**User document for cluster & Spark**

1. **Glossary**

There are plenty of glossaries in Spark and AWS, it would be helpful to clarify them before we start our journey.

1. **Spark/Apache Spark**: is a fast and general engine for large-scale data processing. It contains 4 libraries: Spark SQL and DataFrame, Spark Streaming, Spark MLlib and Graph X. Basically we could focus on Spark MLlib for machine learning and Spark SQL for structured data.

Spark supports Scala, Java, Python and R, but as you may know it doesn’t support R well, we may focus on Python for now.

Website: <http://spark.apache.org/>.

1. **Spark application**: User program built on Spark.
2. **Driver program**: The process running the main() function of the application and creating the SparkContext
3. **Deploy mode, worker mode, executor, task, job, stage:** I am still digesting them now, see definitions in http://spark.apache.org/docs/latest/cluster-overview.html
4. **AWS(Amazon Web Services)** is the cloud solution provide by Amazon. The cloud computing refers to the on-demand delivery of IT resources and applications via the Internet with pay-as-you-go pricing.
5. **EC2(Amazon Elastic Compute Cloud)** provides scalable computing capacity in the AWS. We can see it as a virtual server host.
6. **Instance** is the virtual computing environment; we can see it as a host/server in cloud.
7. **Instance types:** different types provide various configuration of CPU, memory, storage, network capacity and even purpose. See all the types in <http://aws.amazon.com/ec2/instance-types/>.
8. **Key pairs** could provide the secure login information. Dan already sent you the credential for logging in master node(I will talk about node later), the key pair is different from the credential, it would help you in 2 ways: a) logging in a passwordless way; b)create Jupyter server on master node which could be connected from local laptop. The key pairs contains 2 parts, public key and private key, AWS stores the public key, and we store private key in local.
9. **Instance storage volume:** the instance could store temporary data, once the instance is stopped or terminated the data would be deleted.
10. **AWS EMR(Elastic MapReduce):** is the tool could let you analyze and process big data by distributing the computational work across a cluster of virtual servers(EC2). The cluster is managed using an open-source framework called Hadoop.
11. **Hadoop:** Hadoop uses a distributed processing architecture called MapReduce in which a task is mapped to a set of servers for processing, visit here to know more: <http://hadoop.apache.org/>
12. **Cluster** is a set of virtual servers running on EC2 instances, i.e. a cluster of instances.
13. **Node:** we define 2 roles for the virtual servers(instances):a. master node: manage the cluster, do not run tasks; b. slave node: only run tasks.
14. **Amazon S3(Simple Storage Service)** is storage for the Internet. we can read in the data from S3 bucket to cluster by Spark directly, the data would be deployed as distributing data automatically on all of the slave nodes.

This is pretty much what’s in my mind now; let me know if you have any questions.

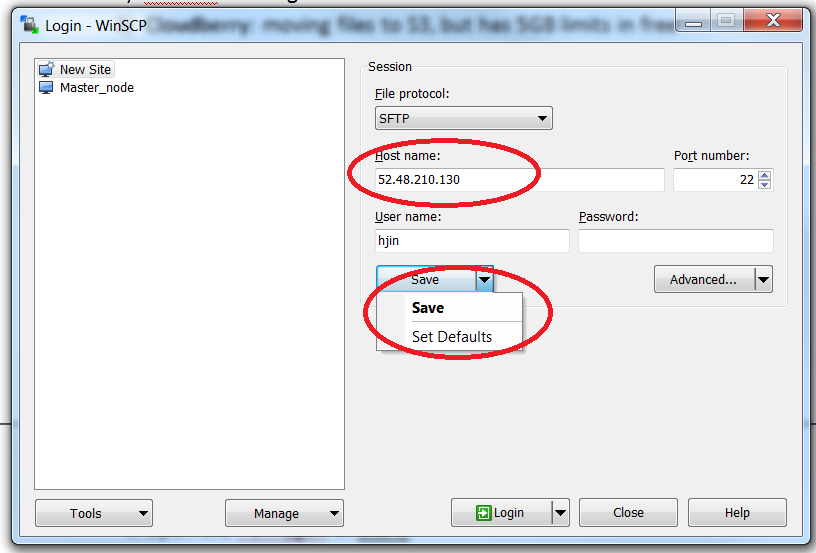
1. **Current environment**

* **Nodes**: 1 master node, 6 slave nodes
* **Instance Type**: m2.xlarge, each has 2 physical CPUs, 16 virtual cores, 17.1 GB memory(14GB available)
* **Master node public IP address**: 52.48.210.130
* **DNS**: ec2-52-48-210-130.eu-west-1.compute.amazonaws.com
* **S3 bucket**: s3://emr-rwes-pa-spark-dev-datastore
* **Storage on cluster**: around 500GB, it might be larger since we asked Dan to add 4 more instances.

