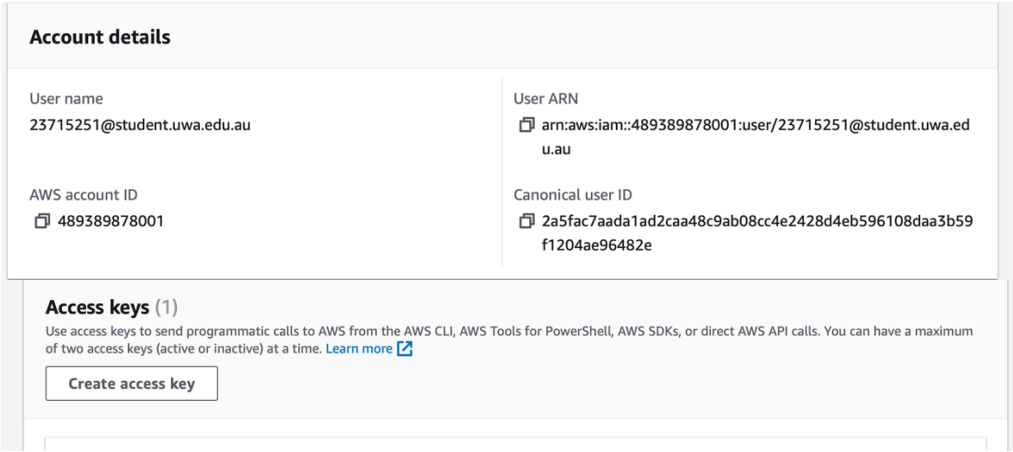
Name: Rongchuan Sun
Student number: 23715251

Lab1

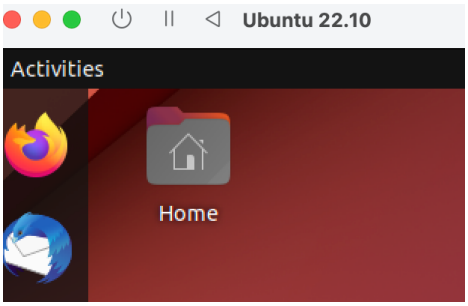
1. **AWS Accounts and login**
   Firstly, we need to login the account by provided email.
   Secondly, we need to click my user account. Then click security credentials to create access key and make a note of the access key id and the secret access key.
   Below is the screenshot of login.



   Below are the keys I have created.

   Access key
   AKIAXD4PI5LYZECWG3FW
   Secret access key
   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
   aws configure
   AWS Access Key ID [None]: AKIAXD4PI5LYZECWG3FW
   AWS Secret Access Key [None]: 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
   Default region name [None]: ap-southeast-2
   Default output format [None]: json

2. **Installing VM software**
   Below is the screenshot that I have installed VM.



3. **Install Python 3.8x**
   Below picture is to install python3 and check the version of python.

```
rongchuan@ubuntu:~$ python3 --version
Python 3.10.7
rongchuan@ubuntu:~$ pip3 --version
pip 22.2 from /usr/lib/python3/dist-packages/pip (python 3.10)
```

4. **Install awscli**

   Below picture is to install awscli.

```
rongchuan@ubuntu:~$ pip3 install awscli --upgrade
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: awscli in ./.local/lib/python3.10/site-packages (
1.29.22)
Successfully installed awscli-1.29.38 botocore-1.31.38
rongchuan@ubuntu:~$
```

5. **Install boto3**

   Below picture is to install boto3.

```
rongchuan@ubuntu:~$ pip3 install boto3
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: boto3 in ./.local/lib/python3.10/site-packages (1
.28.22)
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in ./.local/lib/python3.10
/site-packages (from boto3) (1.0.1)
Requirement already satisfied: botocore<1.32.0,>=1.31.22 in ./.local/lib/python3
.10/site-packages (from boto3) (1.31.38)
Requirement already satisfied: s3transfer<0.7.0,>=0.6.0 in ./.local/lib/python3.
10/site-packages (from boto3) (0.6.1)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/lib/python3/dist-pa
ckages (from botocore<1.32.0,>=1.31.22->boto3) (1.26.9)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/lib/python3/d
ist-packages (from botocore<1.32.0,>=1.31.22->boto3) (2.8.1)
```

6. **Test the aws environment by running**

   Below picture is to test the aws environment by running code.

   We can see the environment is normal.

```
rongchuan@ubuntu:~$ aws ec2 describe-regions --output table
-----------------------------------------------------------------------------
|                              DescribeRegions                              |
+---------------------------------------------------------------------------+
||                                 Regions                                 ||
|+-----------------------------------+----------------------+---------------+|
||            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   ||
```

7. **Test the python environment**

   Below the picture is to test python environment.

   We can see the environment is normal.

```
rongchuan@ubuntu:~$ python3
Python 3.10.7 (main, May 29 2023, 13:51:48) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import boto3
>>> ec2 = boto3.client('ec2')
>>> response = ec2.describe_regions()
>>> print(response)
{'Regions': [{'Endpoint': 'ec2.ap-south-1.amazonaws.com', 'RegionName': 'ap-sout
h-1', 'OptInStatus': 'opt-in-not-required'}, {'Endpoint': 'ec2.eu-north-1.amazon
aws.com', 'RegionName': 'eu-north-1', 'OptInStatus': 'opt-in-not-required'}, {'E
ndpoint': 'ec2.eu-west-3.amazonaws.com', 'RegionName': 'eu-west-3', 'OptInStatus
': 'opt-in-not-required'}, {'Endpoint': 'ec2.eu-west-2.amazonaws.com', 'RegionNa
me': 'eu-west-2', 'OptInStatus': 'opt-in-not-required'}, {'Endpoint': 'ec2.eu-we
st-1.amazonaws.com', 'RegionName': 'eu-west-1', 'OptInStatus': 'opt-in-not-requi
red'}, {'Endpoint': 'ec2.ap-northeast-3.amazonaws.com', 'RegionName': 'ap-northe
ast-3', 'OptInStatus': 'opt-in-not-required'}, {'Endpoint': 'ec2.ap-northeast-2.
amazonaws.com', 'RegionName': 'ap-northeast-2', 'OptInStatus': 'opt-in-not-requi
red'}, {'Endpoint': 'ec2.ap-northeast-1.amazonaws.com', 'RegionName': 'ap-northe
ast-1', 'OptInStatus': 'opt-in-not-required'}, {'Endpoint': 'ec2.ca-central-1.am
azonaws.com', 'RegionName': 'ca-central-1', 'OptInStatus': 'opt-in-not-required'
}, {'Endpoint': 'ec2.sa-east-1.amazonaws.com', 'RegionName': 'sa-east-1', 'OptIn
```

8. **Put this code into python file and tabulate the print to have 2 columns with endpoint and regionName.**

   Below is the code placed into a Python file with a tabulated printout showing the 'Endpont' and 'RegionName' in two columns.

