

Lab 1

Installing UTM and Kali Linux

Step 1:

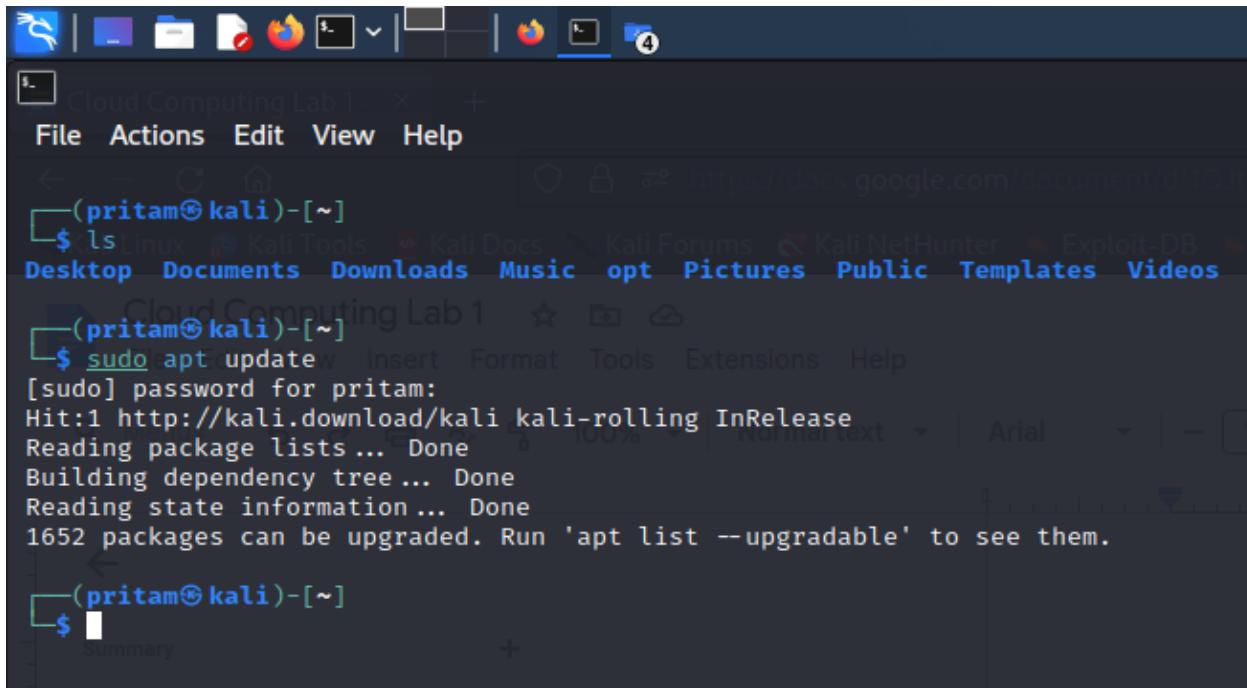
Since my OS was macOS with an M1 chip, I followed the instructions provided in the link below. <https://www.kali.org/docs/virtualization/install-utm-guest-vm/> to set up UTM and Kali Linux on my machine.

UTM is a full-featured system emulator and virtual machine host for iOS and macOS. It is based on QEMU (Quick Emulator) which is a free and open-source emulator. In short, it allows you to run Windows, Linux, and more on your Mac, iPhone, and iPad. More information at <https://getutm.app/> and <https://mac.getutm.app/>

Step 2:

Updated the OS with the following command to make sure all the system packages are updated. Running `sudo apt update` fetches the latest package information for software repositories on your Linux system, ensuring security patches, bug fixes, and new features are available. This step is vital before installing or upgrading software with accurate dependencies. It maintains system security, stability, and functionality.

```
sudo apt update
```