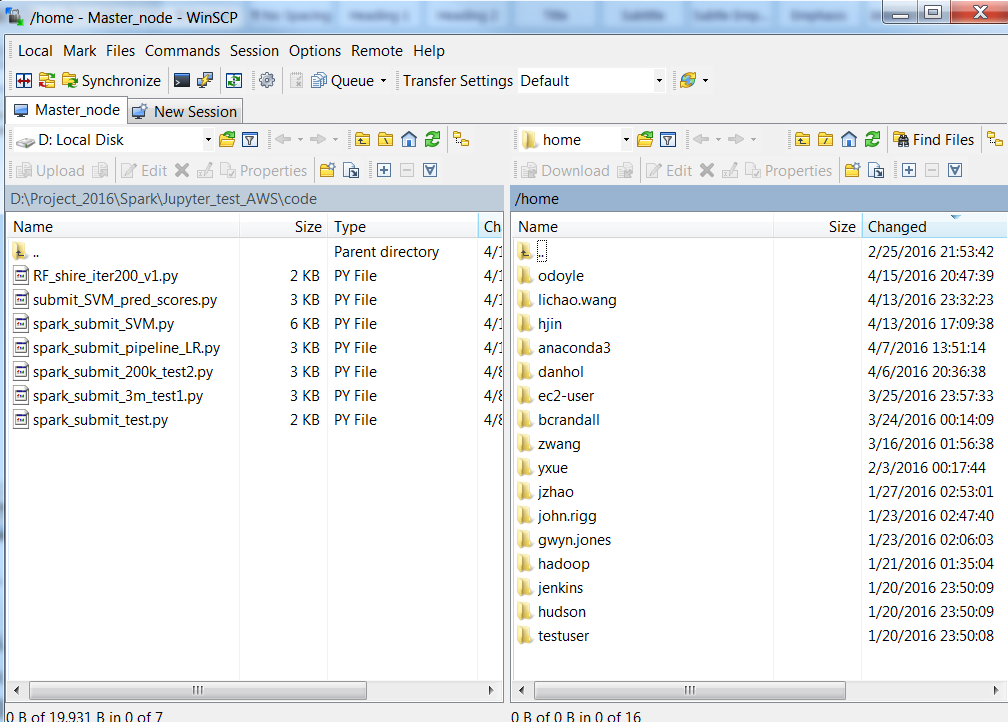
1. **Softwares we need**
2. **PuTTY**: SSH to master node
3. **WinSCP**: moving files to master node
4. **Cloudberry**: moving files to S3, but has 5GB limits in free version(size of one file).
5. **Anaconda3**: provides jupyter notebook in laptop
6. **Chrome**: web interface to connect to Jupyter server on master node, which is our IDE; we could monitor the cluster by chrome
7. **FoxyProxy**: extension on Chrome, provides a set of proxy management tools.
8. **foxyproxy-settings.xml**: proxy setting, don’t need to be installed

Most of the softwares have been installed on your laptop I suppose, if not and you have the authority to install software on your laptop, please go to *F:\Hui\download\Spark* on server 100, all of them are there except Chrome.

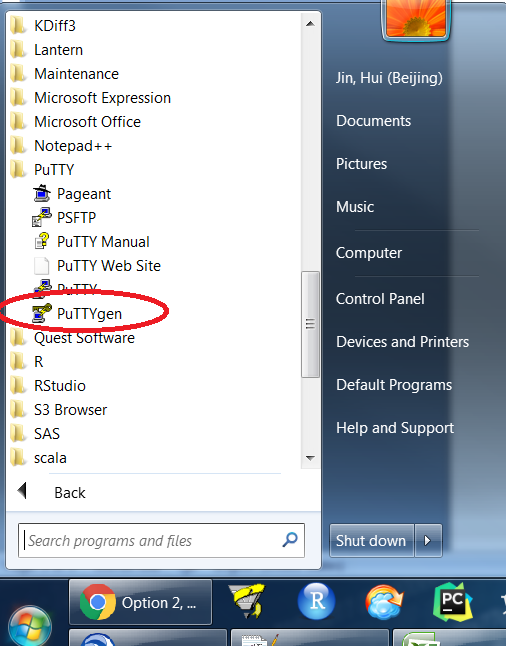
1. **Setting up on laptop(Presume all the softwares above are installed)**
2. **SFTP to master node**
3. Open the WinSCP on your laptop, enter the public IP address of master node and your credential in blank, then name the session and save it.



