OMNI COMPREHENSIVE MAPD-B CLUSTER CREATION, SPARK INSTALLATION AND S3 ACTIVATION

This document has been created by Veronica Bedin and Andrea Marchetti, primarily as a step-by-step guide for us to remember all the important steps that led us to make the cluster work, in case we ever need to modify something. It is not intended to substitute the official CloudVeneto documentation or the guides provided by Professor Pazzini and Ph.D. student Matteo Migliorini, from which we have borrowed various parts.

Towards the end, we realized that it could be helpful to collect all the steps that were previously scattered across the internet, and that it could be beneficial to other students. We are sharing this with you in the hopes that it will make things easier for students, but we do not guarantee the accuracy of all the information. It represents what enabled our project to start working.

If you encounter any issues, or know of any upgrade that could be done, please comment on the file, and we will update it as soon as we find the time. Additionally, please be patient and try different solutions before contacting us. We are not experts, and it required a significant amount of debugging and troubleshooting for certain aspects to function properly. Take your time to attempt to resolve the problems (with the assistance of the almighty ChatGPT).

Apart from this, ⭐**GOOD LUCK FOR YOUR PROJECT**⭐.

⚠️ ATTENTION ⚠️ It seems that you can work with just on VM on the project, all the other ones (that didn't create the file) will have read-only permission to the jupyter notebook. Consider taking just one user and use that for the whole project.

Follow the notes in "**Notes on how to set up and use a CloudVeneto Virtual Machine"** file from moodle until "Creating a new VM". This (Creating a new VM) and the next steps have already been done by the professor. You already have your VM, with their name, IP address and the account information to access them.

⚠️ ATTENTION ⚠️ In all the previous step, you have to use as username and password the ones that were sent to you by cloudveneto when you registered (referred to as `username` and `pwd` from now on). If you registered a long time ago and didn't change the ssh password (error message "expired account" when trying to access), you'll need to write to support@cloudveneto.it to obtain new credentials.

⚠️ ATTENTION ⚠️ The credentials for the machines (sent by Pazzini in the mail when he creates the VM) will be referred to from now on as `VM_username` and `VM_pwd`.

⚠️ ATTENTION ⚠️ In windows the setting of the .pem file with the chmod 600 doesn't seem to work, no solutions have been found to this day.

🚩 CHECK 🚩 If everything works, you should be able to connect to your machine in interactive mode with these steps:

1. Connect to the `cloudveneto.gate.it` server using your CloudVeneto username-password

   ```
   $ ssh username@gate.cloudveneto.it
   ```

2. Now you find yourself inside the network where your VM is “located”. Connect to your VM using its IP address and the key you have placed in the `gate.cloudveneto` server.

   ```
   $ ssh -i private/yourpemfile.pem
   VM_username@YOUR_VM_IP_ADDRESS
   ```

3. You should now find yourself inside the home of the VM. From there, you can use it as a normal machine, e.g. follow the Docker installation guide, pull repositories from GIT, or connect to running containers.

NOW I’M FOLLOWING THIS OTHER GUIDE
The followed steps are the same, here there are some added considerations/tips

⚠️ ATTENTION ⚠️ Here I'm using as a text editor nano, not vim as proposed by the "old" guide proposed by the professor
Once you've finished to modify the document, press CTRL+X to close it , Y to confirm the modification and then ENTER to not modify the file name (they need to remain the same)
If you prefer vim, feel free to modify all the commands

Open the connection to cloudveneto with

```
$ ssh username@gate.cloudveneto.it
```

⚠️ ATTENTION ⚠️

- When I say ALL THE VM I mean: master, slave01 and slave 02
- When I say ONE USER I mean that the changes will be applied to all the different VM of al the components of the group
- When I say ALL USERS, these actions must be done to every different VM of all the components of the group (but if you chose to use the VM from just one user, don’t consider this)
- When you need to access a specific VM, use
  ```
  $ ssh -i ~/private/yourpemfile.pem
  VM_username@YOUR_VM_IP_ADDRESS
  ```
  And after that the VM_pwd

ALL THE VM, ONE USER

Modify the hosts file with

```
$ sudo nano /etc/hosts
```

And modify the file adding the Ip_address of your VM followed by their name, like this

```
  GNU nano 6.2                                        /etc/hosts
127.0.0.1 localhost

10.67.22.35 master
10.67.22.82 slave01
10.67.22.133 slave02

# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
```

and add your VM names and IP address, then save it

Apply the changes to all the machines

```
$ sudo reboot
```

⚠️ ATTENTION ⚠️
After this a little bit of time will be needed before regaining access to the VM (~5-10 s)