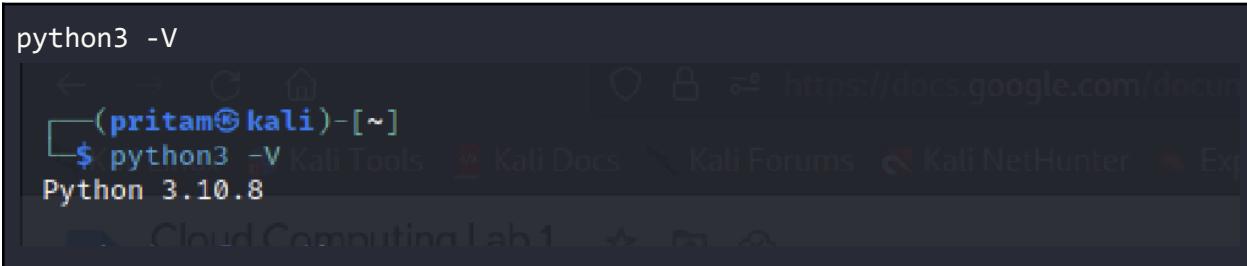


```
(pritam㉿kali)-[~]
$ ls
Desktop Documents Downloads Music opt Pictures Public Templates Videos
(pritam㉿kali)-[~]
$ sudo apt update
[sudo] password for pritam:
Hit:1 http://kali.download/kali kali-rolling InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
1652 packages can be upgraded. Run 'apt list --upgradable' to see them.

(pritam㉿kali)-[~]
$
```

Step 3:

Checked Python version with the following command.

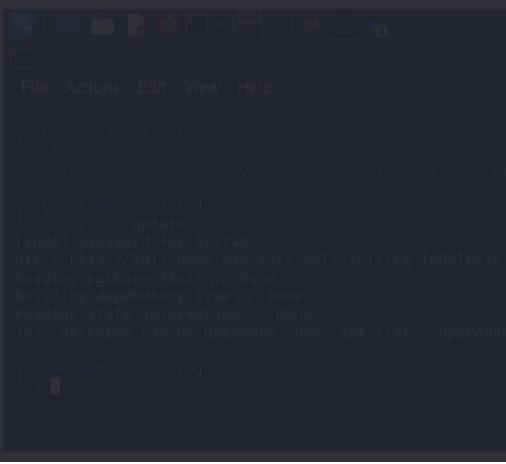


```
python3 -V
(pritam㉿kali)-[~]
$ python3 -V
Python 3.10.8
```

Step 4:

Since the Python version was 3.10.8, I installed pip3 which is a Python package manager. `pip3` is a package installer for Python, allowing you to easily download, manage, and install external Python packages. It's essential as it simplifies adding libraries, tools, and modules, enhancing functionality and saving development time. With `pip3`, you can efficiently expand Python capabilities and share code dependencies.

Note: The usage of `pip3` instead of `pip` is mainly relevant when working with multiple versions of Python installed on your system. `pip3` is associated with Python 3, ensuring that packages are installed for the correct Python version. Using `pip` alone could refer to the default `pip` associated with Python 2, potentially causing compatibility issues if you're primarily using Python 3. It's a way to maintain version-specific package management.



```
pritam@kali:~
```