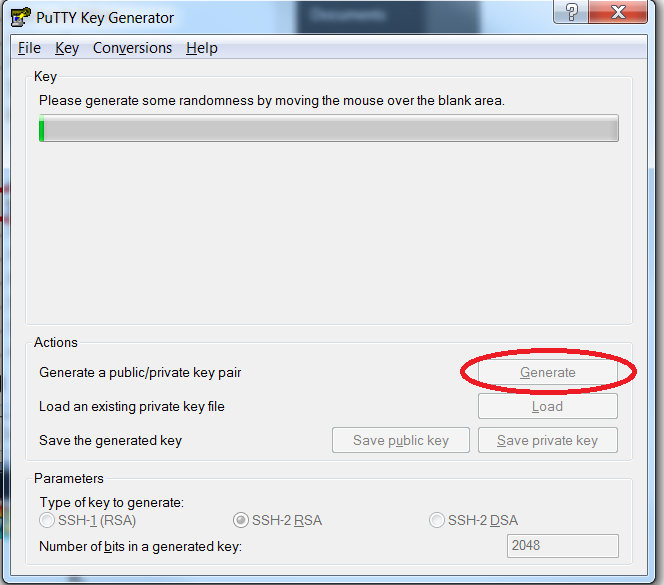
1. Click login, then we connect to the master node by SFTP. Left panel is our local folder, right panel is the master node. The WinSCP supports drag&drop between two windows, we can move files between laptop and master node now. Note you may only have the authority to write/read in your own directory.



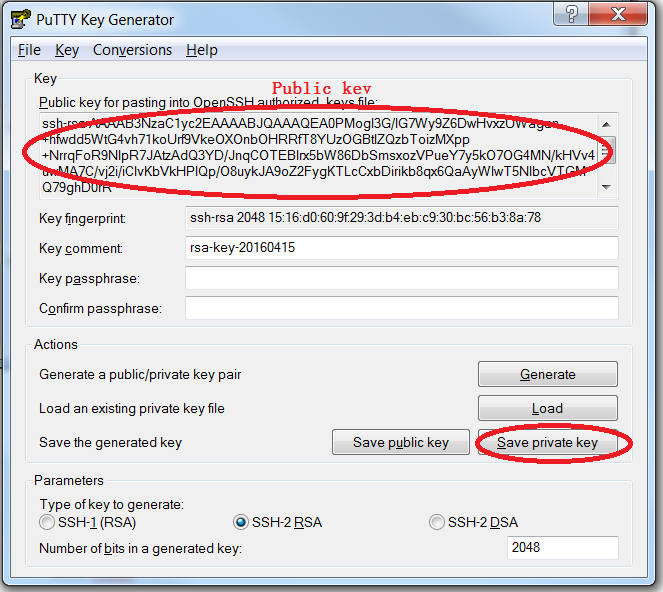
1. **SSH to master node:**
2. Open the PuTTYgen in *Start,*

****

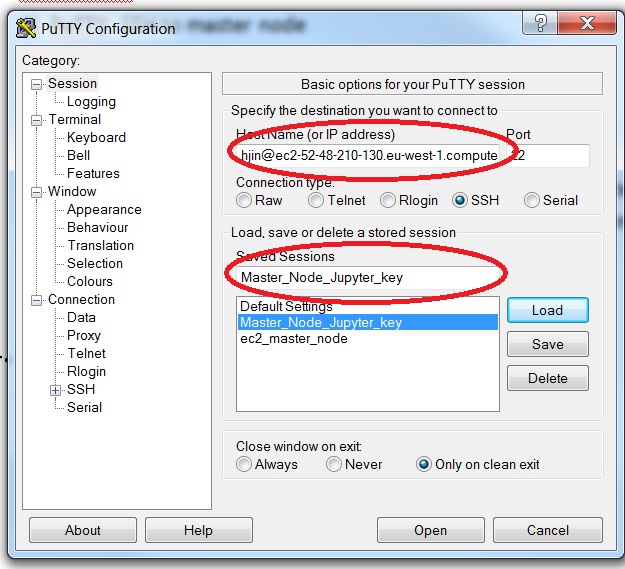
1. Click the generate button and moving the mouse in blank area to generate the key pairs

****

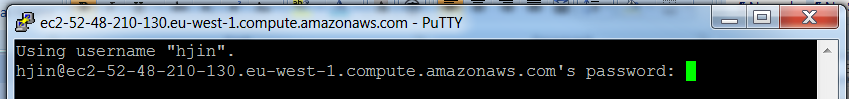
1. Copy the entire public key and save into a txt file, make sure the pub key is in one line and no breaks; then click on Save private key button to save the private key file in somewhere, the private key file will be in .ppk format. You may close the key generator now.



