==> **AWS Setup**

- setup the virtual environment(Optional)

- Install aws-cli as follow:

    sudo pip install awscli

- Get the access_key_id and secret_access_key from AWS Console and setup in the system using following command:

    aws configure

- Create a security group and authorize it as follows:

    aws ec2 create-security-group --group-name sand_security --description "Security Group"

    aws ec2 authorize-security-group-ingress --group-name sand_security --protocol tcp --port 22 --cidr 0.0.0.0/0

- Create a key pair and save in pem file also provide the appropriate permissions to it as follows:

    aws ec2 create-key-pair --key-name sand_key --query 'KeyMaterial' --output text > sand_security.pem

    chmod 400 sand_security.pem

==> **Docker Setup**

- Install docker by following this link - https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-18-04

Question 1.1: Use free-tier images (e.g., ami-cd0f5cb6) if you are using the "free tier" resource. Remember that for free tier instances, you also need to use "--instance-type t2.micro" and specify the subnet id with "--subnet-id". Finally login the instance with ssh. Save your screen shots to show that you have successfully done.

Answer 1.1:

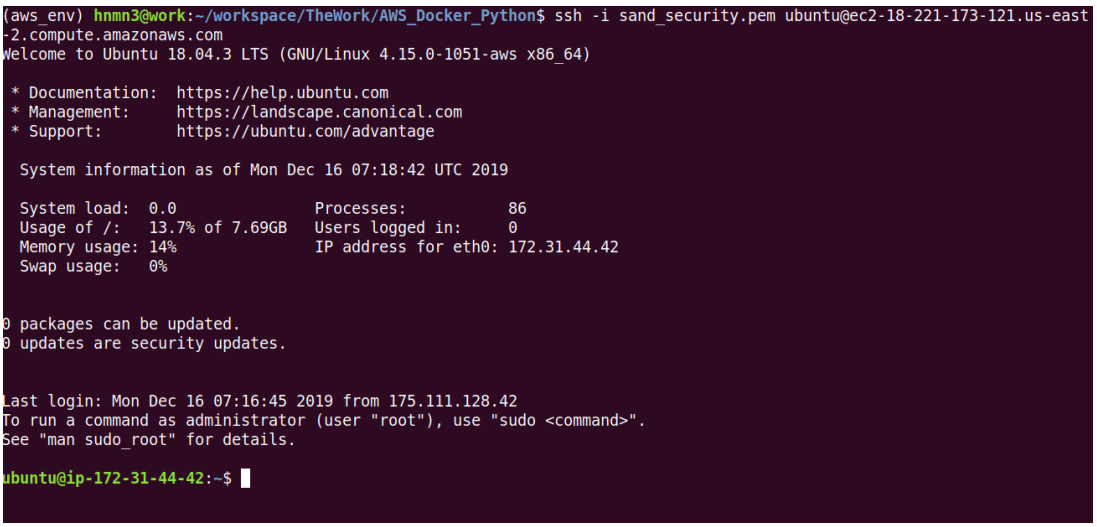
- start AWS EC2 Instance via command promt as below:

aws ec2 run-instances
--image-id ami-0d5d9d301c853a04a
--key-name sand_key
–security-group-ids sg-06b586adaf0fe5caf
--instance-type t2.micro
--subnet-id subnet-16bb3b5a

- In the above command ubuntu 18.04 image is choosen, including the image other details are fetched from aws console.
- Below are the screenshot of successful creation of aws ec2 instance.

- SSH to the instance:

ssh -i sand_security.pem ubuntu@ec2-18-221-173-121.us-east-2.compute.amazonaws.com



Question 1.2: Use the boto APIs to implement a python function start_instances(num_instances), where the parameter num_instances is the number of instances you will be creating. This function will create a number of instances and wait until the state of the instances become "running", and then return the list of instance ids.

Answer 1.2:

script with name answer_1_2.py is added.

Question 1.3 : Write a python script that uses the boto APIs to find out all the files in the bucket "wsu2017fall", print out the contents in the files, and copy the files to your own bucket. Remember to handle exceptions (e.g., empty directories). Keep your bucket undeleted until we finish grading!

Answer 1.3:

script with name answer_1_3.py is added. Just update the source_bucket_name and destination bucket name accordingly.

Question 1.4: Create a Docker image based on ubuntu image. Let's assume a scenario that you can remotely login a running container and debug a pyspark script. Therefore, the image should contain a ssh server, the single-node Spark setup, and a simple pyspark script (e.g., the wordcount program). (1) Post your dockerfile and your Docker hub link. (2) Post the command that starts the container and exposes its ssh port to external via the host's 2222 port (hint: check the -p option). (3) Post the screenshot showing that you can remotely login the container in an AWS instance and test-run it, e.g., "spark-submit wordcount.py" successfully.