File Actions Edit View Help

```
(pritam㉿kali)-[~]
$ sudo apt install -y python3-pip
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed: python3-pip
python3-pip
0 upgraded, 1 newly installed, 0 to remove and 1649 not upgraded.
Need to get 1,353 kB of archives.
After this operation, 6,956 kB of additional disk space will be used.
Get:1 http://http.kali.org/kali kali-rolling/main arm64 python3-pip all 23.2+dfsg-1 [1,353 kB]
Fetched 1,353 kB in 1s (982 kB/s)
Selecting previously unselected package python3-pip.
(Reading database ... 394732 files and directories currently installed.)
Preparing to unpack .../python3-pip_23.2+dfsg-1_all.deb ...
Unpacking python3-pip (23.2+dfsg-1) ...
Setting up python3-pip (23.2+dfsg-1) ...
Processing triggers for man-db (2.11.0-1+b1) ...
Processing triggers for kali-menu (2022.4.1) ...
Scanning processes ...
Scanning processor microcode ...
Scanning linux images ...
Running kernel seems to be up-to-date.

Failed to check for processor microcode upgrades.

No services need to be restarted.

No containers need to be restarted.

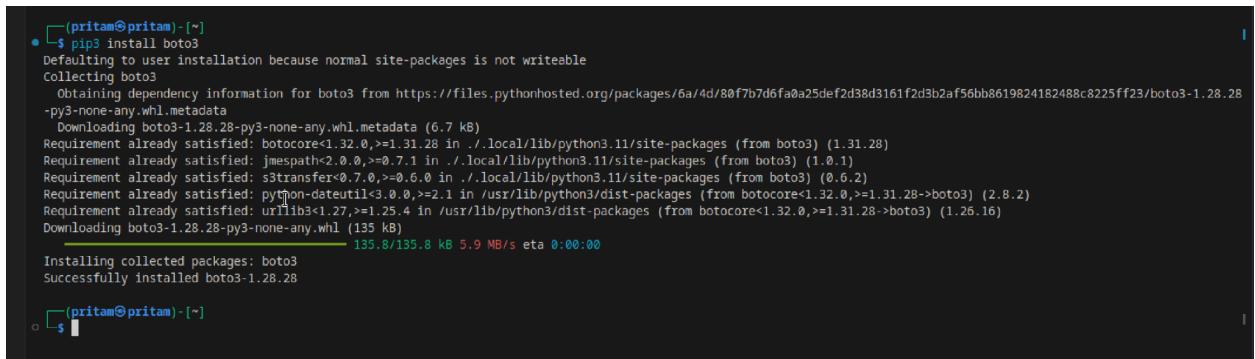
No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.

(pritim@kali)-[~]
$
```

Step 5:

Installed boto3 which is a Python SDK for AWS. `boto3` is a Python library developed by Amazon Web Services (AWS) that provides an interface to interact with AWS services programmatically. It allows developers to write Python code to automate and manage various AWS services, such as Amazon S3 (Simple Storage Service), Amazon EC2 (Elastic Compute Cloud), Amazon DynamoDB, and more. With `boto3`, you can create, configure, and manage AWS resources, and perform actions like uploading files to S3, launching EC2 instances, and accessing data stored in AWS services.



```
pritam@pritam:~
```

```
* → pip3 install boto3
Defaulting to user installation because normal site-packages is not writable
Collecting boto3
  Obtaining dependency information for boto3 from https://files.pythonhosted.org/packages/6a/4d/80f7b7d6fa0a25def2d38d3161f2d3b2af56bb8619824182488c8225ff23/boto3-1.28.28-py3-none-any.whl.metadata (6.7 kB)
    Downloading boto3-1.28.28-py3-none-any.whl.metadata (6.7 kB)
Requirement already satisfied: botocore<1.32.0,>=1.31.28 in ./local/lib/python3.11/site-packages (from boto3) (1.31.28)
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in ./local/lib/python3.11/site-packages (from boto3) (1.0.1)
Requirement already satisfied: s3transfer<0.7.0,>=0.6.0 in ./local/lib/python3.11/site-packages (from boto3) (0.6.2)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/lib/python3/dist-packages (from botocore<1.32.0,>=1.31.28->boto3) (2.8.2)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/lib/python3/dist-packages (from botocore<1.32.0,>=1.31.28->boto3) (1.26.16)
  Downloading boto3-1.28.28-py3-none-any.whl (135 kB)
    135.8/135.8 kB 5.9 MB/s eta 0:00:00
  Installing collected packages: boto3
    Successfully installed boto3-1.28.28
```

```
(pritam@pritam)-[~]
$
```

Step 6:

Installed **awscli** which is a command line interface to interact with AWS. "AWS CLI," stands for Amazon Web Services Command Line Interface. It is a command-line tool provided by AWS that allows users to interact with AWS services using commands in the terminal or command prompt. The AWS CLI enables you to manage and configure AWS resources, and perform tasks such as creating and managing EC2 instances, S3 buckets, and more, all from the command line. It provides an alternative to using the AWS Management Console web interface for interacting with AWS services.