   This script will list the AWS regions along with their respective endpoints and display them in a tabulated format with two columns.

```python
import boto3
import tabulate

# Initialize the EC2 client
ec2 = boto3.client('ec2')

# Get a list of available regions
regions = ec2.describe_regions()

# Create an empty list to store region data
region_data = []

# Iterate through the regions and retrieve endpoint and region name
for region in regions['Regions']:
    region_name = region['RegionName']
    endpoint = f'ec2.{region_name}.amazonaws.com'
    region_data.append([endpoint, region_name])

# Tabulate the region data
table = tabulate.tabulate(region_data, headers=['Endpoint', 'RegionName'], tablefmt='pretty')

# Print the tabulated data
print(table)
```

   Below the picture is the screenshot of result.

   We can see two columns information that are endpoint and regionname.

```
rongchuan@ubuntu:~/Desktop$ python3 6.py
+-----------------------------------+-----------------+
|              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     |
+-----------------------------------+-----------------+
```

Lab2

Create an EC2 instance using awscli

1. **Create a security group**

   As shown in the figure below, we ran the command to create a security group of 23715251-sg. And we can get the group id.

```
rongchuan@ubuntu:~$ aws ec2 create-security-group --group-name 23715251-sg --des
cription "security group for development environment"
{
    "GroupId": "sg-0e86e68494dce18d0"
}
```

2. **Authorise inbound traffic for ssh**

   This command will allow SSH connections from any IP address into the Amazon EC2 instance associated with the security group named 23715251-sg. Below is an screenshot of processing.

```
rongchuan@ubuntu:~$ aws ec2 authorize-security-group-ingress --group-name 237152
51-sg --protocol tcp --port 22 --cidr 0.0.0.0/0
{
    "Return": true,
    "SecurityGroupRules": [
        {
            "SecurityGroupRuleId": "sgr-06a41223b0c0e266d",
            "GroupId": "sg-0e86e68494dce18d0",
            "GroupOwnerId": "489389878001",
            "IsEgress": false,
            "IpProtocol": "tcp",
            "FromPort": 22,
            "ToPort": 22,
            "CidrIpv4": "0.0.0.0/0"
        }
    ]
}
```

3. **Create a key pair that will allow you to ssh to the EC2 instance**
   Command to create a key pair:
   aws ec2 create-key-pair --key-name rongchuan-key --query 'KeyMaterial' --output text > 23715251-key.pem
   We can see the key file of 23715251-key.pem after running the code.
   We can copy the key file to the directory of ~/.ssh by command cp.
   Finally, we can change the permission of the file by chmod 400.

```
rongchuan@ubuntu:~$ cp 23715251-key.pem ~/.ssh
rongchuan@ubuntu:~$ cd ~/.ssh
rongchuan@ubuntu:~/.ssh$ ls
23715251-key.pem  authorized_keys  known_hosts  known_hosts.old
rongchuan@ubuntu:~/.ssh$ cd ..
rongchuan@ubuntu:~$ ls
23715251-key.pem  Downloads   Music                    Public     Videos
Desktop           hello2.txt  note-to-flipper-pals.txt  snap
Documents         hello.txt   Pictures                 Templates
rongchuan@ubuntu:~$ cd ~/.ssh
rongchuan@ubuntu:~/.ssh$ chmod 400 23715251-key.pem
```

4. **Create the instance and note the instance id**
   By running the code from the below picture, we can create the instance and get the instance id.

```
rongchuan@ubuntu:~/.ssh$  aws ec2 run-instances --image-id ami-d38a4ab1 --securi
ty-group-ids 23715251-sg --count 1 --instance-type t2.micro --key-name 23715251-
key --query 'Instances[0].InstanceId'
"i-0fa4ae12836dc7390"
```

5. **Get the public IP address**
   We can get the IP address that is 52.62.22.37 by running the code in the below picture.

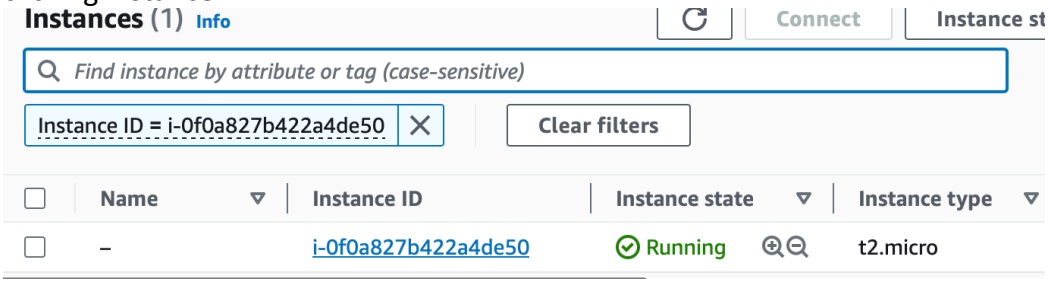
```
rongchuan@ubuntu:~/.ssh$ aws ec2 describe-instances --instance-ids i-0fa4ae12836
dc7390 --query 'Reservations[0].Instances[0].PublicIpAddress'
"52.62.22.37"
```

6. **Connect to the instance**
   We can connect the key file and the public IP address above. The flowing is processes.

```
rongchuan@ubuntu:~/.ssh$ ssh -i 23715251-key.pem ubuntu@52.62.22.37
The authenticity of host '52.62.22.37 (52.62.22.37)' can't be established.
ED25519 key fingerprint is SHA256:q/HOv8gwMMk7hlZk/1rpRMonyeeEyqszAFjP+ix06Fc.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '52.62.22.37' (ED25519) to the list of known hosts.
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-1052-aws x86_64)

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

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

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

7. **Look at the instance using the AWS console**

   Firstly, we need to login AWS and then choose compute in the service. And then click EC2, we can see dashboard of EC2. We can find the instance that we created after clicking instance.



Create an EC2 instance with Python Boto scrip

1. **Create a security group**

   Here's the Python Boto3 script to create a security group:

   Security group name is rongchuan from the below picture.

```python
import boto3

# Initialize the EC2 client
ec2 = boto3.client('ec2')

# Define the security group name and description
security_group_name = 'rongchuan'
security_group_description = 'security group for development environment'

# Create the security group
response = ec2.create_security_group(
    GroupName=security_group_name,
    Description=security_group_description
)

security_group_id = response['GroupId']
print(f"Security group {security_group_id} created.")
```

```
-- INSERT --                                                    1,2
```

After running the python script, we can get the group id.
Below is the screenshot of group id.

