS\_HOSTNAME:HDFS\_PORT -Dspark.default.parallelism=1 "

export SPARK\_JAVA\_OPTS

Edit Spark’s hdfs-site.xmls by adding:

<property>

<name>fs.tachyon.impl</name>

<value>tachyon.hadoop.TachyonFileSystem</value>

<description></description>

</property>

There are multiple hdfs-site.xml files; they can be found here:

/root/spark-ec2/templates/root/ephemeral-hdfs/conf

/root/spark-ec2/templates/root/mapreduce/conf

/root/spark-ec2/templates/root/persistent-hdfs/conf/hdfs-site.xml

Sync configuration to all nodes:

cd /root/tachyon/conf

/root/spark-ec2/copy-dir.sh .

I went back and took OUT the stuff from /root/spark/conf/spark-env.sh about the SPARK\_CLASSPATH and SPARK\_JAVA\_OPTS, i.e. these lines:

export SPARK\_CLASSPATH+=/root/tachyon/target/tachyon-0.4.1-SNAPSHOT-jar-with-dependencies.jar

SPARK\_JAVA\_OPTS+=" -Dtachyon.hdfs.address=hdfs://HDFS\_HOSTNAME:HDFS\_PORT -Dspark.default.parallelism=1 "

export SPARK\_JAVA\_OPTS

Spark won’t create a working SparkContext with those lines in there for some reason.

Run Spark (this takes a while):

/root/spark/bin/spark-shell

Found out that my AWS account had been compromised. Attackers started about 160 servers under my account. Waiting to hear back from Amazon about billing remedies, e.g. refunds, credits, or whatever to cover the almost $700 in charges created by the attackers. Shut everything down, deleted all instances, and deleted my IAM role. Starting over with VoltDB.

***2014.11.17***

Gonna try VoltDB today. Tachyon isn’t really what I want anyway. I don’t want a filesystem. I want a database. I plan to run VoltDB on a cluster. I also got a 30-day trial license for MemSQL today. I might try that if VoltDB doesn’t work out for some reason.

In light of recent AWS account-related circumstances, I’ll be doing everything on a local VM for a while.

***2014.11.22***

Screw everything I said last time. BTW, I heard back from Amazon and they refunded all the charges incurred by the account compromise. I even got back my $100 in credits from Dr. Roussev. Anyway, I’m setting up a Redis cluster a la this guide:

<http://docs.aws.amazon.com/AmazonElastiCache/latest/UserGuide/GettingStarted.CreateCluster.Redis.html>

Key-value store makes more sense anyway since I’m just going to pickle big matrices. I might try to do something with databases for matrix transformations and see the difference between the two, but that won’t make any difference if I’m just using NumPy – only if I’m writing my own algorithms.

Just finished the system diagram, cloudmatrix\_SystemDiagram.vsdx aww yeah

Setting up Ubuntu Server EC2 instance. Installing the following:

sudo apt-get install python-pip

sudo pip install flask

sudo pip install kazoo

**ZooKeeper setup:**

Setting up ZooKeeper server. Creating an image from this. You have to open the following ports:

2181 (client port)

2888 (peer port)

3888 (peer leader election port)

Using Virtual Private Cloud (VPC) to create static IPs for servers. Setting this up for all compute and ZK servers. Hopefully I can do the same for the Redis cluster, though I don’t know if that’s even necessary.

Had to

sudo apt-get update

sudo apt-get install openjdk-7-jre-headless

Edited zk.Env.sh to change the ZooKeeper bin directory to where I stupidly installed it in ~/

Edited the init.d script

* to point to the correct location of zkEnv.sh
* changed shebang at top of init.d script to read

#!/usr/bin/env bash

* changed

chown zookeeper:hadoop ${ZOOPIDDIR}

to

chown ubuntu:ubuntu ${ZOOPIDDIR}

* changed calls to

${ZOOKEEPER\_PREFIX}/sbin/zkServer.sh

to

${ZOOKEEPER\_PREFIX}/bin/zkServer.sh

* removed ‘-c zookeeper’ option to just run as default user (was throwing “user ‘zookeeper’ not found error”

Then I copied the init.d script to /etc/init.d and chmodded 0755 the file

Also created a key pair for each ZK server to use to communicate with one another. Not sure if that did anything.

Create a myid file in the ZooKeeper dataDir. It’s a file with a single line representing which server that machine is in the quorum. So if you’re creating the myid file for server.1, the file should just have

1

in it.

**USE THE HOSTNAMES - DON’T USE THE IP**

**IT’S A VPC-SPECIFIC THING**

**ALSO**

**SUDO EVERY TIME YOU RUN zkServer.sh**

***2014.11.23***

Got ZooKeeper back up and running. I changed the host names of the EC2 servers to

cloudmatrixwebserver

cloudmatrixzk01

cloudmatrixzk02

cloudmatrixzk03

Used

sudo hostname [hostname]

Also edited

/etc/hosts

/etc/hostname

to reflect changes.

Rebooted dems.

I just installed and setup Remote System Explorer in Eclipse. I had to go into

Window -> Preferences -> General -> Network Connections -> SSH2

and then added DriveStoredKey.pem to my private keys (“Add Private Key…” button).

Also, hidden files were hidden by default in the Remote Systems Explorer via Sftp Files -> nieltown-pc:ssh.file -> Root -> / -> home -> ubuntu (for example), so I fixed that by going into

Window -> Preferences -> Remote Systems -> Files

and checking the box labeled “Show hidden files”

*btw, the version of Eclipse I’m using is Android Developer Tools build: v22.0.5-757759*

I can open a terminal and browse my filesystem and everything on EC2 boxes from Eclipse. That is fucking tight.

Had to add a new SSH key for cloudmatrixwebserver. I cloned my GitHub repo for CloudMatrix, right-clicked on that directory in Remote Systes, and used “Create Remote Project.” Now it’s a straight-up PyDev project but linked to an EC2 server. Dope!

**Compute Node Setup**

I’m going to set up the compute nodes now. I want to install Kazoo on them to watch ZK nodes. When a node is updated, i.e. one of the compute nodes has an instruction, the compute node will determine what that instruction is and perform it.

So it goes with the setup (ughh):

$ sudo apt-get update

$ sudo apt-get install python-pip

$ sudo pip install kazoo

$ sudo apt-get install python-numpy python-scipy \

python-matplotlib ipython ipython-notebook python-pandas \

python-sympy python-nose

$ sudo apt-get install git

$ sudo pip install pyzmq-static

$ sudo pip install requests

$ sudo pip install mmh3

Copied

~/.ssh/id\_rsa

~/.ssh/id\_rsa.pub

From cloudmatrix\_webserver to cloudmatrix\_compute01. You can just copy and paste from Remote System to Remote System in the GUI. It’s really nice, like having a ton of WinSCP windows open at once, but easily organized in trees inside your Eclipse.

Cloned the CloudMatrix project from GitHub to cloudmatrix\_compute01.

OK, so in the GitHub project there are two applications (as of now):

1. Flask application; currently it’s really just views.py in ./app which handles the REST requests. It dispatches the requests via ZeroMQ to compute servers, choosing which one to use via ZooKeeper. It then passes the request on to the compute server to do work
2. compute\_listener.py in ./compute runs on the compute servers and listens for requests via socket connection built via ZeroMQ.

**TO DO:**

1. Flesh out the REST API to handle the requests
2. Write code for the Flask server to query ZooKeeper for available compute nodes
3. Write routines for computation to be run on compute nodes
4. Write routines for storage and retrieval of data in/from Redis