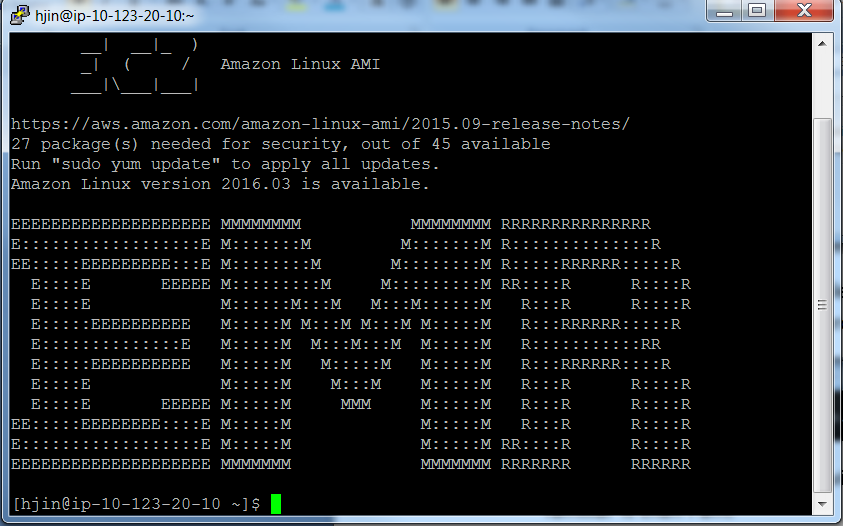
1. Open the PuTTY on your laptop, enter *“xxx@ ec2-52-48-210-130.eu-west-1.compute.amazonaws.com”* in the host name, xxx is the user name Dan sent to you. You also could enter the public IP address here. Then enter the name you prefer in Saved Sessions and click save.

****

1. Click on Open or double-clicking the session name you saved, then enter the password that Dan provided to you, hit enter on keyboard



1. The master node is in Linux environment, I suppose all of you are more familiar with it than I am.

****

1. Now let’s import our public key to the master node. Create ~/.ssh/authorized\_keys file under your directory by:

$ mkdir ~/.ssh #create .ssh directory

$ chmod 700 ~/.ssh #set the right permission

$ touch ~/.ssh/authorized\_keys #create authorized\_keys file

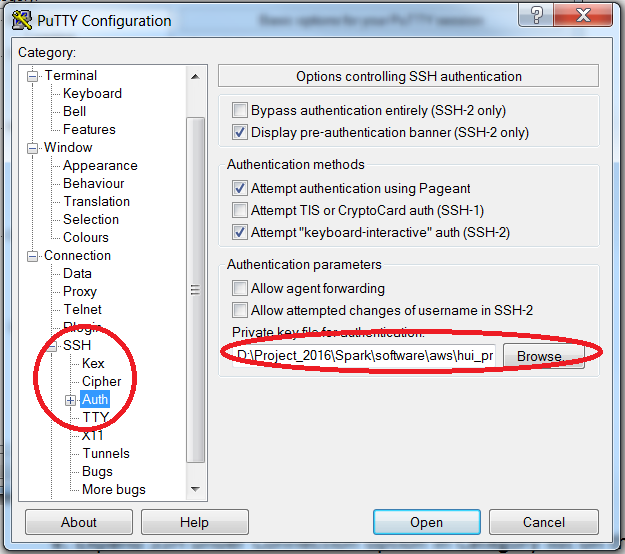
$ chmod 600 ~/.ssh/authorized\_keys #set the right permission

1. Upload the public key file(txt version) we created in step iii) to master node under your directory by WinSCP(drag&drop)
2. On master node under your directory, execute command line:

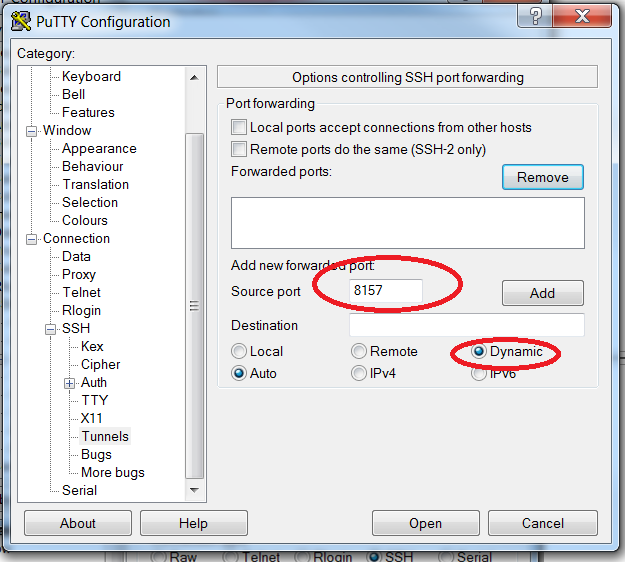
$ cat keys\_file >>/home/xxx/.ssh/authorized\_keys

xxx is your user name. The public key is imported now, you may close the PuTTY.