```
rongchuan@ubuntu:~/Desktop$ python3 ss.py
Security group sg-06ee12be5cf7282ef created.
```

2. **Authorise inbound traffic for ssh**

Below picture is python boto3 script to authorize inbound traffic for SSH(port 22) in a security group above.

```python
import boto3

# Initialize the EC2 client
ec2 = boto3.client('ec2')

# Define the security group ID and SSH rule parameters
security_group_id = 'sg-06ee12be5cf7282ef'
ssh_port = 22  # SSH port number

# Authorize inbound SSH traffic
ec2.authorize_security_group_ingress(
    GroupId=security_group_id,
    IpPermissions=[
        {
            'IpProtocol': 'tcp',
            'FromPort': ssh_port,
            'ToPort': ssh_port,
            'IpRanges': [{'CidrIp': '0.0.0.0/0'}]  # Allow SSH access from any IP
        }
    ]
)

print(f"Inbound SSH traffic (port {ssh_port}) authorized for security group {security_group_id}.")
```

The follow picture is the result.
We can see inbound SSH traffic(port 22) authorized successfully.

```
rongchuan@ubuntu:~/Desktop$ python3 2.py
Inbound SSH traffic (port 22) authorized for security group sg-06ee12be5cf7282ef.
```

3. **Create a key pair that will allow you to ssh to the EC2 instance**

Below code is a python script to create an EC2 key pair.

```python
import boto3

# Initialize the EC2 client
ec2 = boto3.client('ec2')

# Define the key pair name
key_pair_name = 'rongchuan'

# Create an EC2 key pair
response = ec2.create_key_pair(KeyName=key_pair_name)

# Save the private key to a file (you should protect this file)
private_key_file = '23715251-key.pem'
with open(private_key_file, 'w') as f:
    f.write(response['KeyMaterial'])

print(f"EC2 key pair '{key_pair_name}' created and private key saved to '{private_key_file}'.")
```

After running the python script, we can get the result below that indicates we have created the key pair successfully.

```
rongchuan@ubuntu:~/Desktop$ python3 3.py
EC2 key pair 'rongchuan' created and private key saved to '23715251-key.pem'.
```

4. **Create the instance and note the instance id**

Here's a python script to create an EC2 instance and note the instance id.
We can see the information instance type, image id, key pair name and security group id.

```python
import boto3

# Initialize the EC2 client
ec2 = boto3.client('ec2')

# Define the instance details
instance_type = 't2.micro'
image_id = 'ami-d38a4ab1'
key_pair_name = 'rongchuan'
security_group_ids = ['sg-06ee12be5cf7282ef']

# Launch the EC2 instance
response = ec2.run_instances(
    ImageId=image_id,
    InstanceType=instance_type,
    MinCount=1,
    MaxCount=1,
    KeyName=key_pair_name,
    SecurityGroupIds=security_group_ids,
    # You can specify additional parameters such as UserData, tags, etc.
)

# Extract the instance ID from the response
instance_id = response['Instances'][0]['InstanceId']

print(f"EC2 instance {instance_id} has been created.")
```

After running the code, we can get the instance id.
Below is the screenshot of the result.

```
rongchuan@ubuntu:~/Desktop$ python3 4.py
EC2 instance i-0610a184da6c5aade has been created.
```

5. **Get the public IP address**

Here's a python script to create the public address by instance id.

```python
import boto3

# Initialize the EC2 client
ec2 = boto3.client('ec2')

# Define the instance ID of the EC2 instance you want to retrieve the public IP address for
instance_id = 'i-0610a184da6c5aade'

# Use the describe_instances method to retrieve information about the instance
response = ec2.describe_instances(InstanceIds=[instance_id])

# Extract the public IP address from the response
public_ip_address = response['Reservations'][0]['Instances'][0]['PublicIpAddress']

print(f"The public IP address of the instance {instance_id} is {public_ip_address}")
```

Below is the result after running the script.

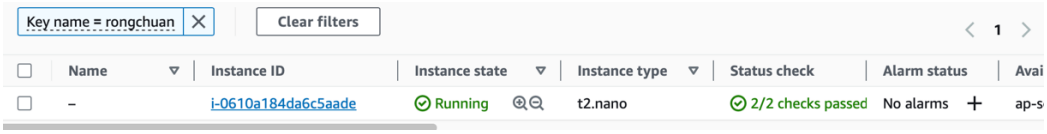
We can get the ip address that is 54.252.185.107

```
rongchuan@ubuntu:~/Desktop$ python3 5.py
The public IP address of the instance i-0610a184da6c5aade is 54.252.185.107
```

6. **Look at the instance using the AWS console**

After completing the above steps, we need to log in to AWS. We need to choose compute under the list of service. Then we choose the instances to view the instance we have created.

Below is the screenshot of the result.



Using Docker

1. **Install Docker**

Below the picture is to show how to install Docker in VM.

```
rongchuan@ubuntu:~$ sudo apt install docker.io
sudo: /etc/sudoers: Not a directory
[sudo] password for rongchuan:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
docker.io is already the newest version (20.10.21-0ubuntu1~22.10.2).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```

2. **Check the version**

We can see the version of docker by code below.

```
rongchuan@ubuntu:~$ docker --version
Docker version 20.10.21, build 20.10.21-0ubuntu1~22.10.2
```

3. **Build and run an httpd container**

Firstly, we create a directory of html by mkdir.

Then we need to create the file of index.html under the directory of html and add the content of the question.

Below is the processes.

```
rongchuan@ubuntu:~$ mkdir html
rongchuan@ubuntu:~$ ls
23715251-key.pem  Downloads   html                      Pictures  Templates
Desktop           hello2.txt  Music                     Public    Videos
Documents         hello.txt   note-to-flipper-pals.txt  snap
rongchuan@ubuntu:~$ cd html
rongchuan@ubuntu:~/html$ ls
rongchuan@ubuntu:~/html$ vim index.html
rongchuan@ubuntu:~/html$ ls
index.html
```

4. **Create a file called "Dockerfile" outside the html directory with the following content**

From the below picture, we can see the file of Dockerfile that has been created.

```
rongchuan@ubuntu:~/html$ cd ..
rongchuan@ubuntu:~$ vim Dockerfile
rongchuan@ubuntu:~$ ls
23715251-key.pem  Downloads   index.html                Public
Desktop           hello2.txt  Music                     snap
Dockerfile        hello.txt   note-to-flipper-pals.txt  Templates
Documents         html        Pictures                  Videos
```

