

University of Western Australia
CITS5503 - Cloud Computing
Labs 1-4 Report

Student Name: Tao Hu
Student Id: 23805764

Introduction	3
Lab 1 - Intro Setup	3
Installing VM software and Kali Linux	3
Verify python3 version	4
Install pip	4
Install AWS CLI	4
Install awscli with pip	5
AWS account and login	5
Obtain access key id and secret access key	6
Configure AWS CLI	7
Install boto3	7
Testing the environment	7
Testing the python environment	8
Python program to print table form region data	8
Lab -2 EC2 Docker	10
Create an EC2 instance using awscli	10
[1] Create a security group	10
[2] Authorise inbound traffic for ssh	10
[3] Create a key pair to be used for ssh connection to the EC2 instance	11
[4] Create the ec2 instance	12
[Optional] Add tag to the created ec2 instance	12
[5] Get the public Ip address	13
[6] Connect to the instance	13
[7] Instance in the AWS console	14
Create another ec2 instance with Python Boto script	14
[s1] Start up python3	14
[s2] Create ec2 object	14
[1] Create a security group	14
[2] Authorise inbound traffic for ssh	15
[3] Create a key pair to be used for ssh connection to the EC2 instance	16
[4] Create the second ec2 instance	17
[Optional] Add tag to the second ec2 instance	18
[5] Get the public IP address	18
[6] Connecting to the second ec2 instance	19
[7] Instance in the AWS console	19
Terminate both ec2 instances	19
Using Docker	20
[1] Install Docker	20
[2] Check the version	24
[3] Build and run an httpd container	25
[4] Create "Dockerfile"	25
[5] Build the docker image	25
[6] Run the image	26

[7] View in browser	27
[8] Other commands	27
Lab 3 - DynamoDB	28
Program	28
[1] Preparation	28
[2] Save to S3	30
[3] Restore from S3	35
[4] Write information about files to DynamoDB	37
Lab 4 - KMS Encryption	44
[1] Apply policy to restrict permissions on bucket	44
[2] AES Encryption using KMS	48
[3] AES Encryption using local python library pycryptodome	55
Answer the following question (Marked)	61

Introduction

This file contains the instructions I took and the explanations for completing labs 1-4 of the unit CITS5503 - Cloud Computing during 2023 semester 2.

The device I have used to complete all the labs is a M1 chip Macbook air. The virtual machine software used is UTM and the linux OS virtualised is Kali Linux.

Lab 1 - Intro Setup

The purpose of this lab is to install a Linux OS using a VM software, installing python 3.8 and required packages using pip, setting up the AWS account and connecting to AWSCLI.

Installing VM software and Kali Linux

From instructions in section 1.2 of the contents under the link <https://uwacyber.gitbook.io/cits1003/cits1003-labs/setting-up-your-laptop>. I installed UTM software onto my Macbook air.

Following the recommendations of the lab description. Kali Linux is chosen as the OS I will download and install using UTM.

From this link <https://www.kali.org/get-kali/#kali-installer-images>, I downloaded the recommended Kali Linux for Apple Silicon (ARM64).

From this link <https://www.kali.org/docs/virtualization/install-utm-guest-vm/>, I followed the instructions to correctly install Kali Linux with UTM.

A summary of what I did:

1. Click add create VM
2. Select Virtualize
3. Select Other
4. Click Browse...
 - a. Find the downloaded Kali Linux .iso file
 - b. Select and confirm
5. Allocate 4096 MB and 4 CPU Cores
6. Allocate 20 GB of storage space
7. Ignore share directory settings and continue
8. Name OS to Kali Linux and Save
9. Click on setting of the new VM
10. Click + New...
 - a. From the drop down select Serial
11. Click save
12. Start VM
13. From the two windows popped up focus on one with name Terminal 1

- a. In the cli select install
 - b. Follow the prompts with correct information
 - c. Adding my location data, timezone, username, password...
 - d. Under package selection no extra packages is selected other than xFce
 - e. On finish close terminal 1 cli window
14. Click settings of VM again
 - a. Right click on serial and click remove
 - b. Save
15. Focus on the right panel displaying VM data
 - a. Scroll to bottom, there is a selection component
 - b. Click on the select component and select clear
16. Now start the VM and it will successfully boot.

Verify python3 version

First try to verify the python version to see if an upgrade required as the lab description specifies python 3.8.x above is to be used.

```
python3 -V
```

The above command is run.

```
Python 3.11.2
```

The above output shows that the python version inside my kali linux OS is already above 3.8. So there is no need for apt update and re installing a higher version python.

Install pip

First verify if pip is installed.

```
pip -V
```

The above command prints the pip version.

```
pip 23.0.1 from /usr/lib/python3/dist-packages/pip (python 3.11)
```

The above output shows that my OS already has pip installed and it is also with the correct python version. Hence there was no need to re-install pip with apt.

Install AWS CLI

Following instructions provided by aws doc from this link

<https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html>

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-aarch64.zip" -o  
"awscliv2.zip"  
unzip awscliv2.zip  
sudo ./aws/install
```

The above commands attempt to install aws.

```
aws -version
```

Using the above to verify if aws is installed

```
aws-cli/2.13.7 Python/3.11.4 Linux/6.1.0-kali9-arm64  
exe/aarch64.kali.2023 prompt/off
```

The above output shows that aws is indeed installed successfully.

```
rm awscliv2.zip  
rm -rd aws
```

Clean up by removing the downloaded aws install files

Install awscli with pip

```
pip3 install awscli -upgrade
```

Above command used to install awscli.

```
pip show awscli
```

Check that awscli is indeed installed.

AWS account and login

Now that I have AWS CLI installed I need to configure my identity, which requires an access key. To set up an aws account I will use my student email address as the username and the provided password for first time login.

From the lab instructions, go to this link <https://489389878001.signin.aws.amazon.com/console> to access my AWS account.

Sign in as IAM user

Account ID (12 digits) or account alias

489389878001

IAM user name

23805764@student.uwa.edu.au

Password

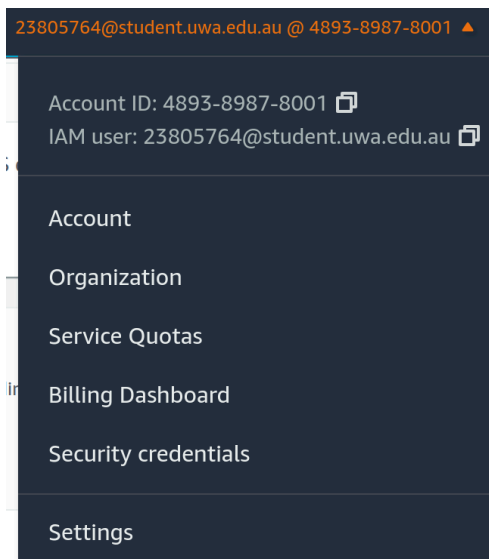
●●●●●●●●●●●●

☐ Remember this account

Sign in

Input my username as email and the password provided then click Sign in. Next input new password and then account will be initialised successfully and the dashboard is provided.

Obtain access key id and secret access key



Click on the profile and then click security credentials.
Click on the create access key button.

Use case



Command Line Interface (CLI)

You plan to use this access key to enable the AWS CLI to access your AWS account.

From use case choose Command Line Interface (CLI)

After this I was prompted to enter a description tag. I entered “To be used for aws cli for unit Cloud Computing”

After clicking next, the access key and secret access key is shown. I copied both of the values and saved them to a file in my google drive.

Configure AWs CLI

The command below is used to configure access key and secret access key.

```
aws configure
AWS Access Key ID [None]: # Inputted with my access key
AWS Secret Access Key [None]: # Inputted with my secret access key
Default region name [None]: ap-southeast-2 # From lab instruction
Default output format [None]: json # From lab instruction
```

Install boto3

Boto3 is a python library that has similar functionalities to the aws cli, allowing access to AWS resources.

```
pip3 install boto3
pip3 show boto3
```

The above command is used to first install boto3 then verify it is installed

```
Name: boto3
Version: 1.28.20
Summary: The AWS SDK for Python
Home-page: https://github.com/boto/boto3
Author: Amazon Web Services
Author-email:
License: Apache License 2.0
Location: /home/taohu/.local/lib/python3.11/site-packages
Requires: botocore, jmespath, s3transfer
Required-by:
```

The above output shows that I have successfully installed boto3 and the version is 1.28.20.

Testing the environment

From the lab instructions, the command below is run to test if configure is correct.

```
aws ec2 describe-regions -output table
```


After some reading, I understood that the above command retrieves the regions for ec2 enabled by my account in a table format.

Endpoint	OptInStatus	RegionName
ec2.ap-south-1.amazonaws.com	opt-in-not-required	ap-south-1
ec2.eu-north-1.amazonaws.com	opt-in-not-required	eu-north-1
ec2.eu-west-3.amazonaws.com	opt-in-not-required	eu-west-3
ec2.eu-west-2.amazonaws.com	opt-in-not-required	eu-west-2

The screen shot above shows the output.

Testing the python environment

```
python3
```

To open interactive python3.

```
>>> import boto3
>>> ec2 = boto3.client("ec2")
>>> response = ec2.describe_regions()
>>> print(response)
```

From the lab instructions, the above commands are entered to perform a similar task as done with AWS CLI previously. First create a ec2 instances then retrieve the regions data and output.

```
{'Regions': [{'Endpoint': 'ec2.ap-south-1.amazonaws.com', 'RegionName': 'ap-south-1', 'OptInStatus': 'opt-in-not-required'}, {'Endpoint': 'ec2.eu-north-1.amazonaws.com', 'RegionName': 'eu-north-1', 'OptInStatus': 'opt-in-not-required'}, {'Endpoint': 'ec2.eu-west-3.amazonaws.com', 'RegionName': 'eu-west-3', 'OptInStatus': 'opt-in-not-required'}, {'Endpoint': 'ec2.eu-west-2.amazonaws.com', 'RegionName': 'eu-west-2', 'OptInStatus': 'opt-in-not-required'}, {'Endpoint': 'ec2.eu-west-1.amazonaws.com', 'RegionName': 'eu-west-1', 'OptInStatus': 'opt-in-not-required'}, {'Endpoint': 'ec2.ap-northeast-3.amazonaws.com', 'RegionName': 'ap-northeast-3', 'OptInStatus': 'opt-in-not-required'}, {'Endpoint': 'ec2.ap-northeast-2.amazonaws.com', 'RegionName': 'ap-northeast-2', 'OptInStatus': 'opt-in-not-required'}, {'Endpoint': 'ec2.ap-northeast-1.amazonaws.com', 'RegionName': 'ap-northeast-1', 'OptInStatus': 'opt-in-not-required'}]}
```

The above screen shot shows the outputs, which can be seen in JSON format.

Python program to print table form region data

Here continues from the interactive python environment.

```
>>> print(isinstance(response, dict))
```

I already know that the output of the response is in JSON format. Now I test to see if the response is a dict data type by simply using the `isinstance` function. The output is True.

Now I need to figure out how to display the dict data in a table format containing only the “Endpoint” and “RegionName” values. From some research with google, I found out the easiest way is to use the python module `tabulate`.

```
from tabulate import tabulate
import boto3

ec2 = boto3.client("ec2")
response = ec2.describe_regions()
print(isinstance(response, dict))
print(
    tabulate(
        [[region["Endpoint"], region["RegionName"]] for region in
response["Regions"]],
        headers=["Endpoint", "RegionName"]
    )
)
```

I created a file called `lab1.py` containing the code above. It works by extracting the two values under the key “Endpoint” and “RegionName” for each region. Then `tabulate` consumes these data and displays them in table format with the provided headers when `print` is called.

Endpoint	RegionName
ec2.ap-south-1.amazonaws.com	ap-south-1
ec2.eu-north-1.amazonaws.com	eu-north-1
ec2.eu-west-3.amazonaws.com	eu-west-3
ec2.eu-west-2.amazonaws.com	eu-west-2
ec2.eu-west-1.amazonaws.com	eu-west-1
ec2.ap-northeast-3.amazonaws.com	ap-northeast-3
ec2.ap-northeast-2.amazonaws.com	ap-northeast-2
ec2.ap-northeast-1.amazonaws.com	ap-northeast-1
ec2.ca-central-1.amazonaws.com	ca-central-1
ec2.sa-east-1.amazonaws.com	sa-east-1
ec2.ap-southeast-1.amazonaws.com	ap-southeast-1
ec2.ap-southeast-2.amazonaws.com	ap-southeast-2
ec2.eu-central-1.amazonaws.com	eu-central-1
ec2.us-east-1.amazonaws.com	us-east-1
ec2.us-east-2.amazonaws.com	us-east-2
ec2.us-west-1.amazonaws.com	us-west-1
ec2.us-west-2.amazonaws.com	us-west-2

The above screen shot shows the output when `lab1.py` is executed with VSCode.

Lab -2 EC2 Docker

The purpose of this lab is to configure and create an EC2 instance, install and configure docker to run a hello world application.

Create an EC2 instance using awcli

[1] Create a security group

```
aws ec2 create-security-group --group-name 23805764-sg --description
"The security group for development envrionment"
```

The above command creates a security group to be used for ec2 instances. From the aws documentation, it is understood that the security group acts as the virtual firewall for controlling the input and output traffic.