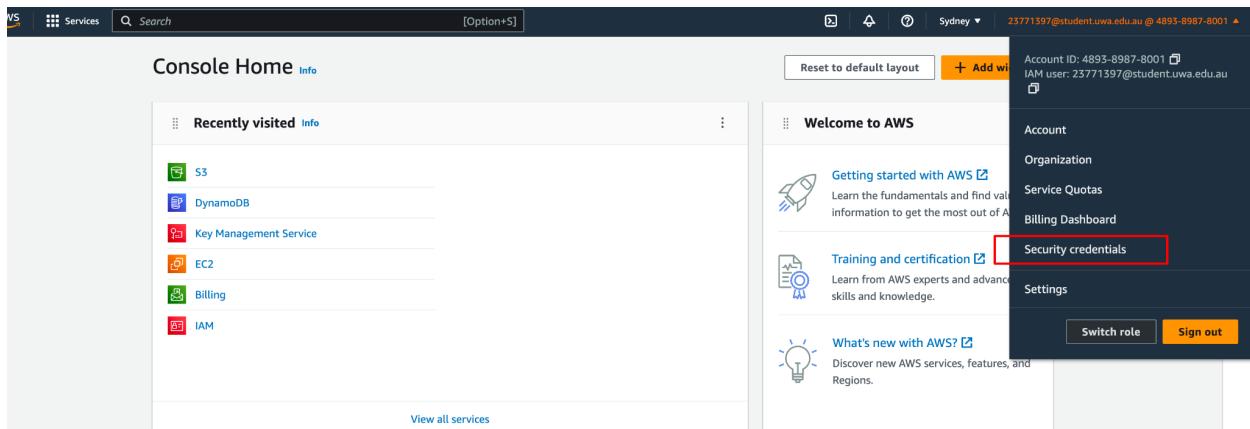


```
(pritam@kali) ~]$ pip install awscli --upgrade
Defaulting to user installation because normal site-packages is not writeable
Collecting awscli
  Obtaining dependency information for awscli from https://files.pythonhosted.org/packages/dc/4b/43978f75c46b5df4a5e/b24d94540e9f930cb95c12c05f64fabdce6138/awscli-1.29.18-py3-none-any.whl.metadata
    Using cached awscli-1.29.18-py3-none-any.whl.metadata (1.4 kB)
Requirement already satisfied: botocore==1.31.18 in ./local/lib/python3.10/site-packages (from awscli (1.31.18))
Requirement already satisfied: docutils<0.17, >0.16.0 in ./local/lib/python3.10/site-packages (from awscli (0.16))
Requirement already satisfied: s3transfer<0.7, >=0.6.0 in ./local/lib/python3.10/site-packages (from awscli (0.6.1))
Requirement already satisfied: rsa<4.5, >=3.1.2 in ./local/lib/python3.10/site-packages (from awscli (3.1.2))
Requirement already satisfied: colorama<0.5.5, >=0.2.5 in ./local/lib/python3.10/site-packages (from awscli (0.4.4))
Requirement already satisfied: rsa<4.8, >=3.1.2 in ./local/lib/python3.10/site-packages (from awscli (4.7.2))
Requirement already satisfied: mephisto<2.0.0, >=0.7.1 in ./local/lib/python3.10/site-packages (from botocore==1.31.18->awscli) (2.0.2)
Requirement already satisfied: s3transfer<0.7, >=0.6.0 in ./local/lib/python3.10/site-packages (from botocore==1.31.18->awscli) (0.7.0)
Requirement already satisfied: pyasn1 >=0.1.3 in /usr/lib/python3/dist-packages (from botocore==1.31.18->awscli) (0.26.12)
Requirement already satisfied: pyasn1 >=0.1.3 in /usr/lib/python3/dist-packages (from rsa<4.8, >=3.1.2->awscli) (0.4.8)
Using cached awscli-1.29.18-py3-none-any.whl (1.4 kB)
WARNING: python-apt 2.0.0 minus b1 has a non-standard version number. pip 23.3 will enforce this behaviour change. A possible replacement is to upgrade to a newer version of python-apt or contact the author to suggest that they release a version with a conforming version number. Discussion can be found at https://github.com/pypa/pip/issues/1206.
Installing collected packages: awscli
Successfully installed awscli-1.29.18
[1]
```

Steps to create access key using AWS Management Console

Step 6a:

Click on the Security Credentials tab to open security details page.