5. **Build the docker image**

Below picture shows the process to build the image.

```
rongchuan@ubuntu:~$ docker build -t my-apache2 .
Sending build context to Docker daemon  457.4MB
Step 1/2 : FROM httpd:2.4
2.4: Pulling from library/httpd
4ee097f9a366: Pull complete
94662ba12854: Pull complete
595a05b7c0ad: Pull complete
03682c916abd: Pull complete
7e88a38e211a: Pull complete
Digest: sha256:333f7bca9fb72248f301fd2ae4892b86d36cc2fdf4c6aa49a6700f27c8e06daf
Status: Downloaded newer image for httpd:2.4
 ---> faf1e61d1928
Step 2/2 : COPY ./html/ /usr/local/apache2/htdocs/
 ---> a399ea34822a
Successfully built a399ea34822a
Successfully tagged my-apache2:latest
```

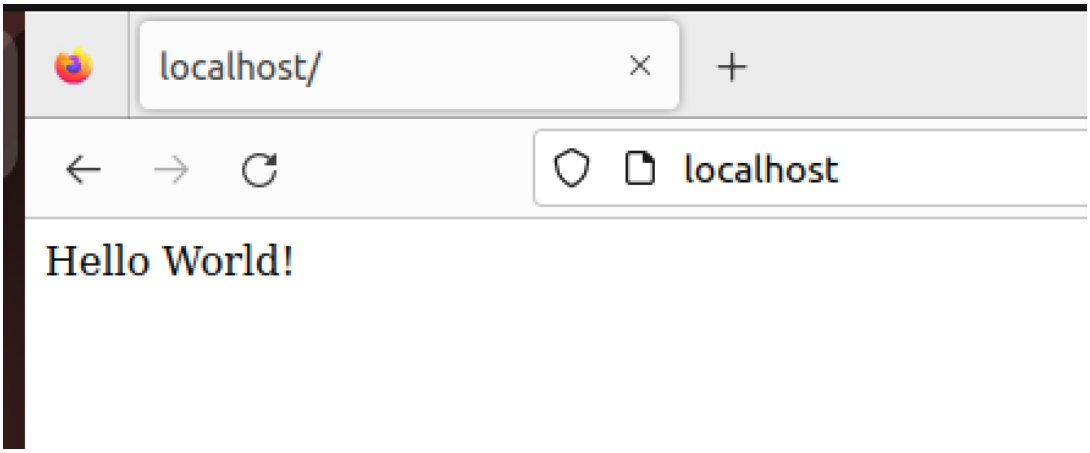
6. **Run the image**

Below is the result by running the image.

```
rongchuan@ubuntu:~$ docker run -p 80:80 -dit --name my-app my-apache2
e8fb31de1f1b8b87559de4fe96bf482fb8104ccbabae066268e585efbd645042
```

7. **Open a browser and access address http://localhost or http://127.0.0.1 Confirm you get Hello World!**

By searching the link http://localhost, we can get the result below.



8. **Other commands**

   By the following pictures, we can see the information of container and image.
   Then we stop the container of my-app.
   Finally, we delete the container of my-app.

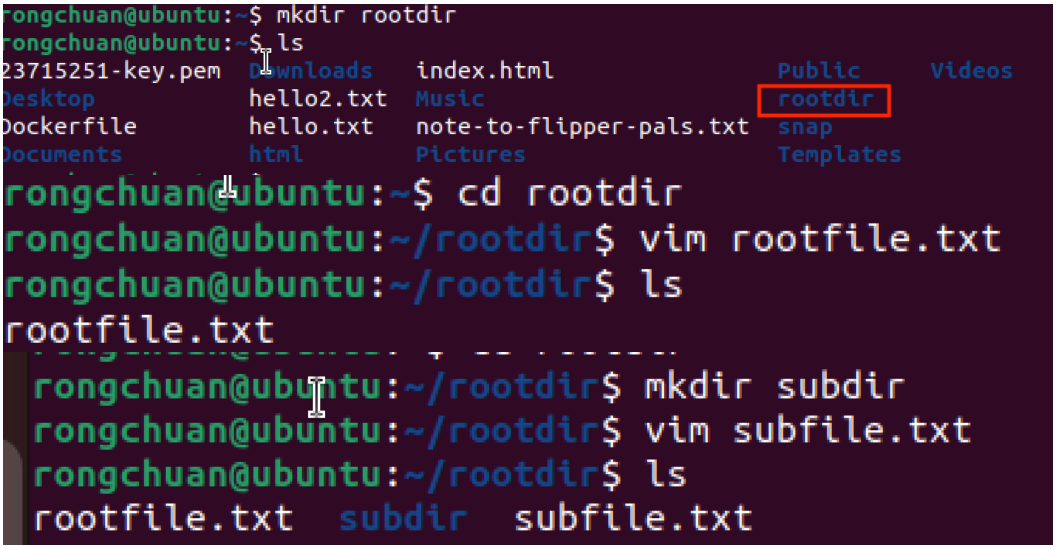
```
rongchuan@ubuntu:~$ docker ps -a
CONTAINER ID   IMAGE                          COMMAND                  CREATED
       STATUS                     PORTS                                      NAMES
e8fb31de1f1b   my-apache2                     "httpd-foreground"       About a minute
 ago   Up About a minute          0.0.0.0:80->80/tcp, :::80->80/tcp          my-app
716bcc8a50e6   uwacyber/cits1003-labs:bash    "/bin/bash"              5 months ago
       Exited (255) 5 months ago                                             interesti
ng_dewdney
484cad1688ca   uwacyber/cits1003-labs:bash    "/bin/bash"              6 months ago
       Exited (255) 5 months ago                                             dreamy_ma
xwell
7d1a6b1e241c   uwacyber/cits1003-labs:bash    "/bin/bash"              6 months ago
       Exited (0) 6 months ago                                               sleepy_va
ughan
667c17b109c1   uwacyber/cits1003-labs:bash    "/bin/bash"              6 months ago
       Exited (0) 6 months ago                                               happy_aus
tin
d31004629ed4   uwacyber/cits1003-labs:bash    "/bin/bash"              6 months ago
       Exited (0) 6 months ago                                               stupefied
_faraday
```

```
rongchuan@ubuntu:~$ docker stop my-app
my-app
rongchuan@ubuntu:~$ docker rm my-app
my-app
```

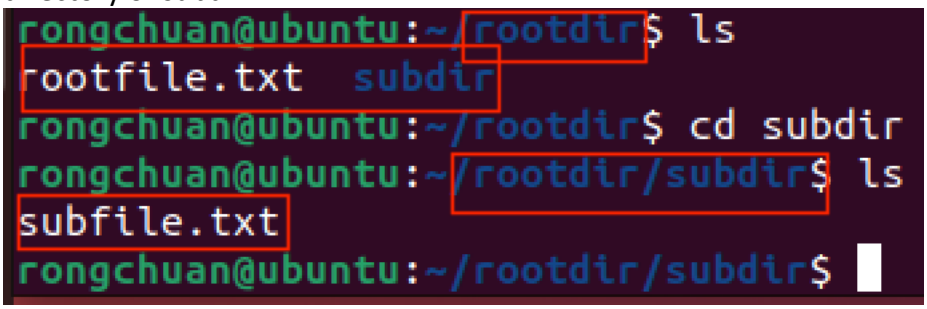
Lab3

1. **Preparation**

   Firstly, I use the command of “mkdir” to create the folder of “rootdir”. Then I need to use the command of “cd” into the rootdir. I need to create rootfile.txt , subfile.txt and the folder of subdir under the directory of rootdir. Finally, I need to move the subfile.txt from rootdir to subdir by the command of “cp”. The following pictures are processing.