```
{
  "GroupId": "sg-07b1ca79dd423b617"
}
```

The above is the output. From the aws documentation, “GroupId” refers to the id of the created security group.

[2] Authorise inbound traffic for ssh

```
aws ec2 authorize-security-group-ingress --group-name 23805764-sg
--protocol tcp --port 22 --cidr 0.0.0.0/0
```

The above command is used to configure the security group created earlier with name of 23805764-sg in the ec2 instance to allow it to receive traffic from the ip address configured by cidr, with port 22 and from tcp protocol.

```
{
  "Return": true,
  "SecurityGroupRules": [
    {
      "SecurityGroupRuleId": "sgr-067fcbcad053a7f0a",
      "GroupId": "sg-07b1ca79dd423b617",
      "GroupOwnerId": "489389878001",
      "IsEgress": false,
      "IpProtocol": "tcp",
      "FromPort": 22,
```

```

        "ToPort": 22,
        "CidrIpv4": "0.0.0.0/0"
    }
]
}

```

The above is the output. The value for “Return” is true which indicates that the command is successful. The rest contains information about the newly added security group rules.

[3] Create a key pair to be used for ssh connection to the EC2 instance

```

aws ec2 create-key-pair --key-name 23805764-key --query "KeyMaterial"
--output text > 23805764-key.pem

```

The above command will create a key pair with the provided unique key name and the key will be in pem format(the default format). This --query parameter specifies to extract the value from the output object. Then the final output which should be the private key is stored as text in 23805764-key.pem locally. From the documentation, aws will generate a key pair, keeping the public key and sending the private key to the user. The private key can be then used to authorise when accessing the ec2 instance from ssh.

```

chmod 400 23805764-key.pem
cat 23805764-key.pem

```

The above command is used to change permission of the key file to be only readable by the owner. Later one, I found out that this is very much required, where ssh will output an error if incorrect permission is found in the private key file.

```

-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAoD1p0GV08ShVQU+Bnso5zPaNlHG1bG0RerximeY6W6AkQW23
JZtRmsd7s0/rDFpzfjIMDKXa4Nhw1b5KjaS5R2dDe9FULrxfZVjzdvPFXYrp0whU
crmNp8B8UHX4Vq7I4pDI2cAB7gZtK9FB2VIsJx+n+DaZe4qHatjFFjmmATwSptq
F4mJOz/Sro+N7fatDn41XLvJoPKX/o1YE24r1BubVtWA0FX1dq1Kvmp5myL6nAht
rSONshdqxusrVWxuu2m+rCOGcqwJ977IX6wzSwCsp8NVyBM1vByCfKbLwxYQ420u
kLOevoA9WUj2jIbebdb0vpa02Uzh0i9SnzHHDQIDAQABaoIBAAnhX4nrBUAXdZJO
Xba1z3SU60vw6vfHjcdcoFPO/2Aw8qtaIYXfGtwJWtZGoj6jwSVR3q5U97cPgX2c
eypastX1RXL2aFNtaBBuuKxoAY5wB1Ts2ZAtYs+qLcAkVgi6REa8GLZcgyH0wXyH
LaFB0NkkuY2ziCk29HyhHLg7dxR+eCI8oRTxtwzpfWCXHBJa4EIAIveZpWsyncrI
+3FKxh3k2UqqTGT6Aw10VzrU2g2wCbqk/I2RMkrqBYIRBUL9AxjsfvXKdivhnuDG
/zUDIpu++EftOBaNPKgQ3KV1Nc1DvykoC6Mssbni50vVdi1y0hxZEfNe46Ext8NJ
JtkDlUECgYEA74aKXkM7ti9Et8KNjX3gq0U9hGojMWXiamsKB9L+2gq+QmEUy+TC
mtkDjHUJV+6jE11hjYP3nFR8mVW8mg/ZVDabYS3iWC4J7NkUR9YTHyr3594pMN0j
HSnDMmEu70ocH9fZX+kwPqg50CvnpBY2mJ0SNMnPRsAmwkp53FoJIJUCgYEAq0LY

```

```
pNkhq1l85G9t3qRgmGDjoZHyjSd/Bcbyu50aZ1jK7rA4Pnf/P0q8tKV8cWxr1JAM
0cbeB8jmn0NIVJTPF1r1BDbEsvdLsY0oUUb+p8V4nNOEOvpwLYWE6p1rHH1wgc61
59KB3Hi073cWoEfAPcJgPctVho2AiAjthNSOlPkCgYAZ6LK06Y7e8RP80JWhNXmE
7TNvTE/M0xoXqsZl/EPG8By0b5PhTjhiqZDBs7M0CMMo7GLt/NVe2qCzsVt6SraW
bmW2SFFQeoJmVUDlWFmvKGwydGx8lZ8Du426uFpsVdbe9ukzf7TRQj8WPWHCmYCE
V5Et1prOK2DF6WpvFGoiVQKBgC27H9aFFHSI5Myq5I+7HQATfzU5n6cJkOWeBlSk
zG4kS6wI6jvkFwhzkIg8WEp/iMwt4X5yt/8kKlEEyl+rjNETvLFvXn7bgh7bLpTl
fEKYfGiEIp8NGL9PE/sBFJT6+fvvkqtgjsSsfJ59ElGiVzcmnEP03lWbm2K335TE
nX6hAoGAPMaUgsT+F791gfxJidYwz07XKDGnvOqUC7hINqXg+Itqkl0FArimCWdy
xVn/PwR0IKpF5w07HWkIh6ktdBbol5Xt1bj2CHMfgfMlAtqPLUF/f/ehQW4PJr4i
5z5g071xPR78G1GoC8iQo4iKpawKzqgM17CkJaERbmRJPztCbmG=
-----END RSA PRIVATE KEY-----
```

The above is the content of the private key file, which contains the private key created by aws.

[4] Create the ec2 instance

```
ec2 run-instances --image-id ami-d38a4ab1 --security-group-ids
23805764-sg --count 1 --instance-type t2.micro --key-name 23805764-key
--query "Instances[0].InstanceId"
```

The above command will create a new ec2 instance and configured to use the previously created security group and key. The image id is provided by the lab specification and the instance is of type t2.micro. Finally, the query option is used to filter out the json response to be received and only keep the `InstanceId`.

```
"i-00dce50ce3eb05af3"
```

The above is the output. This is the id of the newly created ec2 instance.

[Optional] Add tag to the created ec2 instance

```
aws ec2 create-tags --resources "i-00dce50ce3eb05af3" --tags
key=Name,Value=23805764
```

The above command assigns a tag to the ec2 instance, with Key = “Name” and a value of my student id. No output was produced from the command, possibly successful.

<input type="checkbox"/>	23485011	i-0fd92c27f099e5db4	Running		t2.micro	2/2 checks passed	No alarms	+	ap-sou
<input type="checkbox"/>	23805764	i-00dce50ce3eb05af3	Running		t2.micro	2/2 checks passed	No alarms	+	ap-sou
<input type="checkbox"/>	-	i-08b0bd94cf8a4585a	Terminated		t2.micro	-	No alarms	+	ap-sou

The above screenshot was taken in my aws dashboard instances section. As it can be seen the new ec2 instance is created and running with my student id assigned as the tag.

[5] Get the public Ip address

```
aws ec2 describe-instances --instance-id "i-00dce50ce3eb05af3" --query  
"Reservations[0].Instances[0].PublicIpAddress"
```

The above command will retrieve the instance information using the instance id provided. The query parameter will filter the response to keep only the specified path.

```
"13.236.71.36"
```

The above is the output, which contains the public ip of my ec2 instance.

[6] Connect to the instance

```
ssh -i 23805764-key.pem ubuntu@13.236.71.36
```

The above command tries to perform a ssh connection with the ec2 instance created earlier. The -i command will specify the credentials for authorisation, which is the private key retrieved earlier. Then, the ec2 instance's public ip address is used to initiate the connection.

```
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-172-31-7-22:~$
```

The above is the output, which shows that I have successfully connected to the ec2 instance inside a ubuntu terminal environment.

```
Last login: Fri Aug 11 10:24:17 2023 from 60.242.162.18  
To run a command as administrator (user "root"), use "sudo <command>".  
See "man sudo_root" for details.  
individual files in /usr/share/doc/*/copyright.  
ubuntu@ip-172-31-7-22:~$
```

Above is a screenshot of ssh successfully connected.

[7] Instance in the AWS console

<input type="checkbox"/>	23485011	i-0fd92c27f099e5db4	Running		t2.micro	2/2 checks passed	No alarms	+	ap-southeast-1
<input type="checkbox"/>	23805764	i-00dce50ce3eb05af3	Running		t2.micro	2/2 checks passed	No alarms	+	ap-southeast-1
<input type="checkbox"/>	-	i-08b0bd94cf8a4585a	Terminated		t2.micro	-	No alarms	+	ap-southeast-1

The above screenshot taken inside the instances section shows that my ec2 instance is created running successfully with my student id as the name.

Create another ec2 instance with Python Boto script

[s1] Start up python3

```
python3
```

Using interactive python3 in terminal instead of writing .py files. Some boto calls may be wrong, e.g typo or parameter misuse, which will cause some commands to fail. It would be hard to then run the script again. Instead use the python3 in the terminal and perform boto3 commands step by step.

[s2] Create ec2 object

```
>>> import boto3
>>> import json
>>> ec2 = boto3.client("ec2")
```

Import the boto3 module and create an ec2 object. The json module will be used to format the json response for better view when aws responses are printed to the terminal..

[1] Create a security group

```
>>> security_group =
ec2.create_security_group(GroupName="23805764_2-sg", Description="second
security group for second ec2 instance")
>>> print(json.dumps(security_group, indent=2))
```

Create a new security group added _2 to the end of my student id as the name. Then print the stored response.

```
{
  "GroupId": "sg-0c064fff08dc7836f",
  "ResponseMetadata": {
    "RequestId": "72b08287-b24a-407b-b795-3ceee34aa9a9",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "x-amzn-requestid": "72b08287-b24a-407b-b795-3ceee34aa9a9",
      "cache-control": "no-cache, no-store",
      "strict-transport-security": "max-age=31536000;
includeSubDomains",
      "content-type": "text/xml;charset=UTF-8",
      "content-length": "283",
      "date": "Fri, 11 Aug 2023 08:30:23 GMT",
      "server": "AmazonEC2"
    },
    "RetryAttempts": 0
  }
}
```

The above is the output after printing the response, with the GroupId as the first key value pair.

[2] Authorise inbound traffic for ssh

```
>> asgi_response =
ec2.authorize_security_group_ingress(Group_name="23805764_2-sg",
IpProtocol="tcp", FromPort=22, ToPort=22, CidrIp="0.0.0.0/0")
>>> print(json.dumps(asgi_response, indent=2))
```

The command for authorising incoming traffic is almost the same as the using AWS cli, except a FromPort and ToPort is to be specified instead of just --port.

```
{
  "Return": true,
  "SecurityGroupRules": [
    {
      "SecurityGroupRuleId": "sgr-023c529acddfa5163",
      "GroupId": "sg-0c064fff08dc7836f",
      "GroupOwnerId": "489389878001",
      "IsEgress": false,
      "IpProtocol": "tcp",
      "FromPort": 22,
      "ToPort": 22,
      "CidrIpv4": "0.0.0.0/0"
    }
  ]
}
```



```

    ],
    "ResponseMetadata": {
        "RequestId": "ffa5ca5e-316b-40a3-9df2-26b6e0419b05",
        "HTTPStatusCode": 200,
        "HTTPHeaders": {
            "x-amzn-requestid": "ffa5ca5e-316b-40a3-9df2-26b6e0419b05",
            "cache-control": "no-cache, no-store",
            "strict-transport-security": "max-age=31536000;
includeSubDomains",
            "content-type": "text/xml; charset=UTF-8",
            "content-length": "719",
            "date": "Fri, 11 Aug 2023 08:40:14 GMT",
            "server": "AmazonEC2"
        },
        "RetryAttempts": 0
    }
}

```

The above is the output. The “Return” value is true, which shows that the command was successful.

[3] Create a key pair to be used for ssh connection to the EC2 instance

```
>>> kp_response = ec2.create_key_pair(KeyName="23805764_2-key")
```

The above command will create a key pair and store the response in kp_response.

```

>>> keyFile = open("23805764_2-key.pem", "w")
>>> keyFile.write(kp_response["KeyMaterial"])
1678
>>> keyFile.close()

```

The above commands create a file for write and and read. Then the private key is outputted to a file named “23805764_2-key.pem”.

```

>>> import os, stat
>>> os.chmod("23805764_2-key.pem", stat.S_IREAD)

```

Using the os module and the command chmod, the private key file’s permission can be updated. The stat module is also required to retrieve the permission type.