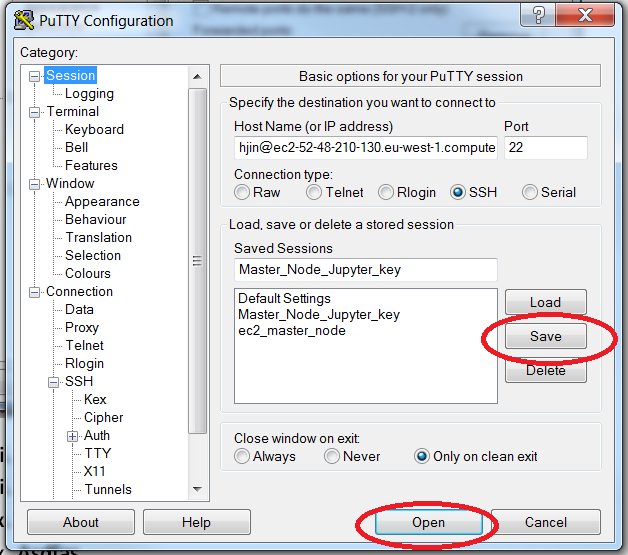
1. Reopen PuTTY, Load the session we saved above. Then expand SSH under Connection option in Category list on the left, choose Auth. Then choose Browse to select the .ppk file we just created in step iii.

****

1. Turn to Tunnels option in SSH, enter 8157 in Source port, leave Destination as blank and select Dynamic and Auto, then click Add, “D8157” will show up.



1. Go back to Session, click Save to save the configuration and click Open, you may SSH to the master node password-less-ly.

****

1. **Install Anaconda 3 on master node**
2. Copy */home/anaconda3/ Anaconda3-2.5.0-Linux-x86\_64.sh* to your directory by

$ cp /home/anaconda3/Anaconda3-2.5.0-Linux-x86\_64.sh /home/XXX/Anaconda3-2.5.0-Linux-x86\_64.sh

1. Then cd to the folder you save the installer and execute

~~$ bash ~/home/xxx/anaconda3/Anaconda3-2.5.0-Linux-x86\_64.sh~~

$ ./anaconda3/Anaconda3-2.5.0-Linux-x86\_64.sh

to install Anaconda 3. Note:

* 1. ~~Replace /xxx/anaconda3 with your actual path~~
  2. Accept the default location or select a user-writable install location
  3. ~~Installing Anaconda might need root privileges so add “~~*~~sudo~~*~~” in front of “~~*~~bash~~*~~” command, it would need your password~~
  4. Accept to add Anaconda directory to your bash shell PATH environment variable. If not you may add this line to the file .bashrc in your home directory

export PATH=”/home/xxx/anaconda3/bin:$PATH”

Replace the /home/xxx/anaconda3 with your actual path

1. Close and re-open PuTTY for changes to take effect.
2. **Add PySpark module to Python**
   * 1. SSH to master node by PuTTY
     2. Add the following lines to .bashrc:

#Amazon Keys (Substitute your credentials here)

export AWS\_ACCESS\_KEY\_ID='AKIAJ3HRKY4L3EXCTLGQ'

export AWS\_SECRET\_ACCESS\_KEY='C4fCFn9D8gOxOk69pIzIJNIR7W9kWKxy0LbLcQRz'

# Add extra PATH variables

export SPARK\_HOME=/usr/lib/spark/

export PATH=$SPARK\_HOME/bin:$PATH

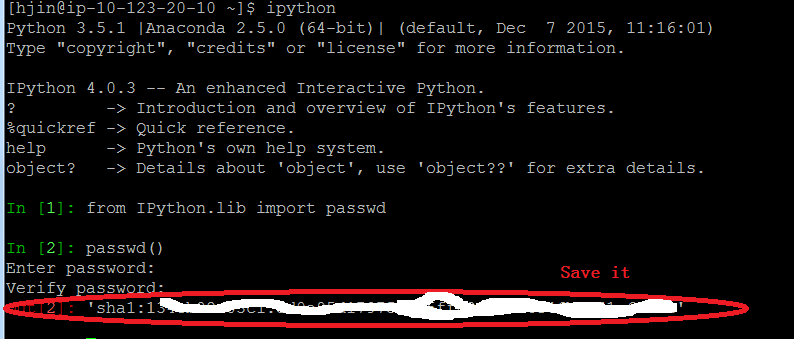
export PYTHONPATH=$SPARK\_HOME/python/:$PYTHONPATH

export PYTHONPATH=$SPARK\_HOME/python/lib/py4j-0.10.1-src.zip:$PYTHONPATH

export PATH=/home/hjin/anaconda3/bin:$PATH

* + 1. Close and re-open PuTTY for changes to take effect

1. **Launch Jupyter server on master node(Presume adding anaconda directory to your bash shell PATH environment)**
2. SSH to your master node by PuTTY
3. Open the IPython Terminal to get an encrypted password so as to use it for logging into our IPython Notebook Server. Remember to copy and save the output of the encrypted password, something like “sha1…”. Then exit out of IPython terminal using “*exit*” command