Answer 1.4:

- Docker image link: https://hub.docker.com/r/hnmn3/sandy_ubuntu_pyspark

- Container Start command:

    sudo docker pull hnmn3/sandy_ubuntu_pyspark

    sudo docker run --name sandy_ubuntu_pyspark -p 2223:22 -itd hnmn3/sandy_ubuntu_pyspark

    sudo docker exec -it sandy_ubuntu_pyspark bash

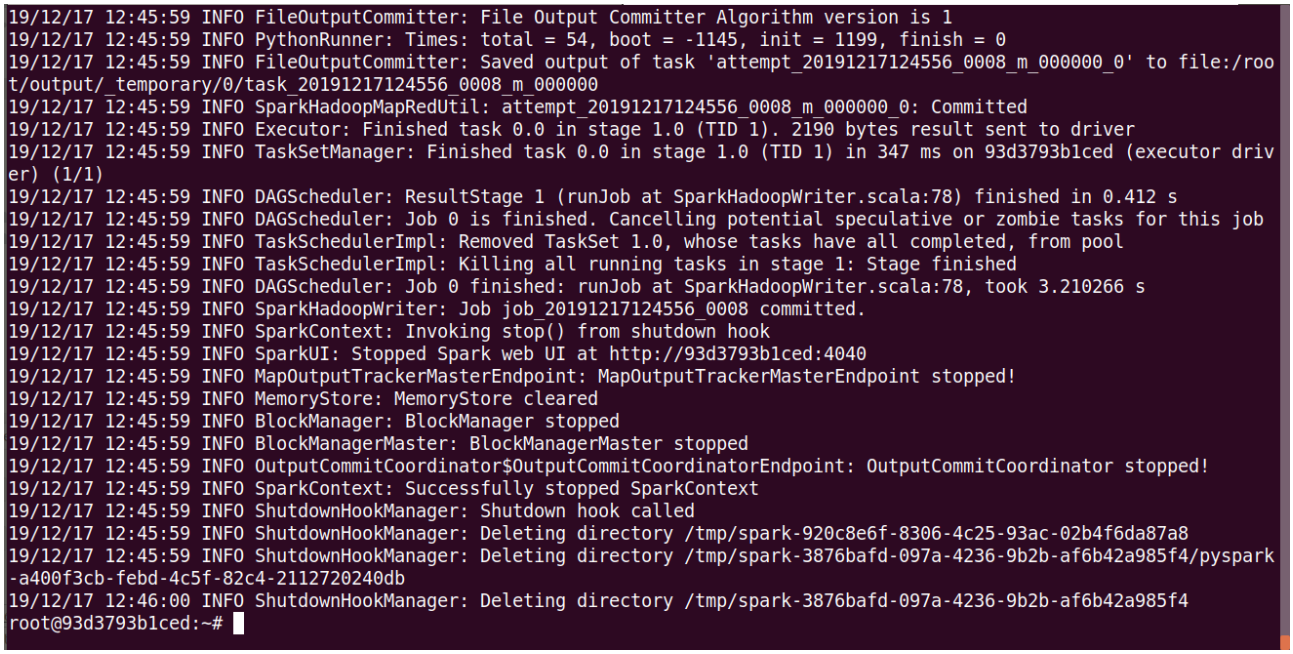
- Running docker on aws ec2 instance:

- ssh to docker container remotely: (Password is root)

- run the word_count program:

spark-submit word_count.py

- sample output screenshot:



Question 2.1 In this task, you will implement a tool with Python Boto3 library and the Paramiko Python SSH library to monitor the status of the instances you created. This monitoring tool will constantly (e.g., every 5 seconds) print out the CPU usage of each instance. Note that you can execute commands in instances remotely via ssh, like

```
ssh -i your_private_key.pem ubuntu@EC2_instance_Public_DNS "top -bn1 | grep Cpu"
```

The command "top -bn1 | grep Cpu" will get the the line of the command "top" output that contains Cpu information. The output of the remote command execution will be sent to you.

In your python code, you will need to create 2 instances using the function created in Q1.2 and then in a loop every 5 seconds the command is executed remotely in the instances by using the ssh functions provided by the Paramiko library, and print out the information "instance_ id \t Cpu usage".

Answer 2.1:

Script with name answer_2_1.py is provided.

Question 2.2 Extend your tool to monitor Docker containers in VM instances. Assume you have started 2 EC2 instances using the python function you developed. It's better to use an image that contains Docker. If not, for each EC2 instance, you can install Docker manually or via ssh commands in your python script. Then, use ssh command in python to start 2 Docker container daemons as follows (e.g., using the ubuntu image). Note that the -d option is used to run the container as a daemon.

```
docker run -d -t ubuntu sh
```

You can retrieve the container IDs (similar to VM instance IDs) using the following or other similar command.

```
docker ps | grep ubuntu
```

To execute a command in the container, for instance, getting the CPU usage, you can use

```
docker exec container_ID top -bn1 | grep CPU
```

Now you implement your python program to monitor the CPU usage of each container in each instance every 5 seconds and print out "instance_ID \t container_ID \t CPU usage".

Answer 2.2:

Install docker in ec2 instance manually and create some docker containers and update the ec2 instance ids in the program answer_2_2.py