```

>>> keyFile = open("23805764_2-key.pem", "r")
>>> print(keyFile.read())

```

To verify that the private key file is stored correctly. Open the key file in read mode and print the content in python.

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAIh5l10x9ilaEOJ+P7o/uIvDRQacbSKdf6JbuBxr35HF0+14z
XFWWF166GJ9Nj8Dq6A/a+J8tipDgdXHcFppoCyT+21KoqN7czQ24PkgUi+Ag3tHy
Y7TJNsNVkq1HFCyD3oxAlQEbgPEaqYLkaobfKKPvneOBu7LFA8pfIDua397EMRRs
wirMrmWryE9lU6x+4qV7j70N1vFBuUnrkLDj04yacxuIvgiOQPY3u+3ppsMjHbqF
Tooq02yjS7aHizhjAZSOFi6P0h+006Yl79EJ+gxGD83btJNwEVcd35fCi4jzcFL1
Bg0k+POqV0cSKfTqK/r52TX6t0iQiBErOak8FQIDAQABAOIBAFSo+yUErJLeM09L
XuMg0BwUtn9ima04DF66HN64FhitvsE0epDvWRbxv1oWU2l5G8Pw7AyXLUGRM7Rp
fTlRndKu855SmHwGb0YXTEgsDDa0LTbqmv8eD6g4qP0qtfgcMniwMlg1/Zweryeb7
d10gnKsBiX+jE0Nk+cQwCIzuXck6MTvQIo7QEDE64Qy3MLrqI12IA3GpPv9WYZQD
wDtBKAZ4eqwJWSiAIck/wlFgX89bnKBE71hoDf/3UvfOX6uJdquSzoCtjm8kglsk
Z9uAL+m7R/O+Ku7J0kVqzKmk5QB39D6DD2SfMcrTHW3F/UVSgnkV0m/xnjJ/jvdz
d90+UIUCgYEAwJU7TdLiANnWc+I1t4P//WiBfje9neZqtg26RTqZK1N+SQ7UGu3N
+plsseiJ7J/EYvVjDVV2tU35YjAI0+iksIQa6IgmCyABVNCEm02LTNW+hcJnnbbC
8h7WN0Vtpp4yFBQnKLsLS5SY0BEoRt/QI+U7YhX1ApvuLzNPpXw/p2MCgYEAt5nT
W4VVVNYZ6Ci7eVmbTCvN4zEoZ0z96gqyp5m3li99BDXI8aSf1XMoXJuWwvXX5rO
OJZcBeb+9VQu/tpwjniy0/8SpeLL6bR5P1B4r7cFv1cD3jPuTAggHHTchRTm1bv1
QrUL0EPbbdLS4XigwF1zN+izItunkdPjwT6BFCCgYEAldwxjsLHQuuTZyYMA0e
mlk01LFaHy7ImSWC4y/lVzkPXLQoW0Ylt3AC8jLv14ZkOUc/+wD/K4i0/xW0lpto
LJYZMEM1spvS73Tnyf0yPvuRgszG/cSKdd8BGC92oqiF1YGIhETLwuZGLfomoA59
60io/W9Cy10HGjUfTSR3twkCgYA/hg51x00gPC+cfrNWqiUsNp+2sqRrEV1s46Sr
mcC4GAko78awDx1SbrctSN1bPQ+4eWR0ed9+/0xJb4YPEV80PD6LUN6P3Kd2xcWJ
ROhsy2NGr4GcGNWjItCRajE5rmCrpKuYF/HtbFz2w5HJByZ/SBX8SxvvnJ3+u5Q6
oTih0wKBgQCYADJSQadBa0UbTd5w7U0HKdr0wkQ3aPvR2r0DHvOTN8A+8wE4W8gW
UuwP9wDwnpf+z/VwaykGU9XVgc9+7mrNz3hCC1/yHvg1NXPnapoWuCnubfXEmuvq
5+Z0D3Y5EhwU1arBPoII3H7LOTst1bYuwAukOqM2m5W0+oPB4xNQ8w==
-----END RSA PRIVATE KEY-----
```

The above output verifies that the private key is successfully stored locally.

```
>>> keyFile.close()
```

Close the file.

[4] Create the second ec2 instance

```
>>> rs_response = ec2.run_instances(ImageId="ami-d38a4ab1",
SecurityGroups=["23805764_2-sg"], MinCount=1, MaxCount=1,
InstanceType="t2.micro", KeyName="23805764_2-key")
```

The above python command will create and run a new ec2 instance with the parameters provided. All parameters are similar to the usage with AWS cli except a MinCount and MaxCount is specified instead of --count.

```
>>> print(rs_response["Instances"][0]["InstanceId"])
i-0b3b1a0e8bb0dab79
```

Using the same key used earlier as path to access the instance id.

[Optional] Add tag to the second ec2 instance

```
>>> ct_response = ec2.create_tags(Resources=["i-0b3b1a0e8bb0dab79"],
Tags=[{"Key": "Name", "Value": "23805764_2"}])
```

The above python command will create a new tag for my second ec2 instance.

Instance ID = i-0b3b1a0e8bb0dab79 X		Name = 23805764_2 X		Clear filters		< 1 >		⚙	
<input type="checkbox"/>	Name ▲	Instance ID	Instance state ▼	Instance type ▼	Status check	Alarm status	Availab		
<input type="checkbox"/>	23805764_2	i-0b3b1a0e8bb0dab79	🟢 Running 🔍	t2.micro	🟢 2/2 checks passed	No alarms +	ap-sout		

This screenshot taken from the AWS console instances section shows that my second ec2 instance is successfully created with the correct tag, but not running.

[5] Get the public IP address

```
>>> di_response =
ec2.describe_instances(InstanceIds=["i-0b3b1a0e8bb0dab79"])
```

The above python command will retrieve information of the second instance.

```
>>>
print(di_response["Reservations"][0]["Instances"][0]["PublicIpAddress"])
3.25.88.59
```

The boto commands in python do not have --query parameters. Instead using dictionary key access to find and print the public address.

[6] Connecting to the second ec2 instance

In a new terminal.

```
ssh -i 23805764_2-key.pem ubuntu@3.25.88.59
```

Performs a ssh connection with the second security key pair and the second ec2 ip address.

```
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. In the AWS console  
  
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by  
applicable law. the IP address  
  
To run a command as administrator (user "root"), use "sudo <command>".  
See "man sudo_root" for details.  
  
ubuntu@ip-172-31-0-217:~$
```

The above screenshot shows that the second ec2 instance is successfully connected through ssh.

[7] Instance in the AWS console

Name: 23805764		Clear filters		< 1 >				
<input type="checkbox"/>	Name ▲	Instance ID	Instance state ▼	Instance type ▼	Status check	Alarm status	Availab	
<input type="checkbox"/>	23805764	i-00dce50ce3eb05af3	Running	t2.micro	2/2 checks passed	No alarms	+	ap-sout
<input type="checkbox"/>	23805764_2	i-0b3b1a0e8bb0dab79	Running	t2.micro	2/2 checks passed	No alarms	+	ap-sout

The above screen shot shows both of the ec2 instances I have created.

Terminate both ec2 instances

```
aws ec2 terminate-instances --instance-ids i-00dce50ce3eb05af3  
i-0b3b1a0e8bb0dab79
```

The above command used to terminate the two ec2 instances I created.

```
{  
  "TerminatingInstances": [  
    {  
      "CurrentState": {  
        "Code": 32,  
        "Name": "shutting-down"  
      },  
      "InstanceId": "i-00dce50ce3eb05af3",  
      "PreviousState": {  
        "Code": 16,  
        "Name": "running"  
      }  
    },  
    {
```

```

    "CurrentState": {
      "Code": 32,
      "Name": "shutting-down"
    },
    "InstanceId": "i-0b3b1a0e8bb0dab79",
    "PreviousState": {
      "Code": 16,
      "Name": "running"
    }
  }
}
]
}

```

The above is the output of the command, showing both instances are shutting down.

Name: 23805764 X		Clear filters		< 1 >		⚙	
<input type="checkbox"/>	Name ▲	Instance ID	Instance state ▼	Instance type ▼	Status check	Alarm status	Availab
<input type="checkbox"/>	23805764	i-00dce50ce3eb05af3	Terminated 🔍	t2.micro	–	No alarms +	ap-sout
<input type="checkbox"/>	23805764_2	i-0b3b1a0e8bb0dab79	Terminated 🔍	t2.micro	–	No alarms +	ap-sout

The above screenshot taken from the instances section in AWS dashboard shows that both of my ec2 instances are indeed terminated.

Using Docker

[1] Install Docker

```
sudo apt install docker.io -y
```

The above command tries to install docker.

```

[sudo] password for taohu:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  cgroupfs-mount containerd criu libintl-perl libintl-xs-perl
  libmodule-find-perl libmodule-scandeps-perl
  libproc-processtable-perl libsort-naturally-perl needrestart runc tini
Suggested packages:
  containernetworking-plugins docker-doc aufs-tools btrfs-progs
  debootstrap rinse rootlesskit xfsprogs
  zfs-fuse | zfsutils-linux iucode-tool

```

```

The following NEW packages will be installed:
  cgroupfs-mount containerd criu docker.io libintl-perl libintl-xs-perl
  libmodule-find-perl
  libmodule-scandeps-perl libproc-processtable-perl
  libsort-naturally-perl needrestart runc tini
0 upgraded, 13 newly installed, 0 to remove and 822 not upgraded.
Need to get 47.2 MB of archives.
After this operation, 222 MB of additional disk space will be used.
Err:4 http://http.kali.org/kali kali-rolling/main arm64 docker.io arm64
20.10.25+dfsg1-1
  404 Not Found [IP: 192.99.200.113 80]
Get:1 http://http.kali.org/kali kali-rolling/main arm64 runc arm64
1.1.5+ds1-1+b1 [2,335 kB]
Get:2 http://http.kali.org/kali kali-rolling/main arm64 containerd arm64
1.6.20~ds1-1+b1 [15.4 MB]
Get:3 http://kali.download/kali kali-rolling/main arm64 tini arm64
0.19.0-1 [209 kB]
Get:5 http://kali.download/kali kali-rolling/main arm64 cgroupfs-mount
all 1.4 [6,276 B]
Get:6 http://kali.download/kali kali-rolling/main arm64 criu arm64
3.17.1-2 [607 kB]
Get:7 http://kali.download/kali kali-rolling/main arm64 libintl-perl all
1.33-1 [720 kB]
Get:8 http://kali.download/kali kali-rolling/main arm64 libintl-xs-perl
arm64 1.33-1 [15.2 kB]
Get:9 http://kali.download/kali kali-rolling/main arm64
libmodule-find-perl all 0.16-2 [10.6 kB]
Get:10 http://kali.download/kali kali-rolling/main arm64
libmodule-scandeps-perl all 1.31-2 [41.7 kB]
Get:11 http://kali.download/kali kali-rolling/main arm64
libproc-processtable-perl arm64 0.636-1 [42.3 kB]
Get:12 http://kali.download/kali kali-rolling/main arm64
libsort-naturally-perl all 1.03-4 [13.1 kB]
Get:13 http://kali.download/kali kali-rolling/main arm64 needrestart all
3.6-5 [59.2 kB]
Fetched 19.5 MB in 7s (2,781 kB/s)

E: Failed to fetch
http://http.kali.org/kali/pool/main/d/docker.io/docker.io_20.10.25%2bdfsg1-1_arm64.deb 404 Not Found [IP: 192.99.200.113 80]
E: Unable to fetch some archives, maybe run apt-get update or try with
--fix-missing?

```

Output shows that there has been an error when installing docker.

```
sudo apt-get update
```

From the error message, an update is performed with the above command.

```
sudo apt install docker.io -y
```

Attempt to install docker again.