****

1. Create the configuration profile for Jupyter Notebook server by

$jupyter notebook -–generate-config

1. Create a self-signed certificate for accessing the Notebook through HTTPS under your own directory by:

$ mkdir certs

$ cd certs

$ sudo openssl req –x509 –nodes –days 365 –newkeys rsa:1024 –keyout mycert.pem –out mycert.pem

It will ask some questions, please answer them to the best of your knowledge as some of them are required to successfully create the certificate.

1. Change the configuration setting of Jupyter server:
2. cd to .jupyter folder and edit jupyter\_notebook\_config.py by:

$ cd ~/.jupyter

$ vi jupyter\_notebook\_config.py

1. add the following settings to the top of the file and leave the rest commented out as it is:

c = get\_config()

#Kernel config

c.IPKernelApp.pylab = ‘inline’ #if you want plotting support

#Notebook config

c.NotebookApp.certfile = u’/home/xxx/certs/mycert.pem’ #location of your certificate file

c.NotebookApp.ip = ‘\*’

c.NotebookApp.open\_browser = False #so the notebook does not opens up a browser by default

c.NotebookApp.password = u’sha1:….’ #the encrypted password we generated above

#put Jupyter notebook on a known, fixed port, like 8888

c.NotebookApp.port = 8888

1. Create a new folder which will store all the notebooks:

$ cd ~

$ mkdir Notebooks

$ cd Notebooks

1. Now launch the Jupyter server as no-hangup by

$ nohup jupyter notebook

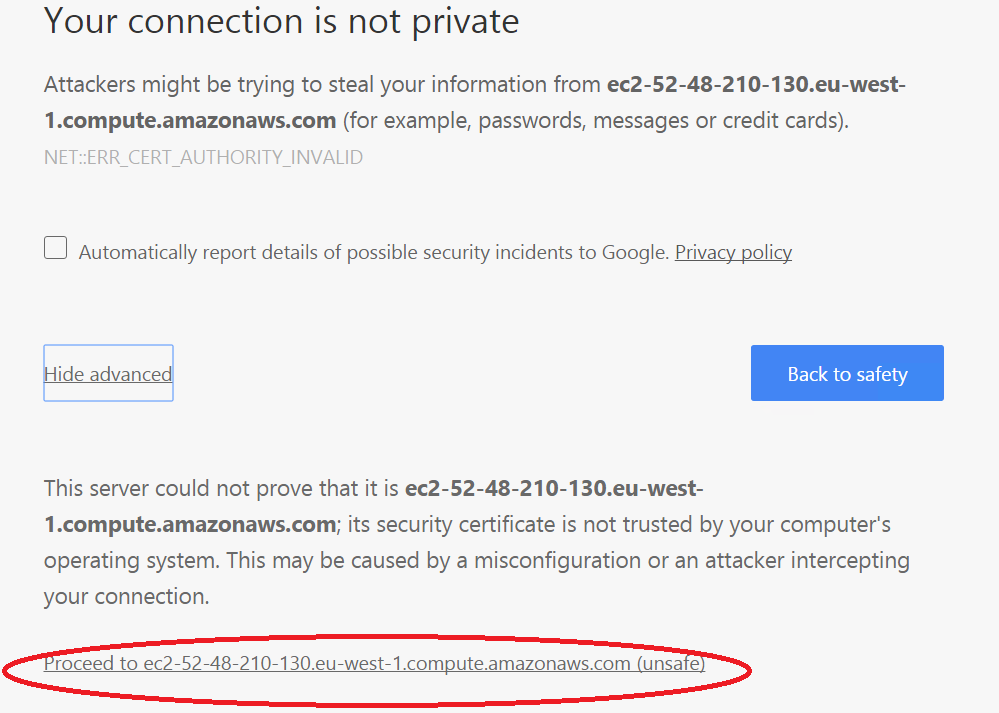
1. To stop the Jupyter server:
   1. List the PID of the process under Notebook folder by :

$ lsof nohup.out

* 1. Kill the Jupyter server by its PID:

$ kill -9 “PID”

