

==> **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 prompt 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.

```
(aws_env) hnm3@work:~/workspace/TheWork/AWS_Docker_Python$ aws ec2 run-instances --image-id ami-0d5d9d301c853a04a --k
ey-name sand_key --security-group-ids sg-06b586adaf0fe5caf --instance-type t2.micro --subnet-id subnet-16bb3b5a
{
  "Groups": [],
  "Instances": [
    {
      "AmiLaunchIndex": 0,
      "ImageId": "ami-0d5d9d301c853a04a",
      "InstanceId": "i-0c13d7a201a45b8e6",
      "InstanceType": "t2.micro",
      "KeyName": "sand_key",
      "LaunchTime": "2019-12-16T06:55:47.000Z",
      "Monitoring": {
        "State": "disabled"
      },
      "Placement": {
        "AvailabilityZone": "us-east-2c",
        "GroupName": "",
        "Tenancy": "default"
      },
      "PrivateDnsName": "ip-172-31-44-42.us-east-2.compute.internal",
      "PrivateIpAddress": "172.31.44.42",
      "ProductCodes": [],
      "PublicDnsName": "",
      "State": {
        "Code": 0,
        "Name": "pending"
      },
      "StateTransitionReason": ""
    }
  ]
}
```

```
      "GroupName": "sand security",
      "GroupId": "sg-06b586adaf0fe5caf"
    }
  ],
  "SourceDestCheck": true,
  "StateReason": {
    "Code": "pending",
    "Message": "pending"
  },
  "VirtualizationType": "hvm",
  "CpuOptions": {
    "CoreCount": 1,
    "ThreadsPerCore": 1
  },
  "CapacityReservationSpecification": {
    "CapacityReservationPreference": "open"
  },
  "MetadataOptions": {
    "State": "pending",
    "HttpTokens": "optional",
    "HttpPutResponseHopLimit": 1,
    "HttpEndpoint": "enabled"
  }
}
},
"OwnerId": "452341676652",
"ReservationId": "r-0a2720eaaf494a562"
}
```

- SSH to the instance:

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

```
(aws_env) hnmn3@work:~/workspace/TheWork/AWS_Docker_Python$ ssh -i sand_security.pem ubuntu@ec2-18-221-173-121.us-east-2.compute.amazonaws.com
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-1051-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Mon Dec 16 07:18:42 UTC 2019

System load:  0.0               Processes:    86
Usage of /:   13.7% of 7.69GB   Users logged in: 0
Memory usage: 14%              IP address for eth0: 172.31.44.42
Swap usage:   0%

0 packages can be updated.
0 updates are security updates.

Last login: Mon Dec 16 07:16:45 2019 from 175.111.128.42
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-44-42:~$
```

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:

```
ubuntu@ip-172-31-43-89:~$ sudo docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS
7a53ec8cda36   ubuntu    "/bin/bash"             5 hours ago   Up 5 hours   0.0.0.0:2222->22/t
cp   ubuntu_pyspark
ubuntu@ip-172-31-43-89:~$
```

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

```
hnmn3@work:~$ ssh -p 2222 root@ec2-52-15-185-57.us-east-2.compute.amazonaws.com
root@ec2-52-15-185-57.us-east-2.compute.amazonaws.com's password:
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-1051-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Tue Dec 17 11:48:07 2019 from 175.111.128.42
root@7a53ec8cda36:~#
```

- run the word_count program:
spark-submit word_count.py
- sample output screenshot:

```
19/12/17 12:45:59 INFO FileOutputCommitter: File Output Committer Algorithm version is 1
19/12/17 12:45:59 INFO PythonRunner: Times: total = 54, boot = -1145, init = 1199, finish = 0
19/12/17 12:45:59 INFO FileOutputCommitter: Saved output of task 'attempt_20191217124556_0008_m_000000_0' to file:/root/output/_temporary/0/task_20191217124556_0008_m_000000
19/12/17 12:45:59 INFO SparkHadoopMapRedUtil: attempt_20191217124556_0008_m_000000_0: Committed
19/12/17 12:45:59 INFO Executor: Finished task 0.0 in stage 1.0 (TID 1). 2190 bytes result sent to driver
19/12/17 12:45:59 INFO TaskSetManager: Finished task 0.0 in stage 1.0 (TID 1) in 347 ms on 93d3793blced (executor driver) (1/1)
19/12/17 12:45:59 INFO DAGScheduler: ResultStage 1 (runJob at SparkHadoopWriter.scala:78) finished in 0.412 s
19/12/17 12:45:59 INFO DAGScheduler: Job 0 is finished. Cancelling potential speculative or zombie tasks for this job
19/12/17 12:45:59 INFO TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks have all completed, from pool
19/12/17 12:45:59 INFO TaskSchedulerImpl: Killing all running tasks in stage 1: Stage finished
19/12/17 12:45:59 INFO DAGScheduler: Job 0 finished: runJob at SparkHadoopWriter.scala:78, took 3.210266 s
19/12/17 12:45:59 INFO SparkHadoopWriter: Job job_20191217124556_0008 committed.
19/12/17 12:45:59 INFO SparkContext: Invoking stop() from shutdown hook
19/12/17 12:45:59 INFO SparkUI: Stopped Spark web UI at http://93d3793blced:4040
19/12/17 12:45:59 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
19/12/17 12:45:59 INFO MemoryStore: MemoryStore cleared
19/12/17 12:45:59 INFO BlockManager: BlockManager stopped
19/12/17 12:45:59 INFO BlockManagerMaster: BlockManagerMaster stopped
19/12/17 12:45:59 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
19/12/17 12:45:59 INFO SparkContext: Successfully stopped SparkContext
19/12/17 12:45:59 INFO ShutdownHookManager: Shutdown hook called
19/12/17 12:45:59 INFO ShutdownHookManager: Deleting directory /tmp/spark-920c8e6f-8306-4c25-93ac-02b4f6da87a8
19/12/17 12:45:59 INFO ShutdownHookManager: Deleting directory /tmp/spark-3876bafd-097a-4236-9b2b-af6b42a985f4/pyspark-a400f3cb-febd-4c5f-82c4-2112720240db
19/12/17 12:46:00 INFO ShutdownHookManager: Deleting directory /tmp/spark-3876bafd-097a-4236-9b2b-af6b42a985f4
root@93d3793blced:~#
```

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