ALL THE VM, ONE USER

```
$ sudo apt-get install software_properties_common
$ sudo add-apt-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install openjdk-11-jdk
```

🚩 CHECK 🚩 if Java is installed

```
$ java -version
```

```
$ sudo apt-get install scala
```

🚩 CHECK 🚩 if Scala is installed

```
$ scala -version
```

ALL USERS (The next instructions seems that needs to be done in all the VM of all the users of the group, since the keys are saved in home/user_name/.ssh and if you try to access to them via another VM that didn't do these steps you won't be able to see anything)

IN MASTER:

```
$ sudo apt-get install openssh-server openssh-client
$ ssh-keygen -t rsa -P ""
```

After this an interactive line will appear and you just need to press Enter

Now make this key an authorized key

```
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

Now copy the content of the file in all the machines

```
$ ssh-copy-id VM_username@master
$ ssh-copy-id VM_username@slave01
$ ssh-copy-id VM_username@slave02
```

IN MASTER

🚩 CHECK 🚩 the connection to the slaves

```
$ ssh slave01
```

Warnings will appear, write `yes` and `ENTER` (only first time, from now on you are connected)
ATTENTION! Now it will enter in slave01, you need to run
`$ exit`

and only then run

```
$ ssh slave02
```

Warnings will appear, write `yes` and ENTER

```
$exit
```

Now you are back at MASTER node

From now on, calling `$ssh slave01` or `$ssh slave02` will let you enter the slaves, with `$exit` you will be back at the MASTER

IN MASTER AND SLAVES, ONE USER

Now download Spark (newest version)

```
$ wget
https://dlcdn.apache.org/spark/spark-3.4.0/spark-3.4.0-bin-hadoop3.t
gz
```

```
$ tar xvf spark-3.4.0-bin-hadoop3.tgz
```

Use the following command to move the spark software files to respective directory (*/usr/local/bin*)

```
$ sudo mv spark-3.4.0-bin-hadoop3 /usr/local/spark
```

IN MASTER AND SLAVES, ALL USERS

```
$ sudo nano ~/.bashrc
```

Add this line of text at the end of the file
⚠️ ATTENTION ⚠️ No spaces, it made my .bashrc file crash and nothing worked anymore

```
export PATH=$PATH:/usr/local/spark/bin
```

Then update the /.bashrc file with

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

IN MASTER

Move to spark *conf* folder and create a copy of the template of *spark-env.sh* and rename it.

```
$ cd /usr/local/spark/conf
$ cp spark-env.sh.template spark-env.sh
```

Edit configuration file

```
$ sudo nano spark-env.sh
```

adding these commands

```
export SPARK_MASTER_HOST='<MASTER-IP>'export
JAVA_HOME=<Path_of_JAVA_installation>
```

Edit the slaves file in *`/usr/local/spark/conf`*

```
$ sudo nano slaves
```

⚠️ ATTENTION ⚠️ In the internet guides, you need to add both master and slave01, slave02. We didn't do it because from the documentation it seems that here there should be present only the slaves, not the master.

Add the lines

```
slave01
slave02
```

Let's try to start our Apache Spark Cluster, hopefully everything is ok!

```
$ cd /usr/local/spark
$ ./sbin/start-all.sh
```

Something like this should appear

```
bedin@mapd-b-2023-15-3:~$ cd /usr/local/spark
bedin@mapd-b-2023-15-3:/usr/local/spark$ ./sbin/start-all.sh
org.apache.spark.deploy.master.Master running as process 1464.  Stop it first.
slave02: org.apache.spark.deploy.worker.Worker running as process 2137.  Stop it first.
slave01: org.apache.spark.deploy.worker.Worker running as process 2651.  Stop it first.
```

To check if the services started we use the command:

```
$ jps
```

In Master, you should see two instances, jps and master, followed by a number. Logging into slaves and running jps again you should see jps and worker, again followed by a number. (This applies to our choice of not putting master in the `slaves` file)

ALL VM

Now you'll need to install

- python3
- pip
- jupyter notebook

Here things were a little confused, check if you already have this programs and if you don't install them, for jupyter notebook the installation with pip is sufficient

To connect Pyspark with Jupyter notebook, modify the ./bashrc file