The picture below is the result. There are a directory of subdir and a file of rootfile.txt under the directory of rootdir. There is a file of subfile.txt under the directory of subdir.

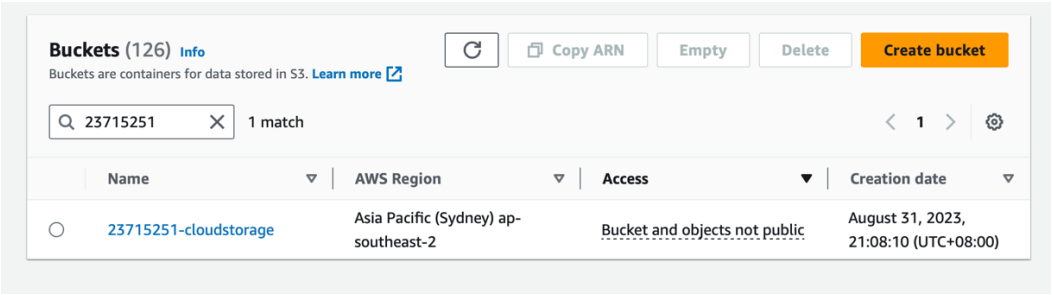


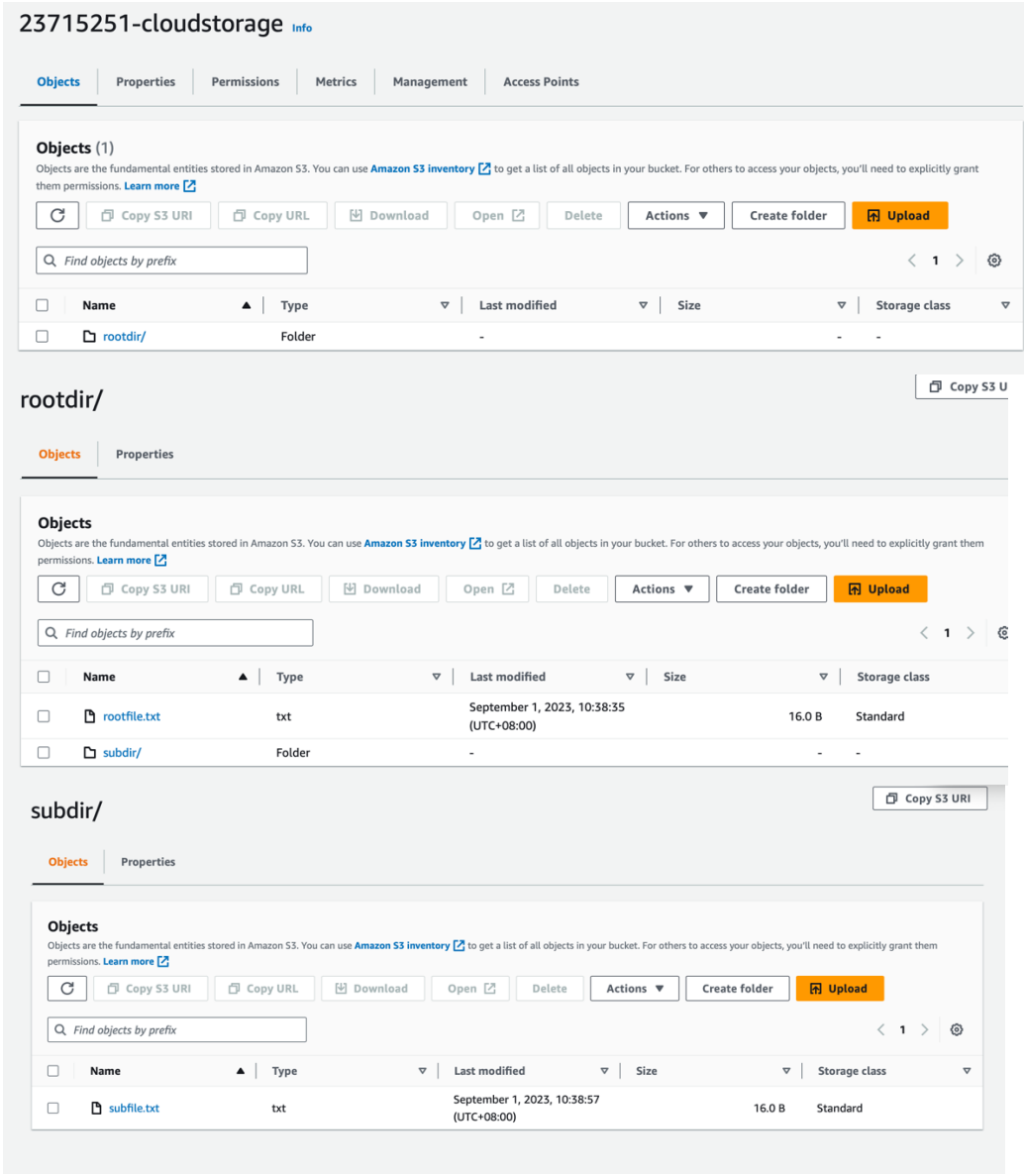
2. **Save to S3**

The picture below is the cloudstorage.py file I downloaded from GitHub, and then I modified the information and ran it to create my own bucket in the AWS bucket.

```
import os
import boto3
import base64

# ------------------------------
# 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 = '/home/rongchuan/rootdir'
ROOT_S3_DIR = '23715251-cloudstorage'


s3 = boto3.client("s3")

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

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


# Main program
# Insert code to create bucket if not there

try:

    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:
        for fname in file_list:
            upload_file("%s/" % dir_name[2:], "%s/%s" % (dir_name, fname), fname)


print("done")
```

The screenshots below are the created buckets. You can see that my bucket name is 23715251-cloudstorage, and there is a rootdir object in it. Then there are two objects in rootdir, subdir and rootfile.txt. Then there is an object in subdir which is subfile.txt