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  cgrouperfs-mount containerd criu libintl-perl libintl-xs-perl
  libmodule-find-perl libmodule-scandeps-perl
  libproc-processtable-perl libsort-naturally-perl needrestart runc tini
Suggested packages:
  containernetworking-plugins docker-doc aufs-tools btrfs-progs
  debootstrap rinse rootlesskit xfsprogs
  zfs-fuse | zfsutils-linux iucode-tool
The following NEW packages will be installed:
  cgrouperfs-mount containerd criu docker.io libintl-perl libintl-xs-perl
  libmodule-find-perl
  libmodule-scandeps-perl libproc-processtable-perl
  libsort-naturally-perl needrestart runc tini
0 upgraded, 13 newly installed, 0 to remove and 944 not upgraded.
Need to get 45.6 MB/47.4 MB of archives.
After this operation, 222 MB of additional disk space will be used.
Get:1 http://http.kali.org/kali kali-rolling/main arm64 runc arm64
1.1.5+ds1-1+b2 [2,321 kB]
Get:2 http://http.kali.org/kali kali-rolling/main arm64 containerd arm64
1.6.20~ds1-1+b2 [15.6 MB]
Get:3 http://http.kali.org/kali kali-rolling/main arm64 docker.io arm64
20.10.25+dfsg1-1+b1 [27.8 MB]
Fetched 45.6 MB in 10s (4,348 kB/s)

Selecting previously unselected package runc.
(Reading database ... 393840 files and directories currently installed.)
Preparing to unpack .../00-runc_1.1.5+ds1-1+b2_arm64.deb ...
Unpacking runc (1.1.5+ds1-1+b2) ...
Selecting previously unselected package containerd.
Preparing to unpack .../01-containerd_1.6.20~ds1-1+b2_arm64.deb ...
Unpacking containerd (1.6.20~ds1-1+b2) ...
Selecting previously unselected package tini.
Preparing to unpack .../02-tini_0.19.0-1_arm64.deb ...
Unpacking tini (0.19.0-1) ...
```

```

Selecting previously unselected package docker.io.
Preparing to unpack .../03-docker.io_20.10.25+dfsg1-1+b1_arm64.deb ...
Unpacking docker.io (20.10.25+dfsg1-1+b1) ...
Selecting previously unselected package cgroupfs-mount.
Preparing to unpack .../04-cgroupfs-mount_1.4_all.deb ...
Unpacking cgroupfs-mount (1.4) ...
Selecting previously unselected package criu.
Preparing to unpack .../05-criu_3.17.1-2_arm64.deb ...
Unpacking criu (3.17.1-2) ...
Selecting previously unselected package libintl-perl.
Preparing to unpack .../06-libintl-perl_1.33-1_all.deb ...
Unpacking libintl-perl (1.33-1) ...
Selecting previously unselected package libintl-xs-perl.
Preparing to unpack .../07-libintl-xs-perl_1.33-1_arm64.deb ...
Unpacking libintl-xs-perl (1.33-1) ...
Selecting previously unselected package libmodule-find-perl.
Preparing to unpack .../08-libmodule-find-perl_0.16-2_all.deb ...
Unpacking libmodule-find-perl (0.16-2) ...
Selecting previously unselected package libmodule-scandeps-perl.
Preparing to unpack .../09-libmodule-scandeps-perl_1.31-2_all.deb ...
Unpacking libmodule-scandeps-perl (1.31-2) ...
Selecting previously unselected package libproc-processtable-perl:arm64.
Preparing to unpack .../10-libproc-processtable-perl_0.636-1_arm64.deb
...
Unpacking libproc-processtable-perl:arm64 (0.636-1) ...
Selecting previously unselected package libsort-naturally-perl.
Preparing to unpack .../11-libsort-naturally-perl_1.03-4_all.deb ...
Unpacking libsort-naturally-perl (1.03-4) ...
Selecting previously unselected package needrestart.
Preparing to unpack .../12-needrestart_3.6-5_all.deb ...
Unpacking needrestart (3.6-5) ...
Setting up runc (1.1.5+ds1-1+b2) ...
Setting up libmodule-find-perl (0.16-2) ...
Setting up tini (0.19.0-1) ...
Setting up libproc-processtable-perl:arm64 (0.636-1) ...
Setting up criu (3.17.1-2) ...
Setting up libintl-perl (1.33-1) ...
Setting up cgroupfs-mount (1.4) ...
update-rc.d: We have no instructions for the cgroupfs-mount init script.
update-rc.d: It looks like a non-network service, we enable it.
Setting up containerd (1.6.20~ds1-1+b2) ...
containerd.service is a disabled or a static unit, not starting it.
Setting up libsort-naturally-perl (1.03-4) ...
Setting up libmodule-scandeps-perl (1.31-2) ...
Setting up needrestart (3.6-5) ...
Setting up docker.io (20.10.25+dfsg1-1+b1) ...

```



```
Adding group `docker' (GID 142) ...
Done.
update-rc.d: We have no instructions for the docker init script.
update-rc.d: It looks like a non-network service, we enable it.
Created symlink
/etc/systemd/system/multi-user.target.wants/docker.service →
/lib/systemd/system/docker.service.
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket →
/lib/systemd/system/docker.socket.
Setting up libintl-xs-perl (1.33-1) ...
Processing triggers for libc-bin (2.36-9) ...
Processing triggers for man-db (2.11.2-2) ...
Processing triggers for kali-menu (2023.2.3) ...
```

The above output showed no error message, meaning that docker has been installed successfully.

```
sudo systemctl start docker
sudo systemctl enable docker
```

The above command will start docker using the linux command systemctl.

```
Synchronizing state of docker.service with SysV service script with
/lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable docker
```

The output above showed no error, meaning docker has successfully started.

[2] Check the version

```
docker --version
```

To display the version of the installed docker.

```
Docker version 20.10.25+dfsg1, build b82b9f3
```

Output showing the docker version.

[3] Build and run an httpd container

Currently inside lab2 directory I created earlier.

```
mkdir html
```

```
cd html
touch index.html
```

The above commands create a directory called html. Then the current directory is set to the newly created html directory. Finally, a file named index.html is created.

```
code . > /dev/null &
```

I have installed VSCode already. The above command opens the current directory with VSCode.

```
<html>
  <head></head>
  <body>
    <p>Hello World</p>
  </body>
</html>
```

The above is the html code I placed into index.html with VSCode.

[4] Create “Dockerfile”

Currently inside the html directory.

```
cd ..
touch Dockerfile
code . > /dev/null &
```

The above commands will change directory to the parent directory, which is outside of the html directory. Then a file named “Dockerfile” is created. Finally, the current directory is opened using VSCode for editing.

```
FROM httpd:2.4
COPY ./html/ /usr/local/apache2/htdocs/
```

The above is the code I put into Dockerfile with VSCode. From docker’s documentation, FROM states the parent image to be used, which in this case is using httpd(an apache http server) version 2.4. On the second line, COPY will copy the html directory to the destination specified.

[5] Build the docker image

```
docker build -t my-apache2 .
```

The above command is used to build the docker image using the Dockerfile specified in the path, which here is the current directory. The -t option specifies the tag to be my-apache2.

```
permission denied while trying to connect to the Docker daemon socket at
unix:///var/run/docker.sock: Post
"http://%2Fvar%2Frun%2Fdocker.sock/v1.24/build?buildargs=%7B%7D&cachefro
m=%5B%5D&cgroupparent=&cpuperiod=0&cpuquota=0&cpusetcpus=&cpusetmems=&cp
ushares=0&dockerfile=Dockerfile&labels=%7B%7D&memory=0&memswap=0&network
mode=default&rm=1&shmsize=0&t=my-apache2&target=&ulimits=null&version=1"
: dial unix /var/run/docker.sock: connect: permission denied
```

The above is the output. It shows an error of permission denied, which means sudo is required.

From the lab specification the solution is to add the current user to the docker group, then logout and login again.

```
sudo usermod -a -G docker taohu
```

This adds my current logged in user to the docker group. Then I logged out and logged in again.

```
docker build -t my-apache2 .
```

Perform docker build again.

```
Sending build context to Docker daemon 494.6kB
Step 1/2 : FROM httpd:2.4
2.4: Pulling from library/httpd
90524f7dc01b: Pull complete
3398ac8b1a30: Pull complete
d17dcb45d570: Pull complete
b4d7481a64dc: Pull complete
c29613c36e2c: Pull complete
Digest:
sha256:d7262c0f29a26349d6af45199b2770d499c74d45cee5c47995a1ebb336093088
```

The output above shows that the build has succeeded.

[6] Run the image

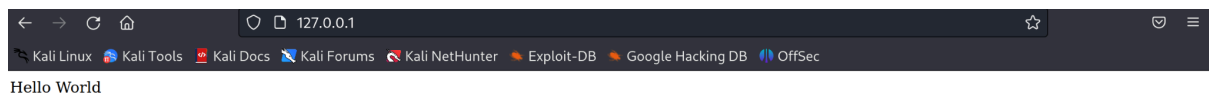
```
docker run -p 80:80 -dit --name my-app my-apache2
```

The above command will run a docker container using the image my-apache2 which was created earlier. The -p configures the port mapping from host to docker container to 80. The -dit will tell docker to run in background, not blocking the terminal input and outputting the container id.

```
5702d31d2f8158bb417bee940a81a821bf9cee39f97a355cb1cd65c887102f66
```

The above is the output, which contains the container id of the running container.

[7] View in browser



The above screenshot shows the top part of the browser, which contains all the content displayed. It can be seen that the url is 127.0.0.1 and the content only contains the world “Hello World” at the top left corner.

[8] Other commands

```
docker ps -a
```

The above command will display all the containers from docker.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAM
ES5702d31d2f81	my-apache2	"httpd-foreground"	17 minutes ago	Up 17 minutes	0.0.0.0:80→80/tcp, :::80→80/tcp	my-app

The above is a screenshot of the output. It can be seen that there is only 1 docker container with name of my-app and the image of my-apache2.

```
docker stop my-app
```

The above command will attempt to stop the container named my-app.

```
my-app
```

The output is the name of the container, which suggests there are no errors when stopping.

```
docker rm my-app
```

The above command will attempt to remove the docker container named my-app.

```
my-app
```

The output suggests there was no error when removing the container my-app.

Lab 3 - DynamoDB

This lab involves creating and configuring S3 buckets, operations on DynamoDB and start an application as its own personal cloud storage.

Program

[1] Preparation

```
pwd
/home/taohu/Documents/CC/labs/lab3 # The output
```

The above command and the output shows my current directory.

The first step of this lab is to retrieve the cloudstorage.py file from the link provided in the lab specification.

```
wget
https://raw.githubusercontent.com/zhangzhics/CITS5503_Sem2_2023/master/L
abs/src/cloudstorage.py
```

Use the above command to retrieve the cloudstorage.py file.

```
ls
cloudstorage.py # The output
```

The above command and output verifies that the file has been downloaded.

The next step is to create a directory called rootdir and a file called rootfile.txt inside with 5 lines containing the line number.

```
mkdir rootdir;
echo -e "1\n2\n3\n4\n5\n" > rootdir/rootfile.txt;
```

The above commands consist of two commands separated by semicolon. Firstly a directory called rootdir is created then a file called rootfile.txt is created and stored inside the newly created directory.

The echo command is used to create the new files with -e option, which enables special characters starting with forward slash, in this case is the “\n” (new line). rootfile.txt contains 5 lines, each line containing the line number from 1 to 5.

```
cat rootdir/rootfile.txt
```

Use the cat command to output the content of rootfile.txt to ensure the file is created and has the correct content.

```
1
2
3
4
5
```

The above content shows that it is correct.

Following the lab specification, a duplicate of the previous step is performed, but the directory and field are placed inside the directory rootdir created previously.

```
cd rootdir
pwd
/home/taohu/Documents/CC/labs/lab3/rootdir      # output
```

The first step is to move the current working directory into rootdir. The above command is run to do so. Then the next step is to repeat what was done earlier but with the directory name of subdir and filename of subfile.txt. The last command pwd is ran to verify that the current working directory is the correct one.

```
mkdir subdir;
echo -e "1\n2\n3\n4\n5\n" > subdir/subfile.txt;
```

Above creates a directory named subdir and place file named subfile.txt inside subdir.

```
cat subdir/subfile.txt
```

Output the content of subfile.txt inside subidr.

```
1
2
3
4
5
```

Output shows that the file is created with the correct content.

```
ls
rootfile.txt  subdir    # output
```

Run ls command to verify again and to show that rootdir has the correct contents.

[2] Save to S3

Continuing from the above, the first step is to change to the correct directory which is the directory lab3.

```
cd ..
pwd
/home/taohu/Documents/CC/labs/lab3 # output
```

First move back to the lab3 directory.

```
code . > /dev/null &
```

Above command used to open the current directory with vscode.

The first change to the cloudstorage.py is to edit the python file to take one argument of “-i” or “--initialise=”, which if true will create a new S3 bucket. After some research, I decided to use the getopt module to read the python file arguments.

```
import base64
import sys
import getopt
```

The first step is to import the getopt module, placed below the import of base64. The sys module is also imported to access the argv variable.

```
# Main program

opt_intialise = False

opts, args = getopt.getopt(sys.argv[1:], "i", ["initialise="])

for opt, arg in opts:
    if opt in ("-i", "--initialise"):
        opt_intialise = True
        if (opt == "--initialise" and arg != "True"):
```

```
opt_intialise = False
```

Then create a boolean called `opt_intialise` which will store the result of the argument read with a default value of false. `getopt.getopt` is called with `argv`, which is the python file arguments and the options specified is “i” and long options specified is “initialise=”. Loop through the returned variable `opts` to check each of the input arguments. `op_intialise` is true if either “-i” is found or if “--initialise=True” is found.

After reading the python file arguments, if the `op_intialise` is true then a S3 bucket should be created with the configurations defined.

```
ROOT_S3_DIR = '23805764-cloudstorage'
```

The above code shows the changed value of the variable `ROOT_S3_DIR` which will be used as the bucket name. It has being changed to contain my student id.

```
# Insert code to create bucket if not there

try:
    if (opt_intialise):
        response = s3.create_bucket(
            Bucket=ROOT_S3_DIR,
            CreateBucketConfiguration=bucket_config
        )
        print(response)
except Exception as error:
    pass
```

The above code is placed after where the `getopt` function has finished executing, the comments included is the indicator of the position in file. The function `create_bucket` is called, which will attempt to create a new s3 bucket. From the boto3 S3 documentation, the `Bucket` argument is the only required argument, which specifies the name of the S3 bucket. The argument `CreateBucketConfiguration` is required here, which uses the value of the variable `bucket_config`. `bucket_config` came with the file `cloudstorage.py`, which contains a value pair to specify the AWS server location for the S3 bucket

Next the `upload_file` function has to be completed. The purpose of this function is to take the path and name of a local file and upload it to the AWS S3 buckets in the cloud.

