

Cloud computing – project 2 - Using AWS and Docker

1.1:

Command:

```
aws ec2 run-instances --image-id ami-cd0f5cb6 --security-group-ids sg-0b0b764f630fcea2c --count 1 --instance-type t2.micro --key-name ec2_key --subnet-id subnet-9a1a1eb4
```

```
(base) ~ % aws ec2 run-instances --image-id ami-cd0f5cb6 --security-group-ids sg-0b0b764f630fcea2c --count 1 --instance-type t2.micro --key-name ec2_key --subnet-id subnet-9a1a1eb4
{
  "Groups": [],
  "Instances": [
    {
      "AmiLaunchIndex": 0,
      "ImageId": "ami-cd0f5cb6",
      "InstanceId": "i-095d64184d60313b4",
      "InstanceType": "t2.micro",
      "KeyName": "vaishnavi_ec2_key",
      "LaunchTime": "2019-10-27T00:30:14.000Z",
      "Monitoring": {
        "State": "disabled"
      },
      "Placement": {
        "AvailabilityZone": "us-east-1a",
        "GroupName": "",
        "Tenancy": "default"
      },
      "PrivateDnsName": "ip-172-31-81-83.ec2.internal",
      "PrivateIpAddress": "172.31.81.83",
      "ProductCodes": [],
      "PublicDnsName": "",
      "State": {
        "Code": 0,
        "Name": "pending"
      },
      "StateTransitionReason": "",
      "SubnetId": "subnet-9a1a1eb4",
      "VpcId": "vpc-d5d69caf",
      "Architecture": "x86_64",
      "BlockDeviceMappings": [],
      "ClientToken": "",
      "EbsOptimized": false,
      "Hypervisor": "xen",
      "NetworkInterfaces": [
        {
          "Attachment": {
            "AttachTime": "2019-10-27T00:30:14.000Z",
            "AttachmentId": "eni-attach-0dbd5190ec94ec5f3",
            "DeleteOnTermination": true,
            "DeviceIndex": 0,
            "Status": "attaching"
          },
          "Description": "",
          "Groups": [
            {
              "GroupName": "vaishnavi_AWS",
              "GroupId": "sg-0b0b764f630fcea2c"
            }
          ],
          "Ipv6Addresses": [],
          "MacAddress": "12:a7:3d:72:3f:50",
          "NetworkInterfaceId": "eni-0b31e295a5e01d1ab",
          "OwnerId": "000030502559",
          "PrivateDnsName": "ip-172-31-81-83.ec2.internal",
          "PrivateIpAddress": "172.31.81.83",

```

SSH:login_command : ssh -i [REDACTED]_ec2_key.pem ubuntu@3.83.204.90

```
(base) [REDACTED]-MBP:~ [REDACTED]$ ssh -i [REDACTED]_ec2_key.pem ubuntu@3.83.204.90
The authenticity of host '3.83.204.90 (3.83.204.90)' can't be established.
ECDSA key fingerprint is SHA256:GozrNhSSoEfi1/7FHPYqedJDeXJCszavSxgJIy7oq9Q.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '3.83.204.90' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.0-1022-aws x86_64)

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

Get cloud support with Ubuntu Advantage Cloud Guest:
http://www.ubuntu.com/business/services/cloud

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

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-81-83:~$
```

1.2:

Command: python aws_boto_createInstances.py 2

Code:

```
import boto3
import time
import sys

ec2=boto3.resource('ec2')

#check status of created instances
def check_Instance_status(instance_id):

    status = True # not running
    while(status):
        i=ec2.Instance(instance_id)
        #print(i.state['Code'])
        #check condition for "pending" and if pending sleep for 10 secs
        if i.state['Code']==0:
            time.sleep(10)
            status=True
        #check condition for shutting-
        down ,terminated,stopping and stopped,if instance status condition is met break loop w
        ith current status of instance
```

```

        elif i.state['Code']==32 or i.state['Code']==48 or i.state['Code']==64 or i.state['Code']==80 :
            if i.state['Code']==32 :
                error_status = 'Shutting-down'
            elif i.state['Code']==48 :
                error_status = 'Terminated'
            elif i.state['Code'] == 64:
                error_status = 'Stopping'
            elif i.state['Code']==80:
                error_status = 'Stopped'
            print("Instance {} has problem to start up or {}".format(instance_id,error_status))

            instance_id=None
            status=False

            #condition check for "running" status of ec2 instance
            elif i.state['Code']==16:
                print("Instance {} has sucessfully started/running".format(instance_id))
                status = False

        return instance_id

#create ec2 instance
def create_ec2_instances(count_instances):
    #creates required number of instances for given ImageId
    return ec2.create_instances(ImageId='ami-
cd0f5cb6',InstanceType='t2.micro',SecurityGroupIds=['sg-0b0b764f630fcea2c'], \
                                SubnetId='subnet-
9a1a1eb4', MaxCount=count_instances, MinCount=1, KeyName='vaishnavi_ec2_key')

if __name__=="__main__":
    num_instances=sys.argv[1]
    instances_created=create_ec2_instances(int(num_instances))
    print("Creating {} EC2 instances".format(int(num_instances)))
    instance_running=[]
    #iteratively check status of the instance created
    for i in instances_created:
        print("Instances with {} id has startup status {}".format(i.id, i.state))
        instance_running.append(check_Instance_status(i.id))
    #print list of instances with running status
    print("list of instance id's with status code as running are \n",instance_running)

    #terminate all running instances .
    for i in ec2.instances.all():
        i.terminate()

```

1.3: Bucket name: ~~xxxxxxxx-cc-bucket~~4a0d4f90-c611-43ae-85c8-c4304d8f625c

Command: python aws_boto_s3_copy.py

Code:

```
import boto3
import botocore
import uuid,time

#generates unique bucket name
def generate_bucket_name(bucketName):
    #generates unique bucket name with bucketName argument as prefix for readable bucket name
    return "".join([bucketName,str(uuid.uuid4())])
#create bucket at particular location where boto3 creates the session
def create_bucket(bucketName,s3):
    session=boto3.session.Session()
    region=session.region_name
    bucketName=generate_bucket_name(bucketName)
    #print(bucketName,region)
    if region == 'us-east-1':
        response=s3.create_bucket(Bucket=bucketName)
    else:
        response=s3.create_bucket(bucket=bucketName>CreateBucketConfiguration={'LocationConstraint': region})

    return bucketName,response

#making bucket public
def make_bucket_public(bucket_name):
    time.sleep(1)
    s3.put_bucket_acl(Bucket=bucket_name,ACL='public-read-write')
    print("ACL set to public - read - write for bucket {}".format(bucket_name))

#copy files from one bucket to another
def bucket_copy(source_bucket_name,destination_bucket_name,key_filename):
    source= {'Bucket':source_bucket_name,'Key':key_filename}
    s3.copy(source,destination_bucket_name,key_filename)
    print("File {} sucessfully copied from {} to {}".format(key_filename,source_bucket_name,destination_bucket_name))

#read contents of bucket
def read_files(bucketName,key):
    data=s3.get_object(Bucket=bucketName,Key=key)
    #(data)
    return data['Body'].read().decode()
```

```

#perform operations on bucket
#check for existence of bucket
#check if bucket is not empty

def Bucket_operations(source_bucket_name,destination_bucket_name):
    #code to check if bucket is not empty
    exists=True
    try:
        s3.head_bucket(Bucket=source_bucket_name)
        print("{} Bucket exist".format(source_bucket_name))
    except botocore.exceptions.ClientError as e:
        #check for any exception if error is 404 bucket does not exist
        #throw exception that the given bucket does not exist
        error_code=int(e.response['Error']['Code'])
        if error_code==404:
            print("{} Bucket does not exist".format(source_bucket_name))
            exists=False

    # perform file read and copy file content to destination bucket if bucket is not empty
    if (exists):
        #finding all files in source bucket
        http_response = s3.list_objects_v2(Bucket=source_bucket_name)
        #return file names if files exist else store None in Keys_exist
        keys_exist=http_response.get('Contents')
        if keys_exist!=None:
            for file_metadata in http_response.get('Contents',[]):
                key = file_metadata['Key']
                #if file_metadata['Size']!=0:
                try:
                    # read contents in files
                    content = read_files(source_bucket_name, key)
                    print("=====" * 10)
                    print("Filename name : {}".format(key))
                    # print content
                    print("*****"*10)
                    print(content)
                    print("*****"*10)
                    # copying files to bucket
                    bucket_copy(source_bucket_name, destination_bucket_name, key)
                except Exception as e:
                    #exception will be passed if directory or file is empty
                    pass
            #else:
            #    print("{} file is empty".format(key))
        else:
            print("Bucket is empty")
    #create resource s3