3. **Restore from S3**

   I created a python script named restorefromcloud.py. The following picture is the code. This script function can read the contents of the files in my bucket and then copy the contents to the file I specify.

```python
import os
import boto3

# AWS credentials and configuration setup
AWS_ACCESS_KEY_ID = "AKIAXD4PI5LYZECWG3FW"
AWS_SECRET_ACCESS_KEY = "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"
BUCKET_NAME = "23715251-cloudstorage"
REGION = "ap-southeast-2"
ROOT_DIR = "/home/rongchuan/rootdir/subdir"

# Create an S3 client
s3 = boto3.client("s3", aws_access_key_id=AWS_ACCESS_KEY_ID, aws_secret_access_key=AWS_SECRET_ACCESS_KEY, region_name=REGION)

# Function to restore files from S3
def restore_file(s3_key, local_path):
    try:
        s3.download_file(BUCKET_NAME, s3_key, local_path)
        print(f"Restored {s3_key} from S3 to {local_path}")
    except Exception as e:
        print(f"Error restoring {s3_key} from S3: {e}")

# Main program
# Parse directory and restore files from S3
for s3_object in s3.list_objects(Bucket=BUCKET_NAME)['Contents']:
    s3_key = s3_object['Key']
    local_path = os.path.join(ROOT_DIR, s3_key)
    os.makedirs(os.path.dirname(local_path), exist_ok=True)
    restore_file(s3_key, local_path)

print("done")
```

The pictures below are the results after I ran them. You can see that rootfile.txt and subfile.txt are copied from s3 to sudir. Then we can view the contents of files by cat command.

```
rongchuan@ubuntu:~/Desktop$ python3 restorefromcloud.py -i
Restored rootfile.txt from S3 to /home/rongchuan/rootdir/subdir/rootfile.txt
Restored subfile.txt from S3 to /home/rongchuan/rootdir/subdir/subfile.txt
done
rongchuan@ubuntu:~/rootdir$ cd subdir
rongchuan@ubuntu:~/rootdir/subdir$ ls
rootfile.txt  subfile.txt
rongchuan@ubuntu:~/rootdir/subdir$ cat rootfile.txt
1\n2\n3\n4\n5\n
rongchuan@ubuntu:~/rootdir/subdir$ cat subfile.txt
1\n2\n3\n4\n5\n
```

4. **Write information about files to DynamoDB**

Firstly, we need to create the folder of dynamodb. The below is a screenshot of processing.

```
rongchuan@ubuntu:~$ mkdir dynamodb
rongchuan@ubuntu:~$ ls
23715251-key.pem  Documents   hello.txt   note-to-flipper-pals.txt  rootdir
cloudstorage.py   Downloads   html        Pictures                  snap
Desktop           dynamodb    index.html  Public                    Templates
Dockerfile        hello2.txt  Music       python.py                 Videos
rongchuan@ubuntu:~$ cd dynamodb
```

Then we need to use commands to install java environment. We need to download the file of dynamobd_local_latest.tar.gz by wget https://s3-ap-northeast-1.amazonaws.com/dynamodb-local-tokyo/dynamodb_local_latest.tar.gz. Then we need to unzip this file to get DynamoDBLocal.jar

java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar –sharedDb is be used to run the DynamoDBLocal.jar

The following pictures are processes.

```
rongchuan@ubuntu:~/dynamodb$ sudo apt-get install default-jre
sudo: /etc/sudoers: Not a directory
[sudo] password for rongchuan:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  ca-certificates-java default-jre-headless fonts-dejavu-extra java-common
  libatk-wrapper-java libatk-wrapper-java-jni openjdk-11-jre openjdk-11-jre-headless
Suggested packages:
  fonts-ipafont-gothic fonts-ipafont-mincho fonts-wqy-microhei | fonts-wqy-zenhei
The following NEW packages will be installed:
  ca-certificates-java default-jre default-jre-headless fonts-dejavu-extra java-commo
  libatk-wrapper-java libatk-wrapper-java-jni openjdk-11-jre openjdk-11-jre-headless
0 upgraded, 9 newly installed, 0 to remove and 0 not upgraded.
Need to get 44.0 MB of archives.
```

```
rongchuan@ubuntu:~$ cd dynamodb
rongchuan@ubuntu:~/dynamodb$ ls
DynamoDBLocal.jar             DynamoDBLocal_lib  README.txt
dynamodb_local_latest.tar.gz  LICENSE.txt        THIRD-PARTY-LICENSES.txt
```

We need to create a table in my DynamoDB, the code screenshot below is me creating a table called CloudFiles in DynamoDB.

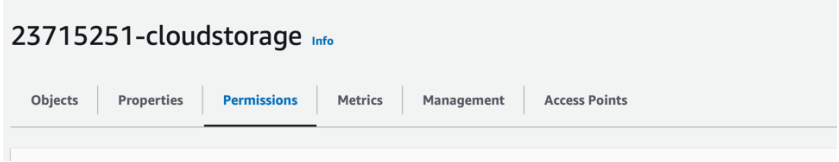
```
import boto3

# Create a DynamoDB client using the local DynamoDB endpoint
dynamodb = boto3.client('dynamodb', endpoint_url='http://localhost:8000')

# define table properties
table_name = 'CloudFiles'
attribute_definitions = [
     {'AttributeName': 'userId', 'AttributeType': 'S'},
     {'AttributeName': 'fileName', 'AttributeType': 'S'},
     {'AttributeName': 'path', 'AttributeType': 'S'},
     {'AttributeName': 'lastUpdated', 'AttributeType': 'S'},
     {'AttributeName': 'owner', 'AttributeType': 'S'},
     {'AttributeName': 'permissions', 'AttributeType': 'S'}
]

# Define primary key and non-primary key attributes
key_schema = [
     {'AttributeName': 'userId', 'KeyType': 'HASH'}
]

# create table
dynamodb.create_table(
     TableName=23715251,
     AttributeDefinitions = attribute_definitions,
     KeySchema=key_schema,
     ProvisionedThroughput = {
         'ReadCapacityUnits': 5,
         'WriteCapacityUnits': 5
     }
)

# Wait for the table to be created
waiter = dynamodb. get_waiter('table_exists')
waiter.wait(TableName=table_name)
print("Table created:", table_name)
```

The following code screenshot is to store the file information in S3 into the table I have created in DynamoDB