```
def upload_file(folder_name, file, file_name):
    s3.upload_file(file, ROOT_S3_DIR, f"{folder_name}{file_name}")
    print("Uploading %s" % file)
```

The above is the completed `upload_file` function. The `upload_file` function of s3 client created is used to upload files to the AWS S3 bucket. The function takes 3 parameters, the first is the file location of

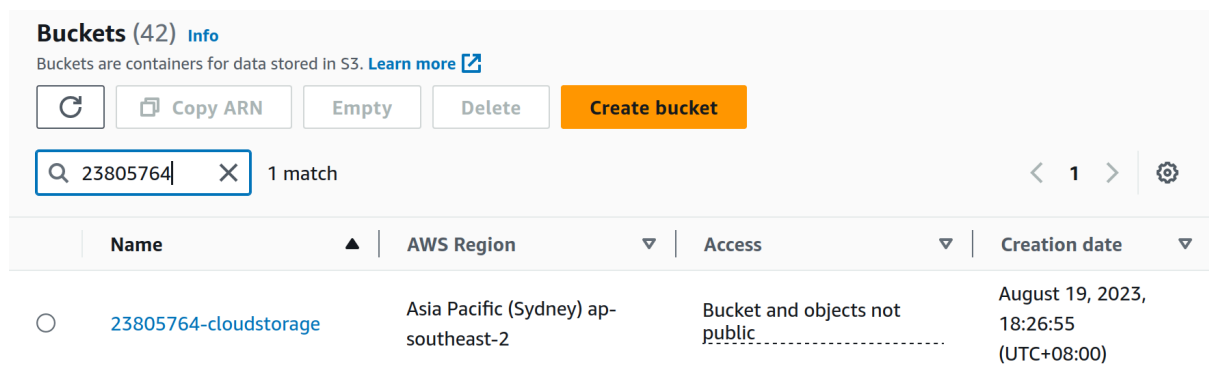
the local file to be uploaded, the second specifies the bucket name and the third is the location in the S3 bucket.

```
python cloudstorage.py -i
```

The above command will execute the python file and -i is included to tell the cloudstoreg.py to create the S3 bucket.

```
{'ResponseMetadata': {'RequestId': '9YQMCWX5PPKCCRGX', 'HostId':  
'cMYP8d+/3bzLkk9hkHndTsloFysEM9KKVqQksdFybJVQ3ubHxypmgXuxRtz6mwH7fFrejN  
xsOM=', 'HTTPStatusCode': 200, 'HTTPHeaders': {'x-amz-id-2':  
'cMYP8d+/3bzLkk9hkHndTsloFysEM9KKVqQksdFybJVQ3ubHxypmgXuxRtz6mwH7fFrejN  
xsOM=', 'x-amz-request-id': '9YQMCWX5PPKCCRGX', 'date': 'Sat, 19 Aug  
2023 10:26:55 GMT', 'location':  
'http://23805764-cloudstorage.s3.amazonaws.com/', 'server': 'AmazonS3',  
'content-length': '0'}}, 'RetryAttempts': 0}, 'Location':  
'http://23805764-cloudstorage.s3.amazonaws.com/'}  
Uploading ./rootdir/rootfile.txt  
Uploading ./rootdir/subdir/subfile.txt  
done
```

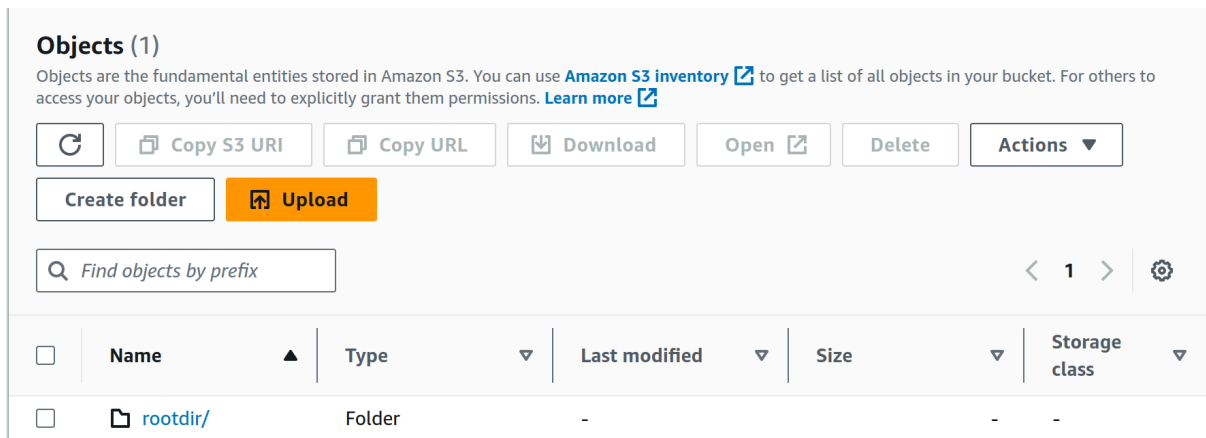
The above is the output, which shows that there was no error in creating and uploading files. The first part of the output wrapped in curly brackets is the response from create_bucket, which contains data of the newly created S3 bucket.



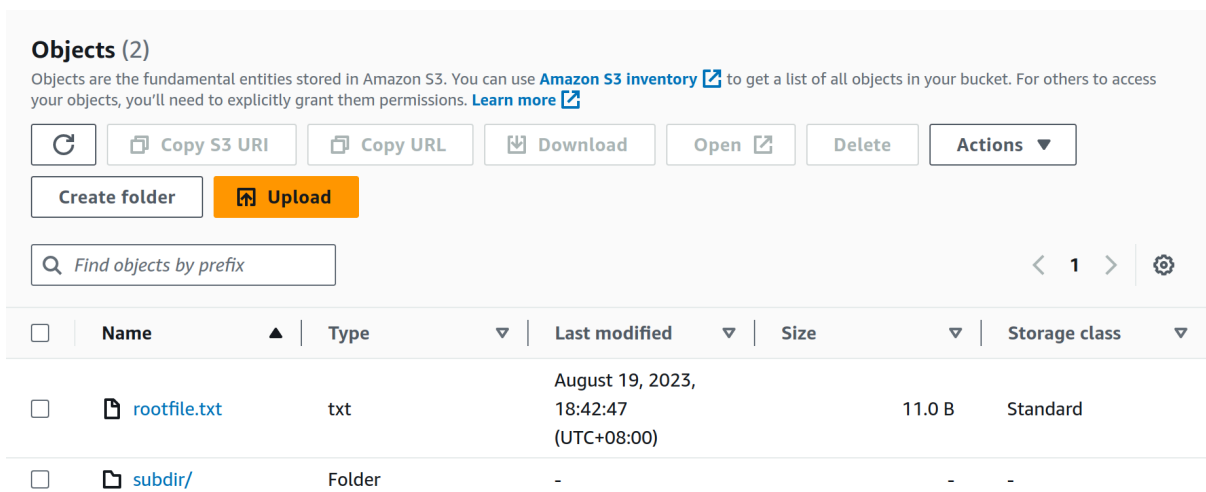
The screenshot shows the AWS console 'Buckets' page. At the top, there's a header 'Buckets (42) Info' with a link to 'Learn more'. Below this are buttons for 'Refresh', 'Copy ARN', 'Empty', 'Delete', and 'Create bucket'. A search bar contains '23805764' and shows '1 match'. Below the search bar is a table with columns: Name, AWS Region, Access, and Creation date. The table lists one bucket: '23805764-cloudstorage' in the 'Asia Pacific (Sydney) ap-southeast-2' region, with 'Bucket and objects not public' access and a creation date of 'August 19, 2023, 18:26:55 (UTC+08:00)'.

Name	AWS Region	Access	Creation date
23805764-cloudstorage	Asia Pacific (Sydney) ap-southeast-2	Bucket and objects not public	August 19, 2023, 18:26:55 (UTC+08:00)

The above is a screenshot taken from the bucket section in AWS console. It can be seen that my new bucket has been successfully created.



The above screenshot is taken of the content inside the bucket. It can be seen that the bucket contains a directory called rootdir. This shows that the correct folder structure has been uploaded to the bucket.



The above screenshot is taken inside the rootdir. It can be seen that the correct files and directories are present.

The complete code of cloudstorage.py.

```
import os
import boto3
import base64
import sys
import getopt

# -----
# CITS5503
#
# cloudstorage.py
#
```

```

# skeleton application to copy local files to S3
#
# Given a root local directory, will return files in each level and
# copy to same path on S3
#
# -----

ROOT_DIR = '.'
ROOT_S3_DIR = '23805764-cloudstorage'

s3 = boto3.client("s3")

bucket_config = {'LocationConstraint': 'ap-southeast-2'}

def upload_file(folder_name, file, file_name):
    s3.upload_file(file, ROOT_S3_DIR, f"{folder_name}{file_name}")
    print("Uploading %s" % file)

# Main program

opt_intialise = False

opts, args = getopt.getopt(sys.argv[1:], "i", ["initialise="])

for opt, arg in opts:
    if opt in ("-i", "--initialise"):
        opt_intialise = True
        if (opt == "--initialise" and arg != "True"):
            opt_intialise = False

# Insert code to create bucket if not there

try:
    if (opt_intialise):
        response = s3.create_bucket(
            Bucket=ROOT_S3_DIR,
            CreateBucketConfiguration=bucket_config
        )
        print(response)
except Exception as error:
    pass

```

```
# parse directory and upload files

for dir_name, subdir_list, file_list in os.walk(ROOT_DIR,
topdown=True):
    if dir_name != ROOT_DIR and not "ss" in dir_name:
        for fname in file_list:
            upload_file("%s/" % dir_name[2:], "%s/%s" % (dir_name,
fname), fname)

print("done")
```

From the above it can be seen that there is also a change in the parse directory and upload files section. I added an extra comparison which excludes a directory called “ss” from uploading. This directory called “ss” is located in my lab3 folder, the same folder cloudstorage.py is in. This directory contains screenshots used for lab3’s report.

[3] Restore from S3

Here, a new python file will be created to read from the S3 bucket created earlier and store what is read in a folder locally.

```
import boto3
import os
import sys

ROOT_S3_DIR = '23805764-cloudstorage'

try:
    out_dir = sys.argv[1]
except:
    print("Error, need to specify the output directory name.")
    print("Usage: python restorefromcloud.py <out_dir_name>")
    exit(0)

print(f"Restoring contents from s3 bucket: {ROOT_S3_DIR} into\n{out_dir}\n")

s3 = boto3.client("s3")
lo_response = s3.list_objects(Bucket=ROOT_S3_DIR)
```

```

for content in lo_response["Contents"]:
    file_key = content["Key"]
    try:
        local_path_file = os.path.join(out_dir, file_key)
        local_path_dir = os.path.dirname(local_path_file)
        if not os.path.exists(local_path_dir):
            os.makedirs(local_path_dir)
        s3.download_file(Bucket=ROOT_S3_DIR, Key=file_key,
Filename=local_path_file)
        print(f"Restored {file_key} from bucket into {local_path_file}
locally.")
    except:
        print(f"Failed to restore {file_key}.")

print("\nDone")

```

The above code is written after some research and trials. The code python file works by having a required file argument representing the directory where the contents from the bucket will be saved into.

The function `list_objects` was found after some research, which will retrieve data of all objects inside the bucket specified by the `Bucket` parameter in the function call. The response will be a dict data type in python containing some data about the bucket. There is a key called `Contents` inside the response, which contains a list of dict, where each of these dict contains the data of each object inside the s3 bucket. In the inner dict, there is a key with name `Key` which contains the object location.

The code starts by checking that the user has provided an output directory name. Then, `list_objects` is called to retrieve data of all objects inside the S3 bucket. Once the response has been received, the value under the key `Contents` is loop. In each loop, the object path in the bucket can be found by accessing the key name `Key` of the dict content. The remote path will also be used locally as a duplicate. A validation is done on the directory path of where the file will be placed locally. If the directory path does not exist then, the directory path will be created using python command `os.makedirs`. Once the directory path does exist the `download_file` function can be called to download the file and place it locally. The `download_file` function takes 3 parameters, the first is the bucket name, the second is the object path in the S3 bucket and the third parameter is the file path to be stored locally.

```
python restorefromcloud.py fromCloud
```

The above command will run the python file and specify the output directory of name `fromCloud`.

```
Restoring contents from s3 bucket: 23805764-cloudstorage into fromCloud
```

```
Restored rootdir/rootfile.txt from bucket into  
fromCloud/rootdir/rootfile.txt locally.  
Restored rootdir/subdir/subfile.txt from bucket into  
fromCloud/rootdir/subdir/subfile.txt locally.  
  
Done
```

The above is the output. It can be seen that 2 files are downloaded from the s3 bucket specified.

```
ls fromCloud  
rootdir          # output
```

The above lists contents of fromCloud directory to verify that the download worked. It can be seen that there is a directory called rootDir inside.