1. **Config FoxyProxy for Chrome**
2. Open **Chrome -> more tools -> extensions**, then drag “Foxyproxy-3.0.7.1-Crx4Chrome.com.crx” into the window. Click on **Add**
3. On the **FoxyProxy standard page**, choose **Import/Export**
4. On the **Import/Export** page, choose **Choose File**, browse to the location of *foxyproxy.settings.xml*, select the file and choose **Open**
5. Choose **Replace** when prompted to overwrite the existing settings.
6. At the top of the page, for **Proxy mode**, choose **Use proxies based on** their predefined patterns and priorities
7. **View websites hosted on the master node**
8. Open the **Chrome -> tools -> extensions**, enable the *FoxyProxy Standard*. If you have other proxy manager in Chrome, please un-enable it.
9. URL for IDE and monitoring websites
   1. **Jupyter Notebook**: go to URL: [*https://ec2-52-213-180-156.eu-west-1.compute.amazonaws.com:8888*](https://ec2-52-213-180-156.eu-west-1.compute.amazonaws.com:8888)*,* be sure to use HTTPS and proceed past the security error which shows due to a self-signed certificate we created above. Then enter the password we setup in item 5), we are in



* 1. **Job-tracker**: [*http://ec2-52-213-180-156.eu-west-1.compute.amazonaws.com:8088*](http://ec2-52-213-180-156.eu-west-1.compute.amazonaws.com:8088)
  2. **Ganglia(monitor):** [*http://ec2-52-213-180-156.eu-west-1.compute.amazonaws.com/ganglia/*](http://ec2-52-213-180-156.eu-west-1.compute.amazonaws.com/ganglia/)

1. **AWS EC2 console(To monitor the cluster)**
2. Open Chrome under the regular proxy setting in IMS
3. Go to URL: *https://756982440859.signin.aws.amazon.com/console/*
4. Login EC2 console using the following credential:
   1. Account: 756982440859
   2. Login: spark-emr
   3. Pass: %parkEMR!!
5. **Connect to S3 (copy to S3, console)**
6. **Cloudberry(presume installed)**:
7. Make sure the IE proxy setting is correct according to your location:

* London:

Proxy script - <http://frboupac01.internal.imsglobal.com/proxy.pac>

Proxy server - frboucpxy01.internal.imsglobal.com

proxy Port – 8080

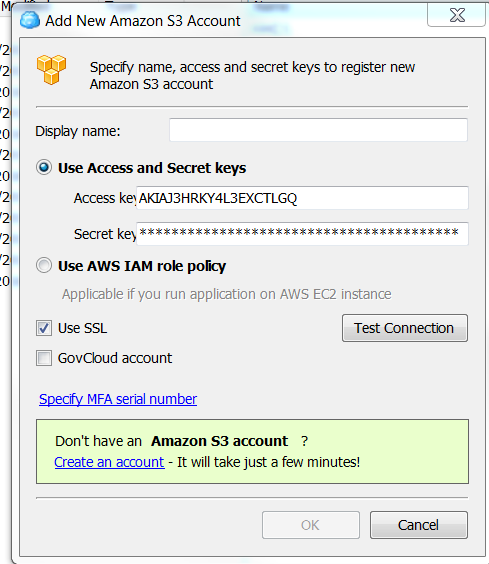
* Beijing:

Proxy script - <http://cnbejpac01.internal.imsglobal.com/proxy.pac>

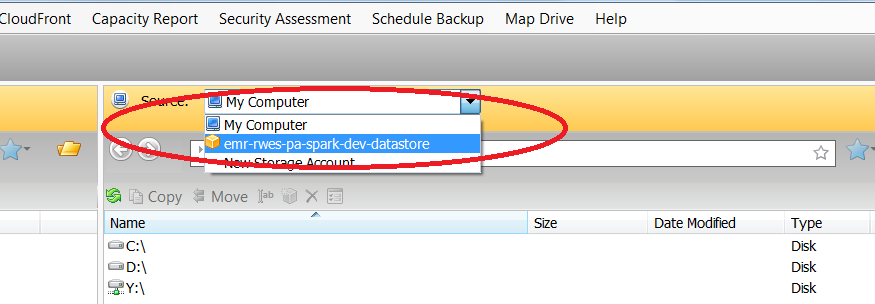
1. Open Cloudberry, in **file** option select new Amazon S3 account
2. In the prompted window, enter the Access key & Secret key as follow:

aws\_access\_key\_id = AKIAJ3HRKY4L3EXCTLGQ

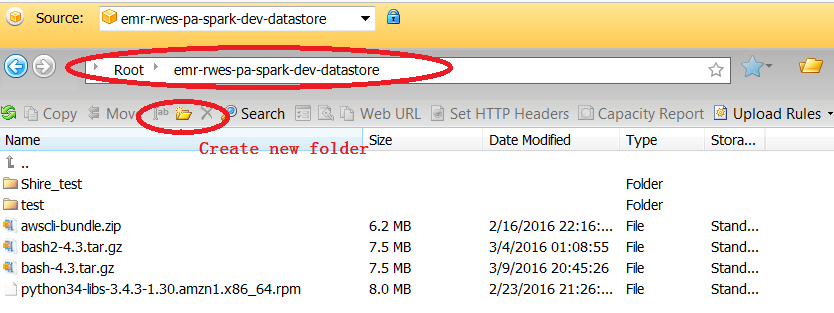
aws\_secret\_access\_key = C4fCFn9D8gOxOk69pIzIJNIR7W9kWKxy0LbLcQRz