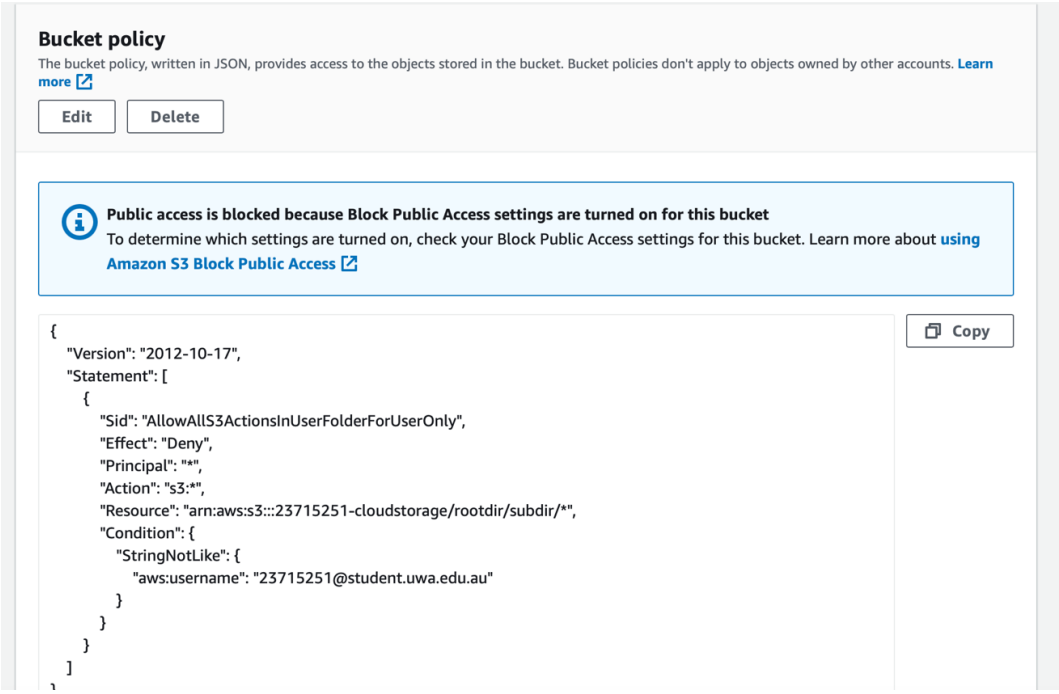
```
import boto3
import os
from datetime import datetime

# Create a DynamoDB client using the local DynamoDB endpoint
dynamodb = boto3.client('dynamodb', endpoint_url='http://localhost:8000')

# Function to obtain S3 file information, here assumes that you already have an S3 client s3_client
def get_s3_file_info(bucket_name):
     files = []
     response = s3_client.list_objects_v2(Bucket=bucket_name)
     for item in response.get('Contents', []):
         file_info = {
             'fileName': item['Key'],
             'path': os.path.dirname(item['Key']),
             'lastUpdated': item['LastModified'].strftime('%Y-%m-%d %H:%M:%S'),
             'owner': 'unknown', # set file owner
             'permissions': 'unknown' # Set file permissions
         }
         files.append(file_info)
     return files

# Get information about files in S3
s3_client = boto3.client('s3')
bucket_name = '23715251-cloudstorage'
files_info = get_s3_file_info(bucket_name)

# Write information to DynamoDB table
table_name = 'CloudFiles'

for file_info in files_info:
     item = {
         'userId': 23715251,
         'fileName': rongchuan,
         'path': rongchuan/dynamodb,
         'lastUpdated': file_info['lastUpdated'],
         'owner': file_info['owner'],
         'permissions': file_info['permissions']
     }
     dynamodb.put_item(TableName=table_name, Item=item)

print("Items added to DynamoDB table:", len(files_info))
```

Lab4

1. Apply policy to restrict permissions on bucket

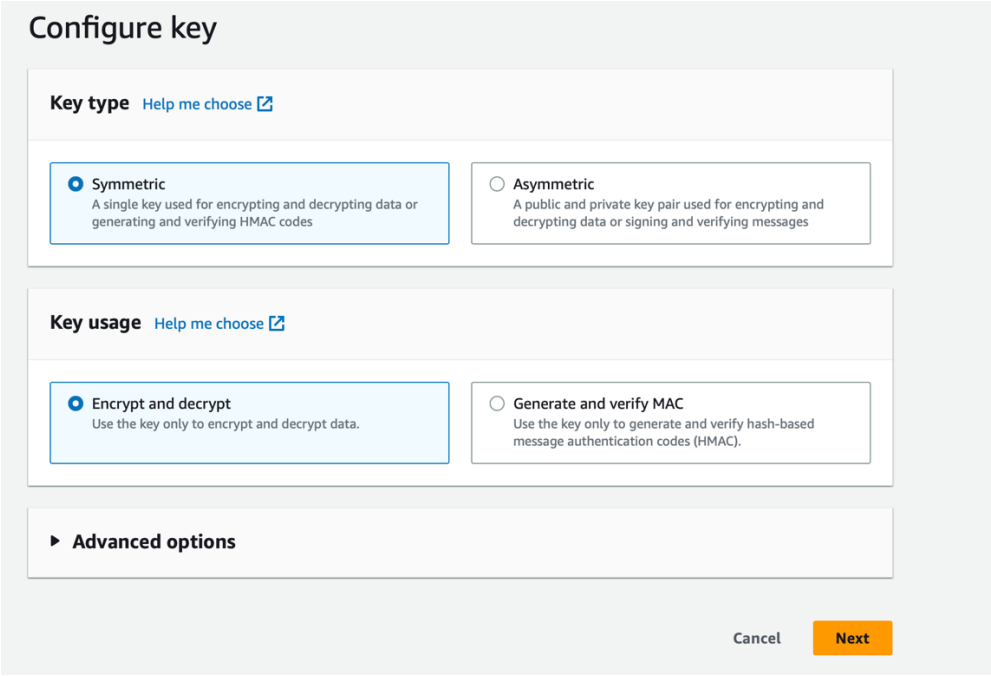


First, you need to log in to the bucket control page of AWS, as shown in the figure above. Then click permissions, modify the experimental code and copy it to the bucket policy code box, and then click Save, as shown in the figure below.