```
ls fromCloud/rootdir  
rootfile.txt  subdir    # output
```

The above shows that there are contents inside the rootdir and it is correct in comparing the structure in the S3 bucket.

```
cat fromCloud/rootdir/rootfile.txt
```

Finally the above command is used to verify that file content is also correct by printing it in terminal.

```
1  
2  
3  
4  
5
```

The above response shows that the file does contain the correct contents.

[4] Write information about files to DynamoDB

The first step is to install DynamoDB on the virtual machine. From the lab specification, a directory called dynamodb is created and then installed inside.

```
cd ~;  
mkdir dynamodb;  
cd dynamodb;
```

The above command first changes my current directory to my home directory, then creates a directory named dynamo db and changes current directory to the created dynamodb directory.

```
sudo apt-get install default-jre
```

The next step is to install JRE (Java runtime)

```
[sudo] password for taohu:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
default-jre is already the newest version (2:1.17-74).
default-jre set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 944 not upgraded.
```

The output shows that default-jre has already been installed in my virtual machine.

```
wget
https://s3-ap-northeast-1.amazonaws.com/dynamodb-local-tokyo/dynamodb_lo
cal_latest.tar.gz
```

The above command is used to download the dynamodb file from url.

```
--2023-08-20 12:53:39--
https://s3-ap-northeast-1.amazonaws.com/dynamodb-local-tokyo/dynamodb_lo
cal_latest.tar.gz
Resolving s3-ap-northeast-1.amazonaws.com
(s3-ap-northeast-1.amazonaws.com)... 52.219.172.40, 52.219.16.62,
52.219.136.244, ...
Connecting to s3-ap-northeast-1.amazonaws.com
(s3-ap-northeast-1.amazonaws.com)|52.219.172.40|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 44278951 (42M) [application/x-tar]
Saving to: 'dynamodb_local_latest.tar.gz'

dynamodb_local_latest.tar.gz
100%[=====>] 42.23M
3.22MB/s in 15s

2023-08-20 12:53:55 (2.83 MB/s) - 'dynamodb_local_latest.tar.gz' saved
[44278951/44278951]
```

Output verifies that the download is successful.

```
tar -xvzf dynamodb_local_latest.tar.gz
```

The above command is used to uncompress the file I have just downloaded. The output of this command will show a list of all the files uncompressed and all stored inside the current working directory.

```
ls -l
```

Use the above command to verify the files uncompressed.

```
-rw-r--r-- 1 taohu taohu 6092913 Jun 30 19:33 DynamoDBLocal.jar
-rw-r--r-- 1 taohu taohu 44278951 Jun 30 19:43
dynamodb_local_latest.tar.gz
drwxr-xr-x 2 taohu taohu      12288 Aug 20 12:55 DynamoDBLocal_lib
-rw-r--r-- 1 taohu taohu      9450 Jun 30 19:33 LICENSE.txt
-rw-r--r-- 1 taohu taohu      5625 Jun 30 19:33 README.txt
-rw-r--r-- 1 taohu taohu     27190 Jun 30 19:33 THIRD-PARTY-LICENSES.txt
```

The above output shows that there is a .jar file named DynamoDBLocal.jar and a directory named DynamoDBLocal_lib. These two files will be used with java to initialise the database.

```
java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar
-sharedDb
```

The above command will initialise a dynamodb locally. The first two options of the java command are required. The first option specifies the java library location for dynamodb which is the directory uncompressed earlier. The second option -jar specifies the .jar file that will contain a main program to be executed in this case. The last option -sharedDb will initialise only one database that is shared, if not specified then a new database is initialised for each region and accesskey

```
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on
-Dswing.aatext=true
Initializing DynamoDB Local with the following configuration:
Port:      8000
InMemory:   false
DbPath: null
SharedDb:   false
shouldDelayTransientStatuses: false
CorsParams: null
```

Above output shows that the command had no error.

Now I need to create a table in the local database, with attributes from the lab specification.


```

import boto3

ddb = boto3.client("dynamodb", endpoint_url="http://localhost:8000")
ct_response = ddb.create_table(
    AttributeDefinitions=[
        {
            "AttributeName": "fileId",
            "AttributeType": "S"
        },
    ],
    KeySchema=[
        {
            "AttributeName": "fileId",
            "KeyType": "HASH"
        }
    ],
    ProvisionedThroughput={
        "ReadCapacityUnits": 10,
        "WriteCapacityUnits": 10
    },
    TableName="CloudFiles"
)

dt_response = ddb.describe_table(TableName="CloudFiles")
print(dt_response)

```

The above is the code from a newly created python file called `initialiseDynamoDb.py`. When this file runs it will create a new table in the local dynamoDb. In the above code, `ddb` is initialised as a `dynamodb` with `endpoint_url` pointing to the local database running. The `create_table` function is called to create the new table. `AttributeDefinitions` defines the primary key attribute and type. `KeySchema` defines how the primary will be used, as composite or single with just `HASH`. `ProvisionedThroughput` defines the read and write limit to the database per second. `TableName` defines the name of the table, which here is `CloudFiles`.

The lab specification states that `userId` should be used as the key for the table. However, this does not make much sense, since I don't know what `userId` refers to and each file does not contain a `userId`. Instead, `fileId` is used which is the `file_key` of the S3 bucket object.

```

{'Table': {'AttributeDefinitions': [{'AttributeName': 'fileId',
'AttributeType': 'S'}], 'TableName': 'CloudFiles', 'KeySchema':
[{'AttributeName': 'fileId', 'KeyType': 'HASH'}], 'TableStatus':
'ACTIVE', 'CreationDateTime': datetime.datetime(2023, 8, 20, 16, 30, 2,
116000, tzinfo=tzlocal()), 'ProvisionedThroughput':
{'LastIncreaseDateTime': datetime.datetime(1970, 1, 1, 8, 0,
tzinfo=tzlocal()), 'LastDecreaseDateTime': datetime.datetime(1970, 1, 1,

```

```

8, 0, tzinfo=tzlocal()), 'NumberOfDecreasesToday': 0,
'ReadCapacityUnits': 10, 'WriteCapacityUnits': 10}, 'TableSizeBytes': 0,
'ItemCount': 0, 'TableArn':
'arn:aws:dynamodb:dynamodblocal:000000000000:table/CloudFiles'},
'ResponseMetadata': {'RequestId':
'f97e7438-12a8-4e23-a0f4-93fa95d45716', 'HTTPStatusCode': 200,
'HTTPHeaders': {'date': 'Sun, 20 Aug 2023 08:30:02 GMT',
'x-amzn-requestid': 'f97e7438-12a8-4e23-a0f4-93fa95d45716',
'content-type': 'application/x-amz-json-1.0', 'x-amz-crc32':
'535222324', 'content-length': '483', 'server':
'Jetty(9.4.48.v20220622)'}, 'RetryAttempts': 0}}

```

The above is the output showing the data of the newly created table. The `describe_table` function is called to truly verify that the database does indeed exist.

The next step is to create another python file that will read all the objects from the S3 bucket and store relevant information into the local dynamodb.

```

import boto3
import os
import json

ROOT_S3_DIR = '23805764-cloudstorage'
TABLE_NAME="CloudFiles"

s3 = boto3.client("s3")
ddb = boto3.client("dynamodb", endpoint_url="http://localhost:8000")

lo_response = s3.list_objects(
    Bucket=ROOT_S3_DIR,
)
contents = lo_response["Contents"]

for content in contents:
    file_key = content["Key"]
    # get the permission or access control list
    goa_response = s3.get_object_acl(
        Bucket=ROOT_S3_DIR,
        Key=file_key
    )
    ddb.put_item(
        TableName=TABLE_NAME,
        Item={
            "fileId": {

```

```

        # assume the userId to be my student id, could not
        # find a explicit userId property in these objects
        "S": file_key
    },
    "fileName": {
        # assume the filename is the name of the file
        "S": os.path.basename(file_key)
    },
    "path": {
        # assume path as the nested directory paths
        "S": os.path.dirname(file_key)
    },
    "lastUpdated": {
        # content has a key called LastModified
        "S": str(content["LastModified"])
    },
    "owner": {
        # Owner key will contain a object with owner name and
        # the owner id, DisplayName is the key to owner name
        "S": content["Owner"]["DisplayName"]
    },
    "permissions": {
        # acces the key Grants containing a list of permissions
        # in my S3 bucket there is only user permission defined
        # list is therefore only length of 1
        "S": str(goa_response["Grants"])
    }
}

)

scan_response = ddb.scan(
    TableName=TABLE_NAME
)

print(json.dumps(scan_response["Items"], indent=4))

```

The above python with the name of `populatecloudfiles.py` is created with the purpose to solve the task defined previously. `list_objects` is used to retrieve information of all objects in the S3 bucket. From previous inspections of the response, I found out that most of the required data are included except permissions. The `put_item` function is called to add a new item into the local dynamodb. The function `get_object_acl` is called to retrieve the access control list of an object. Provide the `file_key` to identify the object. The response will contain an object with a key of `Grants` which contains an object of permission information of different groups. In my case, only one group of permissions are defined which is for the user myself. The `scan` function of dynamodb is called to retrieve a list of all the items

inside the table specified by TableName. The response from this function call is printed to verify that the correct information is stored.

```
python populatecloudfiles.py
```

The above command to execute the python file.

```
[
  {
    "owner": {
      "S": "zhi.zhang"
    },
    "path": {
      "S": "rootdir"
    },
    "lastUpdated": {
      "S": "2023-08-19 10:42:47+00:00"
    },
    "fileName": {
      "S": "rootfile.txt"
    },
    "permissions": {
      "S": "[{'Grantee': {'DisplayName': 'zhi.zhang', 'ID':
'2a5fac7aada1ad2caa48c9ab08cc4e2428d4eb596108daa3b59f1204ae96482e',
'Type': 'CanonicalUser'}, 'Permission': 'FULL_CONTROL'}]"
    },
    "fileId": {
      "S": "rootdir/rootfile.txt"
    }
  },
  {
    "owner": {
      "S": "zhi.zhang"
    },
    "path": {
      "S": "rootdir/subdir"
    },
    "lastUpdated": {
      "S": "2023-08-19 10:42:47+00:00"
    },
    "fileName": {
      "S": "subfile.txt"
    },
    "permissions": {
      "S": "[{'Grantee': {'DisplayName': 'zhi.zhang', 'ID':
```

```
'2a5fac7aada1ad2caa48c9ab08cc4e2428d4eb596108daa3b59f1204ae96482e',
'Type': 'CanonicalUser'}, 'Permission': 'FULL_CONTROL'}]]"
    },
    "fileId": {
        "S": "rootdir/subdir/subfile.txt"
    }
}
]
```

The above is the output. The printed output is from the values under the key Items of the response from scan function. It can be seen that two items are printed with the correct information.

Lab 4 - KMS Encryption

This lab is about creating a KMS key and using it to encrypt and decrypt files from S3 using boto3.

[1] Apply policy to restrict permissions on bucket

With a given policy template from the lab specification. I modified a few values to make the policy to only accept requests from a specific student number.

I first need to find the ARN of the bucket I created.

Steps:

1. Login to AWS console
2. Click services -> storage -> S3
3. In the search box type my student id 23805764
4. Click on the bucket
5. Click on properties
6. ARN can be copied from bucket overview section

arn:aws:s3:::23805764-cloudstorage is my bucket ARN.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "AllowAllS3ActionsInUserFolderForUserOnly",
    "Effect": "Deny",
    "Principal": "*",
    "Action": "s3:*",
    "Resource": "arn:aws:s3:::23805764-cloudstorage/rootdir/subdir/*",
    "Condition": {
      "StringNotLike": {
        "aws:username": "23805765@student.uwa.edu.au"
      }
    }
  }
}
```

```
}  
}
```

The above is the modified policy. My student Id is 23805764 so I changed the aws:username to use 23805765 as the email student id. The policy has an effect of deny so any user without the corresponding username will be denied when accessing the resource. The resource in the above policy is all files inside the rootdir/subdir/ directory of my S3 bucket.

After some trial and error, I found out that the “Deny” should be used for the “Effect” key instead of “DENY”.

After some research, I found out that there is a create_policy function that can be called to create a new policy for IAM, which is the requirement. Then I found out there is a function called put_bucket_policy for s3 which adds a new policy specifically for the s3 bucket defined by the policy “Resource” key.

```
import boto3  
import json  
  
ROOT_S3_DIR = '23805764-cloudstorage'  
  
step_1_policy = {  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowAllS3ActionsInUserFolderForUserOnly",  
            "Effect": "Deny",  
            "Action": "s3:*",  
            "Principal": "*",  
            "Resource":  
"arn:aws:s3:::23805764-cloudstorage/rootdir/subdir/*",  
            "Condition": {  
                "StringNotLike": {  
                    "aws:username": "23805765@student.uwa.edu.au"  
                }  
            }  
        }  
    ]  
}  
  
s3 = boto3.client("s3")  
  
try:  
    pp_response = s3.put_bucket_policy(  

```

```

        Bucket=ROOT_S3_DIR,
        Policy=json.dumps(step_1_policy)
    )
    print("Sucessfully created policy")
    print(json.dumps(pp_response, indent=2, default=str))
except Exception as ex:
    print("Error failed to create policy.")
    print(ex)

```

I created a python file called addPolicy.py containing the code above. The create_policy function is used and the policy data is passed as a json to the PolicyDocument parameter.