1. Enter a display name as you like, then click OK.
2. We could connect to S3 on right panel by selecting Source as the displayed name you just saved



1. The PA bucket is *s3://emr-rwes-pa-spark-dev-datastore*, we could create new folder under the bucket.



1. Cloudberry also supports drag&drop, we could move data between laptop and S3. Note that the free version Cloudberry has a limit of size of uploading data, which is 5 GB. So if we want to move larger dataset, we have to turn to other methods.
2. **WinSCP & master node:**
3. Open WinSCP to connect to master node
4. Upload the file to master node. There is no limit in WinSCP, I tried with 11 GB data and it works fine.
5. SSH to master node by PuTTY, using the example below to move files to S3(assume we are moving awscli-bundle.zip):

* **From master node to S3:**

$ aws s3 cp awscli-bundle.zip s3://emr-rwes-pa-spark-dev-datastore/ awscli-bundle.zip

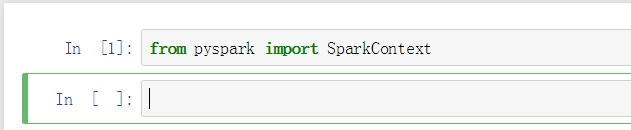
* **From S3 to master node:**

$ aws s3 cp s3://emr-rwes-pa-spark-dev-datastore/awscli-bundle.zip awscli-bundle.zip

1. **EC2 console:**
2. Log into EC2 console under instruction in item 8)
3. On the top left corner, click on Services drop list, choose S3
4. Select *emr-rwes-pa-spark-dev-datastore* on the left list
5. Choose Create Folder on the top left to create a folder
6. Choose Upload to upload a file from laptop.
7. **Submit PySpark in IDE(Jupyter Notebook, to test code, it runs on master node only)**
8. SSH to master node by PuTTY, open Chrome and enable the FoxyProxy.(un-enable other proxy manager if there is)
9. Go to <https://ec2-52-213-180-156.eu-west-1.compute.amazonaws.com:8888> to open Jupyter Notebook
10. Create a new notebook by choose New -> Python 3



1. From pyspark module import SparkContext:



1. It’s implied our pyspark module is imported as we expect, we could continue our job.
2. Please find a template notebook in */home/anaconda3/test\_spark\_template\_hui.ipynb*
3. **Spark-submit command(write a Spark application, it runs on all of the slave nodes)**
4. Spark application template:

## Imports

from pyspark import SparkConf, SparkContext

## CONSTANTS

APP\_NAME = "My Spark Application"

##OTHER FUNCTIONS/CLASSES

## Main functionality

def main(sc):

rdd = sc.parallelize(range(1000), 10)

print rdd.mean()

if \_\_name\_\_ == "\_\_main\_\_":

# Configure OPTIONS

conf = SparkConf().setAppName(APP\_NAME)

sc   = SparkContext(conf=conf)

# Execute Main functionality

main(sc)

sc.stop()

1. Write an application following the template above.
2. Upload the data to S3 bucket and read it from S3 directly as the S3 would deploy the data to all the slave nodes automatically. Otherwise we have to duplicate the data to slave nodes in identical location by hand.
3. Upload the application to master node
4. Execuate the spark-submit as:

$ sudo spark-submit

--deploy-mode client #using cluster mode

--master yarn #connect to yarn

--num-executors 17 #number of executors would use

--executor-cores 5 #number of cores would use

--executor-memory 5g #space of memory would use

--conf spark.yarn.submit.waitAppCompletion=false #run in background

/home/hjin/shire\_test/spark\_submit\_test.py #script location

[] #other arguments

Note that the num-executors, executor-cores and executor-memory are not determined. They are the critical parameters for tuning Spark.

1. Please find a template code in */home/anaconda3/spark\_submit\_template\_hui.py,* you could submit it by

$ sudo spark-submit /home/anaconda3/spark\_submit\_template\_hui.py

1. Using Zeppelin to submit job
2. Install Jupyterhub for multi-user
3. **Useful website**

* <http://spark.apache.org/>
* <http://spark.apache.org/docs/latest/>
* <http://spark.apache.org/docs/latest/mllib-guide.html>
* <http://spark.apache.org/docs/latest/sql-programming-guide.html#spark-sql-datatype-reference>
* <http://docs.aws.amazon.com/ElasticMapReduce/latest/ManagementGuide/emr-what-is-emr.html>
* <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>
* <http://hadoop.apache.org/>