```
$ sudo nano ~/.bashrc
```

adding

```
export PYSPARK_PYTHON=/usr/bin/python3
export PYSPARK_DRIVER_PYTHON='jupyter'
export PYSPARK_DRIVER_PYTHON_OPTS='notebook --no-browser
--port=8889'
```

Where 8889 is the gate from the VM that you will connect to your computer door
⚠️ Remember to use `$ source ~/.bashrc` to update the file
It could be useful to also run `$ sudo reboot` ⚠️

⚠️ TO CONNECT WITH YOUR LOCALHOST, YOU'LL NEED TO CONNECT THE GATES OF THE VM TO THE ONES OF YOUR COMPUTER ⚠️

I did this running in one new terminal the three different connection to the three different ports needed for the project (the number of the ports are the default one):

- jupyter notebook (pyspark really) **8889** (or the number you set before)
- Spark UI **8080**
- Application UI **4040**

Before doing this, start jupyter notebook with pyspark (next chapter)

TO START THE JUPYTER NOTEBOOK

This is what you'll need to do every time you start to work on the project

(or at least this is what made our program work, more intelligent solutions could be applied)

```
$ ssh user_name@gate.cloudveneto.it
```

```
$ ssh -i ~/private/PoD_projects.pem VM_username@MASTER_IP
```

```
$ cd /usr/local/spark
```

```
$ ./sbin/start-all.sh
```

```
bedin@mapd-b-2023-15-3:/usr/local/spark$ ./sbin/start-all.sh
org.apache.spark.deploy.master.Master running as process 8156.  Stop it first.
master: org.apache.spark.deploy.worker.Worker running as process 8311.  Stop it
first.
slave02: org.apache.spark.deploy.worker.Worker running as process 3802.  Stop it
 first.
slave01: org.apache.spark.deploy.worker.Worker running as process 3524.  Stop it
 first.
```

Something like this should appear

⚠️ It's important to add the specification of the master, otw pyspark will start to run with just the master as worker and 0 workers ⚠️

⚠️ Open this when you are in `/usr/local/spark` ⚠️

```
$ pyspark --master spark://MASTER_IP:7077
```

IN ANOTHER TERMINAL, open (you can change the port values on the left to the ones you prefer)

```
ssh -J username@gate.cloudveneto.it -L 8080:localhost:8889 -L
1234:localhost:8080 -L 4321:localhost:4040 VM_username@VM_IP_ADDRESS
```

ON THE INTERNET (three times, one for each port)

```
localhost:4321/
localhost:8080/
localhost:1234/
```

WHEN YOU FINISH, REMEMBER
`$ ./sbin/stop-all.sh`

s3 CONNECTION PYSPARK

To find your credentials, in the Dashboard (cloudveneto) go to **Project** → **API Access** and then click on **View Credentials**: the relevant attributes are referred to as "EC2 Access Key" and "EC2 secret Key".

IN ALL THE VM

Download the cloudveneto certificate with

```
$ wget
https://repo1.maven.org/maven2/org/apache/hadoop/hadoop-aws/3.2.0/ha
doop-aws-3.2.0.jar \
```

```
-P $SPARK_HOME/jars/
$ wget
https://repo1.maven.org/maven2/com/amazonaws/aws-java-sdk-bundle/1.1
1.375/aws-java-sdk-bundle-1.11.375.jar \
-P $SPARK_HOME/jars/
```

If it has problem finding the `$SPARK_HOME` variable, set it in the `.bashrc` file with the path to spark (usually `/usr/local/spark`) or move into the folder `/usr/local/spark/jars` and download this files here removing `-P $SPARK_HOME/jars/` from the command

Import the `jar` files to instruct Spark to handle connections via the S3 interface

```
$ wget
https://raw.githubusercontent.com/CloudVeneto/CertCA/master/CloudVen
etoCAs.pem
$ keytool -trustcacerts \
-keystore /usr/lib/jvm/java-11-openjdk-amd64/lib/security/cacerts \
-storepass <changeit> \
-alias CloudVeneto \
-import -file CloudVenetoCAs.pem
```

⚠️ ATTENTION ⚠️ You may need to run this command as `sudo`, because you need to be a root user to access the `cacerts` file

The storepass can be set to whatever you want, only restriction is that is must be longer than 6 digits, also “changeit” is ok as a storepass

Create a `$HOME/.s3cfg` file and fill it with this (create it with `sudo nano .s3cfg`)