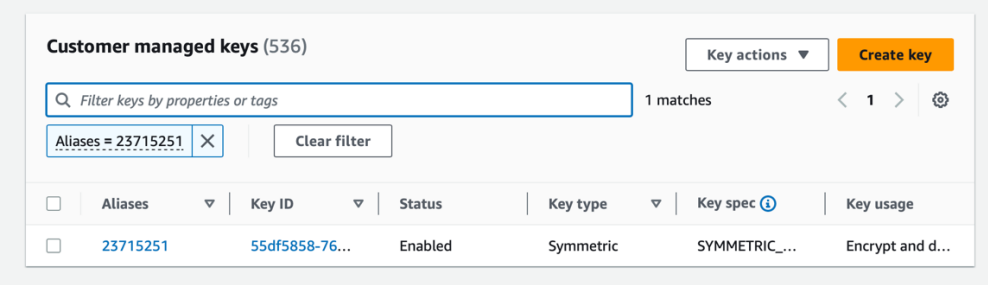


2. AES Encryption using KMS

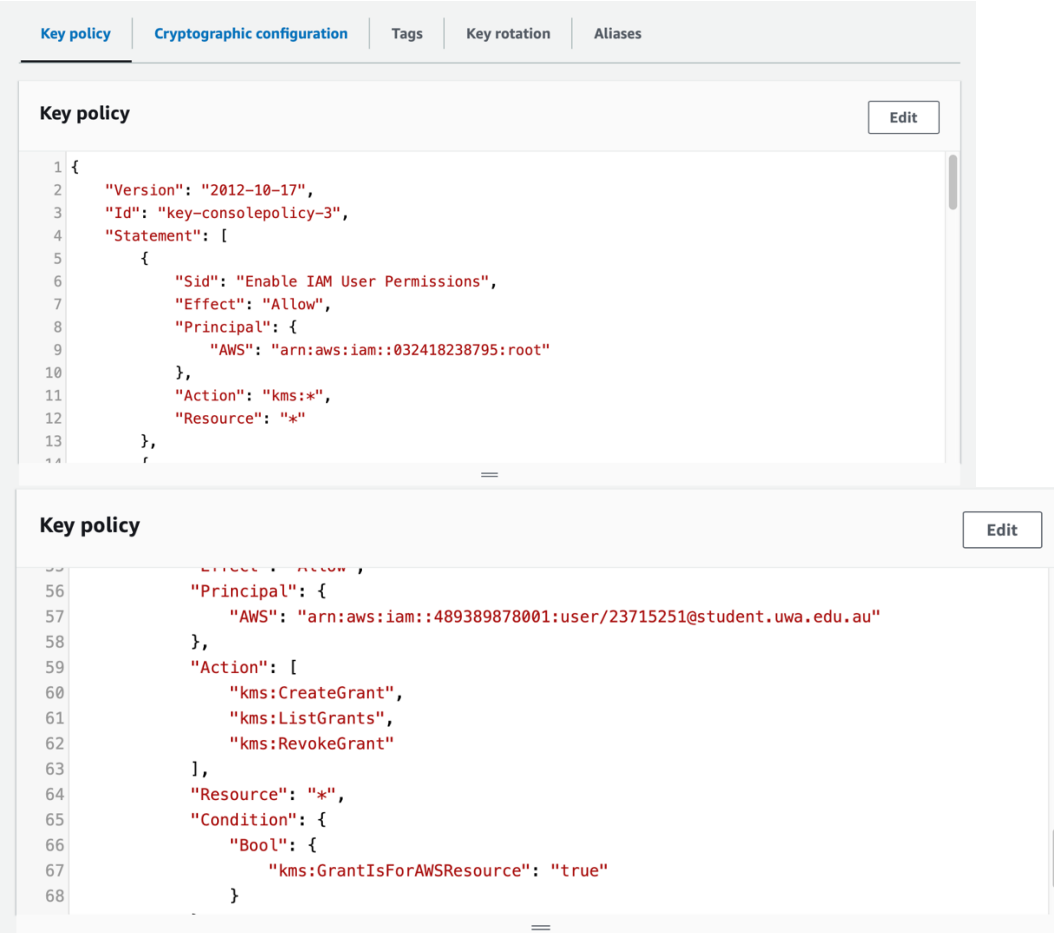
The first step is to log in to the kms control page of AWS, and then choose to create a key, as shown in the figure below.



Then select the following steps as required, and finally a pair of keys will be created, as shown in the picture below.



Next we need to modify the key policy. We need to modify the code in the question and paste it into the key policy box, then save it. In this way we have created the encryption key, as shown in the figure below.



3. **AES Encryption using local python library pycryptodome**

I wrote a python script based on the code provided. Below is a screenshot of the code. The code is capable of AES encryption.

code explanation:

The code defines functions to encrypt and decrypt files using AES encryption in Cipher Block Chaining (CBC) mode. It generates an encryption key from the provided password, randomly generates an initialization vector (IV), and processes the input file in chunks. The encrypted file is saved with the .enc extension. The decryption process uses the provided password to generate the decryption key, reads the original file size and IV from the encrypted file, and decrypts the content back to its original state. The decrypted file is saved with the "afilenew_dec.txt" filename.

```
import os
import struct
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
import hashlib

BLOCK_SIZE = 16
CHUNK_SIZE = 64 * 1024

def encrypt_file(password, in_filename, out_filename):
    # Generate encryption key from password
    key = hashlib.sha256(password.encode("utf-8")).digest()

    # Generate a random initialization vector (IV)
    iv = get_random_bytes(AES.block_size)

    # Create AES encryptor in Cipher Block Chaining (CBC) mode
    encryptor = AES.new(key, AES.MODE_CBC, iv)

    # Get the size of the input file
    filesize = os.path.getsize(in_filename)

    # Open input and output files
    with open(in_filename, 'rb') as infile:
        with open(out_filename, 'wb') as outfile:
            # Write the original file size and IV to the output file
            outfile.write(struct.pack('<Q', filesize))
            outfile.write(iv)

            # Encrypt and write file content in chunks
            while True:
                chunk = infile.read(CHUNK_SIZE)
                if len(chunk) == 0:
                    break
                elif len(chunk) % 16 != 0:
                    chunk += b' ' * (16 - len(chunk) % 16)

                outfile.write(encryptor.encrypt(chunk))


def decrypt_file(password, in_filename, out_filename):
    # Generate decryption key from password
    key = hashlib.sha256(password.encode("utf-8")).digest()

    # Open input file
    with open(in_filename, 'rb') as infile:
        # Read original file size and IV from the input file
        origsize = struct.unpack('<Q', infile.read(struct.calcsize('Q')))[0]
        iv = infile.read(16)

        # Create AES decryptor in CBC mode
        decryptor = AES.new(key, AES.MODE_CBC, iv)

        # Open output file
        with open(out_filename, 'wb') as outfile:
            # Decrypt and write file content in chunks
            while True:
                chunk = infile.read(CHUNK_SIZE)
                if len(chunk) == 0:
                    break
                outfile.write(decryptor.decrypt(chunk))

            # Truncate output file to the original size
            outfile.truncate(origsize)


# Main program
password = 'kitty and the kat'

# Encrypt the file "afile1_dec.txt" and save it as "afile1_dec.txt.enc"
encrypt_file(password, "afile1_dec.txt", out_filename="afile1_dec.txt.enc")

# Decrypt the encrypted file "afile1_dec.txt.enc" and save it as "afilenew_dec.txt"
decrypt_file(password, "afile1_dec.txt.enc", out_filename="afilenew_dec.txt")
```

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

- KMS: Managed KMS services are optimized for performance and scalability. These services are designed to handle encryption and decryption operations efficiently.
- Custom Solution: The performance of a custom solution depends on factors such as the encryption algorithm, key size, and implementation details. While libraries like **pycryptodome** can provide good performance, they might not be as optimized as cloud provider KMS services.