Before creating the policy I created another python file that will run the get_object function in one of the files inside that directory. From lab 3, there is a file called subfile.txt inside rootdir/subdir. This file will be the subject file that I will run get_object on.

```

import boto3
import json

ROOT_S3_DIR = '23805764-cloudstorage'

s3 = boto3.client("s3")

file_key = "rootdir/subdir/subfile.txt"

try:
    go_response = s3.get_object(
        Bucket=ROOT_S3_DIR,
        Key=file_key
    )
    print(f"Successfully ran get_object on {file_key}")
    print(json.dumps(go_response, indent=2, default=str))
except Exception as ex:
    print(f"Error with exception of {type(ex).__name__}")

```

The above is the python code inside a file I created called getSubfile.py. The purpose is to use this to test later on if the policy did work.

Now I will first test to see if I can successfully run the function on the file first before applying the policy.

```
python3 getSubfile.py
```

Use the above command to run the python program

```
Successfully ran get_object on rootdir/subdir/subfile.txt
{
  "ResponseMetadata": {
    "RequestId": "GPNQK0N8VCHDTCC2",
    "HostId":
"4UW7thkKOYJv3FV3ShNpxbv5m4kCghfuyXoRJ7/pB07sayiohzwpyONgEsQOpPMYE9g1Goh
D4SM=",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "x-amz-id-2":
"4UW7thkKOYJv3FV3ShNpxbv5m4kCghfuyXoRJ7/pB07sayiohzwpyONgEsQOpPMYE9g1Goh
D4SM=",
      "x-amz-request-id": "GPNQK0N8VCHDTCC2",
      "date": "Wed, 30 Aug 2023 07:48:58 GMT",
      "last-modified": "Sat, 19 Aug 2023 10:42:47 GMT",
      "etag": "\"9aecac258878f548545afde321f8a79c\"",
      "x-amz-server-side-encryption": "AES256",
      "accept-ranges": "bytes",
      "content-type": "binary/octet-stream",
      "server": "AmazonS3",
      "content-length": "11"
    },
    "RetryAttempts": 0
  },
  "AcceptRanges": "bytes",
  "LastModified": "2023-08-19 10:42:47+00:00",
  "ContentLength": 11,
  "ETag": "\"9aecac258878f548545afde321f8a79c\"",
  "ContentType": "binary/octet-stream",
  "ServerSideEncryption": "AES256",
  "Metadata": {},
  "Body": "<botocore.response.StreamingBody object at 0xffff8ba9c730>"
}
```

The above output in the second line shows that I have successfully retrieved the object data and displayed it.

```
python3 addPolicy.py
```

Above command to run the python file.

```
Sucessfully created policy
{
```



```

    "ResponseMetadata": {
        "RequestId": "YZCR43J39CXRS3SS",
        "HostId":
    "dxn/rSVlphp04DnYN6JnVZRwp73RWVgtwRo58WEpgM2mVKWCAg/JjVSqu7T09VNJkB9m2p1
f0KY=",
        "HTTPStatusCode": 204,
        "HTTPHeaders": {
            "x-amz-id-2":
    "dxn/rSVlphp04DnYN6JnVZRwp73RWVgtwRo58WEpgM2mVKWCAg/JjVSqu7T09VNJkB9m2p1
f0KY=",
            "x-amz-request-id": "YZCR43J39CXRS3SS",
            "date": "Wed, 30 Aug 2023 09:46:26 GMT",
            "server": "AmazonS3"
        },
        "RetryAttempts": 0
    }
}

```

The above output shows I have successfully created the policy. It also shows the response received from the `create_policy` function in `boto3`.

Now try to perform s3 actions on the file and again and there should be access denied error.

```
python3 getSubfile.py
```

Above to run the python file to get the object again.

```

An error occurred (AccessDenied) when calling the GetObject operation:
Access Denied

```

Above output shows that I can no longer access the object, which is as expected.

[2] AES Encryption using KMS

In lab 3, a s3 bucket is created and a local directory called `rootdir` is also created. `cloudstorage.py` is a python file that will upload all contents inside `rootdir` into the s3 bucket. An encryption method is not set, which means the default AWS s3 bucket encryption is used. The goal of this section of the lab is to create a modified version of `cloudstorage.py`, which uses KMS as the encryption method.

The lab specifications provided a template for the policies to be used for KMS encryption. I then read about KMS for `boto3` and researched on how KMS is to be used with s3 bucket using `boto3`. I found that as part of the `upload_file` function from `obto3`, some optional parameters can be used to specify the file to use KMS and also specify the KMS to be used.

No extra parameters are required for `download_file`. The reason is my configured AWS has an access key which can identify me on each request. When `download_file` is invoked AWS will check my identity in the request and if I have permission for the KMS key, then AWS will decrypt on the server side and successfully send back the file. This means I have to make sure when the KMS key is created the policy attached must give me the permission to use the KMS key.

The policy given from the lab specification is outdated with incorrect AWS reference. After some research, I found a AWS cli command which can be used to find out my correct user reference.

```
aws sts get-caller-identity
```

The above command is run to find out my user information, containing my user reference to the ARN.

```
{
  "UserId": "AIDAXD4PI5LY2NHRKJCRB",
  "Account": "489389878001",
  "Arn":
  "arn:aws:iam::489389878001:user/23805764@student.uwa.edu.au"
}
```

The above is the output, which shows my ARN. I will replace the outdated AWS ARN in the provided JSON policy with the new one from above.

The first step is to create a KMS key. In the boto3 documentation, `create_key` function can be used to create a new KMS key. An optional parameter in the function called `Policy` can be used to take the KMS policy in JSON format.

```
import boto3
import json

user_ref = "arn:aws:iam::489389878001:user/23805764@student.uwa.edu.au"

key_policy = {
  "Version": "2012-10-17",
  "Id": "key-consolepolicy-3",
  "Statement": [
    {
      "Sid": "Enable IAM User Permissions",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::032418238795:root"
      },
      "Action": "kms:*",
      "Resource": "*"
    }
  ]
}
```

```

},
{
  "Sid": "Allow access for Key Administrators",
  "Effect": "Allow",
  "Principal": {
    "AWS": user_ref
  },
  "Action": [
    "kms:Create*",
    "kms:Describe*",
    "kms:Enable*",
    "kms:List*",
    "kms:Put*",
    "kms:Update*",
    "kms:Revoke*",
    "kms:Disable*",
    "kms:Get*",
    "kms>Delete*",
    "kms:TagResource",
    "kms:UntagResource",
    "kms:ScheduleKeyDeletion",
    "kms:CancelKeyDeletion"
  ],
  "Resource": "*"
},
{
  "Sid": "Allow use of the key",
  "Effect": "Allow",
  "Principal": {
    "AWS": user_ref
  },
  "Action": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt*",
    "kms:GenerateDataKey*",
    "kms:DescribeKey"
  ],
  "Resource": "*"
},
{
  "Sid": "Allow attachment of persistent resources",
  "Effect": "Allow",

```

```

    "Principal": {
        "AWS": user_ref
    },
    "Action": [
        "kms:CreateGrant",
        "kms:ListGrants",
        "kms:RevokeGrant"
    ],
    "Resource": "*",
    "Condition": {
        "Bool": {
            "kms:GrantIsForAWSResource": "true"
        }
    }
}
]
}

kms_client = boto3.client("kms")

try:
    ck_response = kms_client.create_key(
        Policy=json.dumps(key_policy),
        Description="Key for s3 bucket, used in lab 4."
    )
    key_id = ck_response["KeyMetadata"]["KeyId"]
    print(f"Successfully created KMS key with id: {key_id}")

    ca_response = kms_client.create_alias(
        AliasName="alias/23805764",
        TargetKeyId=key_id
    )
    print(f"Successfully created alias")
    print(json.dumps(ca_response, indent=2, default=str))
except Exception as ex:
    print("Failed to create key")
    print(ex)

```

The above is the python code of a new file I created called createKMSKey.py. This program when ran will create a new KMS key and create an alias for the key with my student id as the value.

create_key function is used to create the key. Two parameters are used, Policy containing the key policy and Description. create_alias function is used to add an alias to the newly created KMS key.

Two parameters are used, the TargetKeyId to identify the KMS key and AliasName to specify the alias name but must begin with alias/.

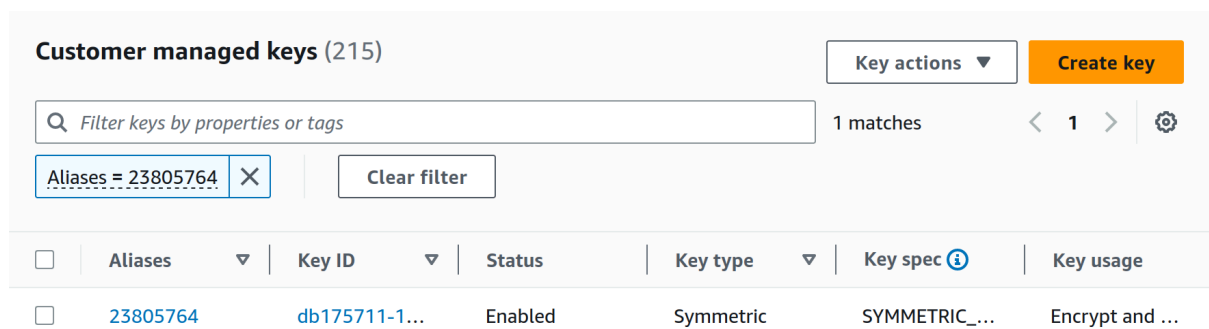
```
python3 createKMSKey.py
```

Run the python file to create a new KMS key.

```
Successfully created KMS key with id:
db175711-178b-4975-8afb-a2adf6d66400
Successfully created alias
{
  "ResponseMetadata": {
    "RequestId": "7574e5d6-12a1-45e9-8765-ba4390f92e57",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "x-amzn-requestid": "7574e5d6-12a1-45e9-8765-ba4390f92e57",
      "cache-control": "no-cache, no-store, must-revalidate, private",
      "expires": "0",
      "pragma": "no-cache",
      "date": "Wed, 30 Aug 2023 11:46:07 GMT",
      "content-type": "application/x-amz-json-1.1",
      "content-length": "0",
      "connection": "keep-alive"
    },
    "RetryAttempts": 0
  }
}
```

The above is the output, which shows the newly created KMS key id.

To verify that all operations are indeed all successful, I can check in the AWS console. This can be done by navigating to the KMS dashboard through the search bar and then search alias 23805764.



Customer managed keys (215)							Key actions ▾	Create key
🔍 Filter keys by properties or tags							1 matches	< 1 > ⚙️
Aliases = 23805764 ✕ Clear filter								
<input type="checkbox"/>	Aliases ▾	Key ID ▾	Status	Key type ▾	Key spec ⓘ	Key usage		
<input type="checkbox"/>	23805764	db175711-1...	Enabled	Symmetric	SYMMETRIC_...	Encrypt and ...		

The above screenshot taken from the KMS dashboard shows that my key does indeed exist with the correct alias.

The next step is to create a new cloudstorage.py file to make use of this KMS key for encryption.

```
cp -rd ../lab3/rootdir .;  
mv rootdir rootdir2
```

Above command to make a copy of the rootdir content used in lab 3 into the current lab4 directory. Then rootdir is renamed to rootdir2, this will help to distinguish from the rootdir uploaded to S3 bucket in lab3.

```
ls -l
```

List current directory to verify.

```
total 24  
-rw-r--r-- 1 taohu taohu 821 Aug 30 17:46 addPolicy.py  
-rw-r--r-- 1 taohu taohu 1439 Aug 30 19:58 cloudstorage.py  
-rw-r--r-- 1 taohu taohu 2206 Aug 30 19:45 createKMSKey.py  
-rw-r--r-- 1 taohu taohu 375 Aug 30 17:08 getSubfile.py  
drwxr-xr-x 3 taohu taohu 4096 Aug 30 19:59 rootdir2  
drwxr-xr-x 2 taohu taohu 4096 Aug 30 19:52 ss
```

Above output shows that rootdir is successfully copied to lab4.

First I made a copy of the cloudstoreage.py file from lab3 to lab4 inside my VSCode IDE. Then I need to modify the file to use KMS encryption when uploading files from local to the S3 bucket.

```
ROOT_DIR = '.'  
ROOT_S3_DIR = '23805764-cloudstorage'  
  
KMS_KEY_ID = "db175711-178b-4975-8afb-a2adf6d66400"
```

In the constant area, I define a constant for my KMS key id.

```
def upload_file(folder_name, file, file_name):  
    s3.upload_file(file, ROOT_S3_DIR, f"{folder_name}{file_name}",  
ExtraArgs={  
        "ServerSideEncryption": "aws:kms",  
        "SSEKMSKeyId": KMS_KEY_ID  
    })  
    print("Uploading %s" % file)
```

Then change the upload_file function. ExtraArgs parameter is added to enable KMS encryption for the files uploaded.

```
python3 cloudstorage.py
```

Above command to run the python program.

```
Uploading ./rootdir2/rootfile.txt
Uploading ./rootdir2/subdir/subfile.txt
done
```

Output shows that there are no errors during the program.