```

```

if __name__=="__main__":
    s3=boto3.client('s3')
    source_bucket_name="wsu2017fall"
    destination_bucket_name,b_response=create_bucket("vaishnavi-cc-bucket",s3)
    print("My bucket Name{}".format(destination_bucket_name))
    make_bucket_public(destination_bucket_name)
    Bucket_operations(source_bucket_name,destination_bucket_name)

```

1.4:

1.4.1 :Docker_File

```

FROM ubuntu:latest

#Intial setup of commands and required softwares.
RUN apt-get update &&\
    apt-get install -y openssh-server openjdk-8-jdk python3 curl &&\
    ln -s /usr/bin/python3 /usr/bin/python && \
    apt-get clean

#ssh server setup
RUN ssh-keygen -f /root/.ssh/id_rsa -t rsa -
N "" && chmod 644 /root/.ssh/id_rsa.pub && \
    chmod 600 /root/.ssh/id_rsa && echo "" > /root/.ssh/known_hosts
#copy localhost/client publickey to docker container
COPY authorized_keys /root/.ssh/authorized_keys
#By default ubuntu is not creating this folder so need to create it and change the per
mission mode.
RUN mkdir /var/run/sshd && chmod 0755 /var/run/sshd

#Spark-Setup
RUN curl -O https://www-us.apache.org/dist/spark/spark-2.4.4/spark-2.4.4-bin-
hadoop2.7.tgz && tar xvf spark-2.4.4-bin-hadoop2.7.tgz
RUN echo 'export SPARK_HOME=/spark-2.4.4-bin-hadoop2.7'>> ~/.bashrc
RUN echo 'JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64/jre'>> ~/.bashrc
RUN echo 'PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin:$JAVA_HOME/bin'>> ~/.bashrc
# RUN echo ${SPARK_HOME} \
#     ${JAVA_HOME}\
#     ${PATH}
#spark word countprograms and text file
COPY wordcount.py /
COPY wordcount_inputfile /
#RUN /usr/sbin/sshd
EXPOSE 22
CMD ["/usr/sbin/sshd","-D"]

```

1.4.1: Docker_hub_link:

[https://cloud.docker.com/repository/docker/\[REDACTED\]/ubuntu_spark](https://cloud.docker.com/repository/docker/[REDACTED]/ubuntu_spark)

1.4.2:

Command: `docker run -ti -d -p 2222:22 [REDACTED]/ubuntu_spark`

1.4.3:

Login command: `ssh root@localhost -p 2222`

test-run command: `spark-submit /wordcount.py /wordcount_inputfile`

screenshots for remote login and test_run success:

```
debug: Exit status 0
(base) [REDACTED]-MacBook-Pro:vaishnavi [REDACTED] $ ssh root@localhost -p 2222
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.9.184-linuxkit 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 Oct 29 02:01:43 2019 from 172.17.0.1
root@5d36fd1b46fa:~#
```

```
(base) [REDACTED]-MacBook-Pro:vaishnavi [REDACTED] $ ssh root@localhost -p 2222
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.9.184-linuxkit 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 Oct 29 02:01:43 2019 from 172.17.0.1
root@5d36fd1b46fa:~# spark-submit /wordcount.py /wordcount_inputfile
19/10/29 02:06:11 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
19/10/29 02:06:12 INFO SparkContext: Running Spark version 2.4.4
19/10/29 02:06:12 INFO SparkContext: Submitted application: PythonWordCount
19/10/29 02:06:12 INFO SecurityManager: Changing view acls to: root
19/10/29 02:06:12 INFO SecurityManager: Changing modify acls to: root
19/10/29 02:06:12 INFO SecurityManager: Changing view acls groups to:
19/10/29 02:06:12 INFO SecurityManager: Changing modify acls groups to:
19/10/29 02:06:12 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(root); groups with view permissions: Set(root); groups with modify permissions: Set(root); groups with modify permissions: Set()
19/10/29 02:06:12 INFO Utils: Successfully started service 'sparkDriver' on port 41471.
19/10/29 02:06:12 INFO SparkEnv: Registering MapOutputTracker
19/10/29 02:06:12 INFO SparkEnv: Registering BlockManagerMaster
19/10/29 02:06:12 INFO BlockManagerMasterEndpoint: Using org.apache.spark.storage.DefaultTopologyMapper for getting topology information
19/10/29 02:06:12 INFO BlockManagerMasterEndpoint: BlockManagerMasterEndpoint up
19/10/29 02:06:13 INFO DiskBlockManager: Created local directory at /tmp/blockmgr-80565aef-d8be-4a27-8098-639c8a7098e5
19/10/29 02:06:13 INFO MemoryStore: MemoryStore started with capacity 366.3 MB
19/10/29 02:06:13 INFO SparkEnv: Registering OutputCommitCoordinator
19/10/29 02:06:13 INFO Utils: Successfully started service 'SparkUI' on port 4040.
19/10/29 02:06:13 INFO SparkUI: Bound SparkUI to 0.0.0.0, and started at http://5d36fd1b46fa:4040
19/10/29 02:06:13 INFO Executor: Starting executor ID driver on host localhost
19/10/29 02:06:13 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 45691.
19/10/29 02:06:13 INFO NettyBlockTransferService: Server created on 5d36fd1b46fa:45691
19/10/29 02:06:13 INFO BlockManager: Using org.apache.spark.storage.RandomBlockReplicationPolicy for block replication policy
19/10/29 02:06:13 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, 5d36fd1b46fa, 45691, None)
19/10/29 02:06:13 INFO BlockManagerMasterEndpoint: Registering block manager 5d36fd1b46fa:45691 with 366.3 MB RAM, BlockManagerId(driver, 5d36fd1b46fa, 45691, None)
19/10/29 02:06:13 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, 5d36fd1b46fa, 45691, None)
19/10/29 02:06:13 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, 5d36fd1b46fa, 45691, None)
19/10/29 02:06:14 INFO SharedState: Setting hive.metastore.warehouse.dir ('null') to the value of spark.sql.warehouse.dir ('file:/root/spark-warehouse').
19/10/29 02:06:14 INFO SharedState: Warehouse path is 'file:/root/spark-warehouse'.
19/10/29 02:06:15 INFO StateStoreCoordinatorRef: Registered StateStoreCoordinator endpoint
19/10/29 02:06:18 INFO FileSourceStrategy: Pruning directories with:
19/10/29 02:06:18 INFO FileSourceStrategy: Post-Scan Filters:
19/10/29 02:06:18 INFO FileSourceStrategy: Output Data Schema: struct<value: string>
19/10/29 02:06:18 INFO FileSourceScanExec: Pushed Filters:
19/10/29 02:06:19 INFO CodeGenerator: Code generated in 335.2484 ms
19/10/29 02:06:19 INFO MemoryStore: Block broadcast_0 stored as values in memory (estimated size 282.9 KB, free 366.0 MB)
19/10/29 02:06:19 INFO MemoryStore: Block broadcast_0_piece0 stored as bytes in memory (estimated size 23.2 KB, free 366.0 MB)
19/10/29 02:06:19 INFO BlockManagerInfo: Added broadcast_0_piece0 in memory on 5d36fd1b46fa:45691 (size: 23.2 KB, free: 366.3 MB)
19/10/29 02:06:19 INFO SparkContext: Created broadcast 0 from javaToPython at NativeMethodAccessorImpl.java:0
19/10/29 02:06:19 INFO FileSourceScanExec: Planning scan with bin packing, max size: 4194304 bytes, open cost is considered as scanning 4194304 bytes.
19/10/29 02:06:19 INFO Deprecation: mapred.output.dir is deprecated. Instead, use mapreduce.output.fileoutputformat.outputdir
19/10/29 02:06:19 INFO HadoopMapRedCommitProtocol: Using output committer class org.apache.hadoop.mapred.FileOutputCommitter
```



```

19/10/29 02:06:22 INFO TaskSchedulerImpl: Removed taskSet 3.0, whose tasks have all completed, from pool
19/10/29 02:06:22 INFO DAGScheduler: ResultStage 3 (collect at /wordcount.py:41) finished in 0.087 s
19/10/29 02:06:22 INFO DAGScheduler: Job 1 finished: collect at /wordcount.py:41, took 0.092343 s
This: 1
is: 1
word: 1
count: 1
program: 1
test: 1
file: 2
for: 1
docker: 1
spark: 1
single: 1
node: 1
cluster: 1
setup: 1
19/10/29 02:06:22 INFO SparkUI: Stopped Spark web UI at http://5d36fd1b46fa:4040
19/10/29 02:06:22 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
19/10/29 02:06:22 INFO MemoryStore: MemoryStore cleared
19/10/29 02:06:22 INFO BlockManager: BlockManager stopped
19/10/29 02:06:22 INFO BlockManagerMaster: BlockManagerMaster stopped
19/10/29 02:06:22 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
19/10/29 02:06:22 INFO SparkContext: Successfully stopped SparkContext
19/10/29 02:06:23 INFO ShutdownHookManager: Shutdown hook called
19/10/29 02:06:23 INFO ShutdownHookManager: Deleting directory /tmp/spark-acdc8aa0-64ee-4b88-be8d-adcb0cca786e/pyspark-11238659-9c9b-4403-87f6-88c66fc0d150
19/10/29 02:06:23 INFO ShutdownHookManager: Deleting directory /tmp/spark-62416749-587d-480c-9d01-b781f54925df
19/10/29 02:06:23 INFO ShutdownHookManager: Deleting directory /tmp/spark-acdc8aa0-64ee-4b88-be8d-adcb0cca786e
root@5d36fd1b46fa:~#