```
host_base = rgw-cloud.pd.infn.it:443
host_bucket = rgw-cloud.pd.infn.it:443
use_https = true
ca_certs_file = /etc/grid-security/certificates/CloudVenetoCAs.pem
access_key = <your access key>
secret_key = <your secret key>
```

You need to fill it with the access_key `=` EC2 Access Key and secret_key = EC2 secret Key (that you got before).
The path to `ca_certs_file` can be whatever path brings you to the `CloudVenetoCAs.pem file.` With these steps it’s installed in `$HOME`

Now Start the Spark master and workers normally.

When creating a Spark Session, load the proper java modules

```
from pyspark.sql import SparkSession
spark = SparkSession.builder \
.master("spark://localhost:7077")\
.appName("Your Spark Application Name")\
.config('spark.jars.packages',
'org.apache.hadoop:hadoop-common:3.2.0')\
.config('spark.jars.packages',
'org.apache.hadoop:hadoop-aws:3.2.0')\
.config('spark.jars.packages',
'com.amazonaws:aws-java-sdk:1.11.375')\
.config("spark.executor.memory", "512m")\
.config("spark.sql.execution.arrow.pyspark.enabled", "true")\
.config("spark.sql.execution.arrow.pyspark.fallback.enabled",
"false")\
.config('spark.hadoop.fs.s3a.aws.credentials.provider',
'org.apache.hadoop.fs.s3a.SimpleAWSCredentialsProvider')\
.config('spark.hadoop.fs.s3a.access.key',
'79f100fb66c34d338833171110cb12fe')\
.config('spark.hadoop.fs.s3a.secret.key',
'--------------------------------')\
.config('spark.hadoop.fs.s3a.endpoint',
'https://cloud-areapd.pd.infn.it:5210')\
.config("spark.hadoop.fs.s3a.impl",
"org.apache.hadoop.fs.s3a.S3AFileSystem") \
.config("spark.hadoop.fs.s3a.metadatastore.impl",
"org.apache.hadoop.fs.s3a.s3guard.NullMetadataStore") \
.config("spark.hadoop.fs.s3a.path.style.access", "true") \
.config("spark.hadoop.fs.s3a.connection.ssl.enabled","false") \
.config("com.amazonaws.sdk.disableCertChecking","true") \
.getOrCreate()
# create and print the content of the data rdd
rdd =
spark.sparkContext.textFile('s3a://mapd-minidt-stream/data_*.txt')
rdd.take(3)
```

```python
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .master("spark://localhost:7077")\
    .appName("Your Spark Application Name")\
    .config('spark.jars.packages', 'org.apache.hadoop:hadoop-common:3.2.0')\
    .config('spark.jars.packages', 'org.apache.hadoop:hadoop-aws:3.2.0')\
    .config('spark.jars.packages', 'com.amazonaws:aws-java-sdk:1.11.375')\
    .config("spark.executor.memory", "512m")\
    .config("spark.sql.execution.arrow.pyspark.enabled", "true")\
    .config("spark.sql.execution.arrow.pyspark.fallback.enabled", "false")\
    .config('spark.hadoop.fs.s3a.aws.credentials.provider', 'org.apache.hadoop.fs.s3a.SimpleAWSCredentialsProvider')\
    .config('spark.hadoop.fs.s3a.access.key', '79f100fb66c34d338833171110cb12fe')\
    .config('spark.hadoop.fs.s3a.secret.key', '-------------------------------')\
    .config('spark.hadoop.fs.s3a.endpoint', 'https://cloud-areapd.pd.infn.it:5210')\
    .config("spark.hadoop.fs.s3a.impl", "org.apache.hadoop.fs.s3a.S3AFileSystem") \
    .config("spark.hadoop.fs.s3a.metadatastore.impl", "org.apache.hadoop.fs.s3a.s3guard.NullMetadataStore") \
    .config("spark.hadoop.fs.s3a.path.style.access", "true") \
    .config("spark.hadoop.fs.s3a.connection.ssl.enabled","false") \
    .config("com.amazonaws.sdk.disableCertChecking","true") \
    .getOrCreate()
# create and print the content of the data rdd
rdd = spark.sparkContext.textFile('s3a://mapd-minidt-stream/data_*.txt')
rdd.take(3)
```

PACKAGE INSTALL

⚠️ ATTENTION ⚠️ When you are installing the packages for your project, even pip, you need to do it in every VM