I need to verify that my files are actually encrypted using the KMS key. This can be done from the AWS console. I first navigate to the S3 dashboard then find my bucket using my student id 23805764. Then I view the content inside my S3 bucket and find the rootdir2 directory. Navigating inside I should find a file called rootfile.txt. Viewing the data of the file, I should be able to confirm if it is encrypted using KMS.



Server-side encryption settings [Info](#) Edit

Server-side encryption protects data at rest.


Encryption type [Info](#)

Server-side encryption with AWS Key Management Service keys (SSE-KMS)

Encryption key ARN

 [arn:aws:kms:ap-southeast-2:489389878001:key/db175711-178b-4975-8afb-a2adf6d66400](#) 

Bucket Key

When KMS encryption is used to encrypt new objects in this bucket, the bucket key reduces encryption costs by lowering calls to AWS KMS. [Learn more](#) 

Disabled

The above screenshot is taken in the rootfile.txt's properties section in the S3 bucket dashboard. It can be seen that the file is indeed encrypted using my KMS key.

```
cp ../lab3/restorefromcloud.py
```

Get the python file from lab 3, then use it to verify that I can download and decrypt the files from S3 bucket successfully.

```
python3 restorefromcloud.py fromcloud
```

The above command is run to restore all contents inside my S3 bucket and store inside a directory named fromcloud.

```
Restoring contents from s3 bucket: 23805764-cloudstorage into fromcloud
```

```
Restored ./rootdir/rootfile.txt from bucket into
fromcloud/./rootdir/rootfile.txt locally.
Restored ./rootdir/subdir/subfile.txt from bucket into
fromcloud/./rootdir/subdir/subfile.txt locally.
Restored rootdir/rootfile.txt from bucket into
fromcloud/rootdir/rootfile.txt locally.
Failed to restore rootdir/subdir/subfile.txt.
Restored rootdir2/rootfile.txt from bucket into
fromcloud/rootdir2/rootfile.txt locally.
Restored rootdir2/subdir/subfile.txt from bucket into
fromcloud/rootdir2/subdir/subfile.txt locally.

Done
```

The output shows that most are successful except for one file. The reason is previously I made a mistake and added a duplicated clone of rootdir into my S3 bucket. Importantly, rootdir2 is the interested subject directory, which is the directory uploaded with KMS encryption. rootdir2 is successfully restored.

```
cat fromcloud/rootdir2/rootfile.txt
```

Use the above command to print the content of rootfile.txt inside rootdir2.

```
1
2
3
4
5
```

The above output shows that I do indeed have the decrypted version of rootfile.txt.

All above processes conclude that I have correctly uploaded files locally to my S3 bucket and encrypting with my newly created KMS key. I am also able to download the decrypted version from S3 bucket. The decryption and encryption are both done at the server side by AWS. I am able to do so because my KMS key has a policy that specified I am allowed to do so.

[3] AES Encryption using local python library pycryptodome

In step 2, KMs is used to encrypt and decrypt the files uploaded and downloaded from S3 bucket. The encryption and decryption are both done on the server side by AWS.

Now, pycryptodome will be used as the method to encrypt and decrypt files to be stored in S3 bucket. Since pycryptodome is a python library to be installed locally and AWS does not support this method. The encryption and decryption will be done locally on the client side.

A copy of cloudstorage.py will be created with the name of cloudstorage2.py. The upload_file function will be modified to first encrypt the file using pycryptodome locally and upload the encrypted file to S3 bucket.

A performance comparison between KMS and pycryptodome is also required. The time module will be used to record the start and end time of uploading the files for both versions. To do so, first modify the cloudstoage.py file to print the time it takes to complete the upload.

```
# the start time in milliseconds
start_time = int(time.time() * 1000)

# parse directory and upload files

for dir_name, subdir_list, file_list in os.walk(ROOT_DIR,
topdown=True):
    if dir_name != ROOT_DIR and not "ss" in dir_name:
        for fname in file_list:
            upload_file("%s/" % dir_name[2:], "%s/%s" % (dir_name,
fname), fname)

stop_time = int(time.time() * 1000)

# print the time taken in milliseconds
print(f"done in {stop_time - start_time} milliseconds")
```

Modify the main executing section by having a start time before upload starts and end time after the program completes. The total time taken in milliseconds is printed right before the program ends.

The same modification is also applied to the restorefromcloud.py file, to print out the time it takes to retrieve all files from S3 bucket.

```
pip3 install pycryptodome
```

First use pip to install the pycryptodome library.

```
Defaulting to user installation because normal site-packages is not
writeable
Collecting pycryptodome
  Downloading pycryptodome-3.18.0-cp35-abi3-manylinux2014_aarch64.whl
(2.1 MB)

-----
— 2.1/2.1 MB 4.6 MB/s eta 0:00:00
Installing collected packages: pycryptodome
```

```
Successfully installed pycryptodome-3.18.0
```

The above output shows the pycryptodome library is successfully installed.

```
wget
https://raw.githubusercontent.com/zhangzhics/CITS5503_Sem2_2023/master/Labs/src/fileencrypt.py
```

The above command is used to download fileencrypt.py from the course's github repository. This file contains a function to encrypt and decrypt using pycryptodome.

```
--2023-08-31 17:22:07--
https://raw.githubusercontent.com/zhangzhics/CITS5503_Sem2_2023/master/Labs/src/fileencrypt.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1943 (1.9K) [text/plain]
Saving to: 'fileencrypt.py'

fileencrypt.py
100%[=====>]      1.90K
--.-KB/s      in 0.001s

2023-08-31 17:22:08 (2.81 MB/s) - 'fileencrypt.py' saved [1943/1943]
```

Above output shows the file is successfully downloaded. The main program contains an example which I opened and removed.

```
LOCAL_PASSWORD = "Password for step 3 lab 4"
BLOCK_SIZE = 16
CHUNK_SIZE = 64 * 1024
```

A LOCAL_PASSWORD constant is also declared for use when encrypting and decrypting later.

```
cp cloudstorage.py cloudstorage2.py
```

Create a copy of cloudstorage.py called cloudstorage2.py then modify the program to use pycryptodome.

Now I open cloudstorage2.py with VSCode IDE to modify the program.

```

import os
import boto3
import base64
import sys
import getopt

import time
from fileencrypt import LOCAL_PASSWORD, encrypt_file

```

First add a line to import fileencrypt.

```

def upload_file(folder_name, file, file_name):
    # define the file name to save encrypted file into
    encrypted_file = f"{file}.enc"
    encrypt_file(LOCAL_PASSWORD, file, encrypted_file)
    try:
        print("Uploading %s" % file)
        # upload the encrypted version of the file to S3 bucket
        s3.upload_file(encrypted_file, ROOT_S3_DIR,
f"{folder_name}{file_name}")
    except:
        print(f"Error failed to upload {file}")
        # clean up by remove ing the encrypted file
        os.remove(encrypted_file)

```

Modify the upload_file to the above, which uses the encrypt function inside fileencrypt.py. The function works by first calling the encrypt_file function inside fileencrypt.py, which will take the name of the file and create a new file to store the encrypted content. Then the encrypted file is uploaded to S3 bucket and removed locally.

Now I need to make a new restorefromcloud.py file for the custom method.

```
cp restorefromcloud.py restorefromcloud2.py
```

Make a new copy of restorefromcloud.py for the custom method.

```

import boto3
import os
import sys

import time
from fileencrypt import LOCAL_PASSWORD, encrypt_file

ROOT_S3_DIR = '23805764-cloudstorage'

```

```

try:
    out_dir = sys.argv[1]
except:
    print("Error, need to specify the output directory name.")
    print("Usage: python restorefromcloud.py <out_dir_name>")
    exit(0)

print(f"Restoring contents from s3 bucket: {ROOT_S3_DIR} into {out_dir}\n")

s3 = boto3.client("s3")
lo_response = s3.list_objects(Bucket=ROOT_S3_DIR)

# the start time in milliseconds
start_time = int(time.time() * 1000)

for content in lo_response["Contents"]:
    file_key = content["Key"]
    try:
        local_path_file = os.path.join(out_dir, file_key)
        # store the downloaded encrypted file
        local_path_file_enc = f"{local_path_file}.enc"
        local_path_dir = os.path.dirname(local_path_file)
        if not os.path.exists(local_path_dir):
            os.makedirs(local_path_dir)
        # save to encrypted file name
        s3.download_file(Bucket=ROOT_S3_DIR, Key=file_key,
Filename=local_path_file_enc)
        decrypt_file(LOCAL_PASSWORD, local_path_file_enc,
local_path_file)
        print(f"Restored {file_key} from bucket into {local_path_file} locally.")
        # remove the encrypted file
        os.remove(local_path_file_enc)
    except:
        print(f"Failed to restore {file_key}.")

stop_time = int(time.time() * 1000)

# print the time taken in milliseconds
print(f"done in {stop_time - start_time} milliseconds")

```

The above is python code inside restorefromcloud2.py. Encrypted files are downloaded then decrypted locally. Executing time is also printed at the end before the program exits.

Now is time to execute the program and see the performance of the KMS version.

```
rm -rd fromcloud;  
python3 cloudstorage.py;  
python3 restorefromcloud.py fromcloud;
```

The above command first rm the fromcloud directory which contains previously downloaded files. Then cloudstorage.py is run to upload all files inside the local rootdir directory. Then restorefromcloud.py is called to download all the files from S3 bucket.

```
Uploading ./rootdir2/rootfile.txt  
Uploading ./rootdir2/subdir/subfile.txt  
done in 615 milliseconds  
Restoring contents from s3 bucket: 23805764-cloudstorage into fromcloud  
  
Restored rootdir/rootfile.txt from bucket into  
fromcloud/rootdir/rootfile.txt locally.  
Failed to restore rootdir/subdir/subfile.txt.  
Restored rootdir2/rootfile.txt from bucket into  
fromcloud/rootdir2/rootfile.txt locally.  
Restored rootdir2/subdir/subfile.txt from bucket into  
fromcloud/rootdir2/subdir/subfile.txt locally.  
done in 610 milliseconds
```

The above output shows that on average the upload and download are both around 610 milliseconds.

Now it is time to run the upload and download process with the custom encryption method.

```
touch __init__.py
```

The above creates an empty python file, the points to allow imports from the current directory.

Now run repeat with the custom encryption method

```
rm -rd fromcloud;  
python3 cloudstorage2.py;  
python3 restorefromcloud2.py fromcloud;
```

The above are the commands.

```
Uploading ./__pycache__/fileencrypt.cpython-311.pyc
Uploading ./rootdir2/rootfile.txt
Uploading ./rootdir2/subdir/subfile.txt
done in 710 milliseconds
Restoring contents from s3 bucket: 23805764-cloudstorage into fromcloud

Restored __pycache__/fileencrypt.cpython-311.pyc from bucket into
fromcloud/__pycache__/fileencrypt.cpython-311.pyc locally.
Failed to restore rootdir/rootfile.txt.
Failed to restore rootdir/subdir/subfile.txt.
Restored rootdir2/rootfile.txt from bucket into
fromcloud/rootdir2/rootfile.txt locally.
Restored rootdir2/subdir/subfile.txt from bucket into
fromcloud/rootdir2/subdir/subfile.txt locally.
done in 684 milliseconds
```

The above is the output showing that rootdir2 has been restored correctly.

```
cat fromcloud/rootdir2/rootfile.txt
```

Check files are decrypted and stored correctly.

```
1
2
3
4
5
```

Above output shows that decryption is successful.

Answer the following question (Marked)

What is the performance difference between using KMS and using the custom solution ?

Assuming the performance difference is to be calculated from the time it takes to upload and retrieve the file. The KMS method has server side encryption when uploading files and server side decryption when downloading files. This means I cannot find a clear way of obtaining the actual time it took to encrypt and decrypt the file. With the custom encryption method, the actual time for encryption and decryption can be calculated because it is done locally. To align with the KMS method, performance measurement is therefore to be the total time measure for uploading all files and downloading all files.

The procedure used to first upload all files from my local rootdir2 directory to the S3 bucket and the total time is recorded and output at the end. Then all files from S3 bucket are downloaded with the time it takes to do so, recorded and outputted at the end as well. This procedure is done twice, one for the KSM method and other for the custom method. Then the time recorded is compared.

From my experiment, the outputs as shown above. It took around 610 milliseconds for both upload and download files when the KMS method is used. For the custom method, 710 milliseconds are used to upload all files and 684 milliseconds are used to download all files. Both processes only download and upload based on 2 files, which is the two used since lab 3.

The result shows clearly that the KMS method is much faster. However, this result does not truly reflect in the encryption and decryption speed. In the custom method, there is overhead of creating an extra file when uploading and downloading, then removing it. This is how the functions inside `filenecrypt.py` works, which is provided by the lab description.