```

Part 2: Monitoring VM instances and Docker containers

2.1:

command: python paramaiko_cpuUsage.py

Code:

```

import paramiko
import boto3
import time
import aws_boto_createInstances

def cpu_utilization():
    ec2_client=boto3.client("ec2")

    instances_details=ec2_client.describe_instances()

    #print(ec2_client.describe_instance_status())

    username="ubuntu"
    #connect SSH_client
    ssh_client=paramiko.SSHClient()

    #Connect to an SSH server and authenticate to it. The server's host key is checked
    against the system host keys
    ssh_client.load_system_host_keys()

    #Policy for automatically adding the hostname and new host key to the local HostKe
ys object

```



```

ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())

#load key
key=paramiko.RSAKey.from_private_key_file("/Users/vaishnaviv/vaishnavi_ec2_key.pem")

#fetch instance details and display cpu utilization
while(True):
    for reservation in instances_details['Reservations']:
        for instance in reservation['Instances']:
            instance_id=instance['InstanceId']
            if instance['State']['Name']=='running':
                #connection to SSH server
                ssh_client.connect(hostname=instance['PublicDnsName'],username=use
rname,pkey=key)
                stdin,stdout,stderr=ssh_client.exec_command("top -
bn1 | grep Cpu")
                cpu_info = stdout.readline()
                print("%s \t %s"%(instance_id,cpu_info))
            time.sleep(5)

if __name__=="__main__":

    num_instances = 2
    instances_created = aws_boto_createInstances.create_ec2_instances(int(num_instance
s))
    print("Creating {} EC2 instances".format(int(num_instances)))
    instance_running=[]
    #iteratively check status of the instance created
    for i in instances_created:
        print("Instances with {} id has startup status {}".format(i.id, i.state))
        instance_running.append(aws_boto_createInstances.check_Instance_status(i.id))
    #print list of instances with running status
    print("list of instance id's with status code as running are \n",instance_running)
    time.sleep(10)
    cpu_utilization()

```

2.2:

Command: python paramaiko_cpuUsage_containers.py

Code:

```
import paramiko
import boto3
import time,sys
import aws_boto_createInstances

def cpu_usage_container():
    ec2_client=boto3.client("ec2")

    instances_details=ec2_client.describe_instances()
    #print(instances_details)

    username="ubuntu"
    #connect SSH_client
    ssh_client=paramiko.SSHClient()

    #Policy for automatically adding the hostname and new host key to the local HostKeys object
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())

    #load key
    key=paramiko.RSAKey.from_private_key_file("/Users/vaishnaviv/vaishnavi_ec2_key.pem")

    #installing docker on instances running
    error=[]
    for reservation in instances_details['Reservations']:
        for instance in reservation['Instances']:
            instance_id = instance['InstanceId']
            if instance['State']['Name']=='running':
                try:
                    # connection to SSH server on instances at given publicdns id
                    ssh_client.connect(hostname=instance['PublicDnsName'], username=username, pkey=key)

                    stdin, stdout, stderr = ssh_client.exec_command("sudo apt-get -y update ")

                    stdout.channel.recv_exit_status()
                    stdin, stdout, stderr = ssh_client.exec_command("sudo apt-get install -y apt-utils")

                    stdout.channel.recv_exit_status()
                    stdin, stdout, stderr = ssh_client.exec_command("sudo apt autoremove docker docker-engine docker.io")

                    stdout.channel.recv_exit_status()
```

```

        print("Docker installation in progress")
        D_stdin, D_stdout, D_stderr = ssh_client.exec_command("sudo apt-
get install -y docker.io")
        D_stdout.channel.recv_exit_status()
        # #print(stderr.readlines())
        stdin, stdout, stderr = ssh_client.exec_command("sudo chmod 666
/var/run/docker.sock")
        # print(stdout.readlines())
        # print(stderr.readlines())
        print("docker service start in progress")
        stdin, stdout, stderr = ssh_client.exec_command('sudo systemctl
start docker && sudo systemctl enable docker',get_pty=True)
        print(stdout.readlines())
        print("executing docker")
        run_stdin, run_stdout, run_stderr = ssh_client.exec_command("sud
o docker run -d -t ubuntu sh")
        #print(run_stdout.readline())
        container_id=run_stdout.readline().rstrip()
        #print(container_id)
        #print(run_stderr.readline())
        print("succesfully ran docker whose created container id is {} on
instance {}".format(container_id,instance_id))

    except Exception as e:
        print(e)

#fetch instance details and display cpu utilization
#press contol+c to exit the while loop
while(True):
    container_list=[]
    try:
        for reservation in instances_details['Reservations']:
            for instance in reservation['Instances']:
                instance_id=instance['InstanceId']
                if instance['State']['Name']=='running':
                    #connection to SSH server
                    ssh_client.connect(hostname=instance['PublicDnsName'],user
name=username,pkey=key)
                    stdin, stdout, stderr = ssh_client.exec_command("sudo do
cker ps | grep ubuntu")
                    for output in stdout.readlines():
                        container_id=output.split()[0]
                        container_list.append(container_id)
                        stdin,stdout,stderr=ssh_client.exec_command("sudo do
cker exec {} top -bn1 | grep Cpu".format(container_id))
                        cpu_info=stdout.readline()

```

```

        print("{} \t {} \t {}".format(instance_id, container_id
,cpu_info))

        #sleep for 5 sec and run the loop again
        time.sleep(5)
        print("Press control+c to terminate the loop which executes every 5 se
conds")

    except KeyboardInterrupt as e:
        print("Docker kill is in progress")
        for container_id in container_list:
            ssh_client.exec_command("sudo docker kill {}".format(container_id)
)

        print("dockers killed sucessfully")
        _in,_out,_error=ssh_client.exec_command("sudo docker ps")
        #print(_out.readlines())
        # terminate all running instances .
        for i in aws_boto_createInstances.ec2.instances.all():
            i.terminate()
        print("instances killed succesfully")

        sys.exit()
if __name__=="__main__":
    num_instances=2
    instances_created=aws_boto_createInstances.create_ec2_instances(int(num_instances)
)

    print("Creating {} EC2 instances".format(int(num_instances)))
    instance_running=[]
    #iteratively check status of the instance created
    for i in instances_created:
        print("Instances with {} id has startup status {}".format(i.id, i.state))
        instance_running.append(aws_boto_createInstances.check_Instance_status(i.id))
    #print list of instances with running status
    print("list of instance id's with status code as running are \n",instance_running)
    #call to cpu utilization function
    time.sleep(20)
    cpu_usuage_container()

```