**Part 1: Setup AWS and Docker**

1.1

* Create your AWS access keys. Go to the menu "My Security Credentials" and find "Access keys" to create the Access Key ID and Secret Access Key.
* Setup AWS command line tool. Use the following commands in your ubuntu box.
* > sudo apt-get install python-pip
* > sudo pip install awscli

use "aws help" to explore the command usage. Config the aws tool

> aws configure

AWS Access Key ID [None]: XXXXXXXX

AWS Secret Access Key [None]: XXXXXXXX

Default region name [None]: us-east-1

Default output format [None]: json

* Create a security group.
* > aws ec2 create-security-group --group-name your\_security\_group\_name --description "Your security group"
* > aws ec2 authorize-security-group-ingress --group-name your\_security\_group\_name --protocol tcp --port 22 --cidr 0.0.0.0/0
* Create a key pair.
* > aws ec2 create-key-pair --key-name your\_key\_name
* --query 'KeyMaterial' --output text > your\_key\_name.pem
* > chmod 400 your\_key\_name.pem
* Some instances may need to specify the subnet id. Please check [VPC and subnets](https://docs.aws.amazon.com/vpc/latest/userguide/default-vpc.html) or use "aws ec2 describe-subnets" to find the subnet information.

 Need to use boto3 to access EC2 and S3.

1.2 setup Docker in your Linux system.

Now, answer the following questions:

**Question 1.1** create an instance with the commands discussed in the lecture note. 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.

**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. Paste your code in the answer.

**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). Paste your code here.

**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) Paste the content of your dockerfile and your Docker hub link here. (2) Give 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.

**Part 2: Remotely monitoring VM instances and Docker containers**

**Question 2.1** In this task, you will implement a tool with Python Boto3 library and the [Paramiko](http://www.paramiko.org/) 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".

**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".

**Deliverables**

Turn in (1) the code and answers for Questions 1.1-1.4 and 2.1-2.2 in one PDF file.

(2) All codes in zip files

(3) Read Me - explaining the execution instructions.

(4) Screen video file showing the execution

**Make sure that you have terminated all instances after finishing your work!**