The screenshot shows the AWS Management Console Home page. On the left, there's a sidebar with 'Recently visited' links for S3, DynamoDB, Key Management Service, EC2, Billing, and IAM. The main area features a 'Welcome to AWS' section with links for 'Getting started with AWS', 'Training and certification', and 'What's new with AWS?'. On the right, there's a navigation bar with 'Account ID: 4893-8987-8001', 'IAM user: 23771397@student.uwa.edu.au', and a sign-in status. Below the navigation bar, there are links for 'Account', 'Organization', 'Service Quotas', 'Billing Dashboard', and 'Settings'. The 'Security credentials' link under 'Settings' is highlighted with a red box.

Step 6b:

Choose 'Create Access key'

Access keys (1)

Use access keys to send programmatic calls to AWS from the AWS CLI, AWS Tools for PowerShell, AWS SDKs, or direct AWS API calls. You can have a maximum of two access keys (active or inactive) at a time. [Learn more](#)

Create access key

AKIAJD4PI5LY7YTII7FO	
Description	Status
CITS5503-AWS-LABS	Active
Last used	Created
20 hours ago	14 days ago
Last used region	Last used service
ap-southeast-2	S3

Actions ▾

Step 6c:

Select CLI option to use access key for awscli

Use case

Command Line Interface (CLI)
You plan to use this access key to enable the AWS CLI to access your AWS account.

Local code
You plan to use this access key to enable application code in a local development environment to access your AWS account.

Application running on an AWS compute service
You plan to use this access key to enable application code running on an AWS compute service like Amazon EC2, Amazon ECS, or AWS Lambda to access your AWS account.

Third-party service
You plan to use this access key to enable access for a third-party application or service that monitors or manages your AWS resources.

Application running outside AWS
You plan to use this access key to enable an application running on an on-premises host, or to use a local AWS client or third-party AWS plugin.

Other
Your use case is not listed here.

Alternatives recommended

- Use [AWS CloudShell](#), a browser-based CLI, to run commands. [Learn more](#)
- Use the [AWS CLI V2](#) and enable authentication through a user in IAM Identity Center. [Learn more](#)

Confirmation

I understand the above recommendation and want to proceed to create an access key.

Cancel **Next**

Step 6d:

Add tag to remember and manage access key

Set description tag - optional Info

The description for this access key will be attached to this user as a tag and shown alongside the access key.

Description tag value

Describe the purpose of this access key and where it will be used. A good description will help you rotate this access key confidently later.

Maximum 256 characters. Allowed characters are letters, numbers, spaces representable in UTF-8, and: _ . : / = + - @

[Cancel](#)[Previous](#)[Create access key](#)

Step 6e:

Download .csv file for future reference.

Retrieve access keys Info

Access key

If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make the old key inactive.

Access key

Secret access key

AKIAXD4P15LYZD3K5A4P

 ***** [Show](#)

Access key best practices

- Never store your access key in plain text, in a code repository, or in code.
- Disable or delete access key when no longer needed.
- Enable least-privilege permissions.
- Rotate access keys regularly.

For more details about managing access keys, see the [best practices for managing AWS access keys](#).

[Download .csv file](#)[Done](#)

Step 7:

Configured AWS CLI with Access key and Secret key. Access Key ID acts as username and secret Key acts as password for the aws cli. These should never be shared and stored securely. The command `aws configure` is used to set up the configuration for the AWS CLI (Command Line Interface) on our local machine. When we run this command, we be prompted to provide our AWS Access Key ID, Secret Access Key, default region, and output format preferences. These credentials are used by the AWS CLI to authenticate your interactions with AWS services. Once configured, we can use the AWS CLI to manage and interact with your AWS resources using command-line commands.

```
(pritam㉿kali)-[~]
$ aws configure
AWS Access Key ID [None]: [REDACTED]
AWS Secret Access Key [None]: [REDACTED]
Default region name [None]: ap-southeast-2
Default output format [None]: json

(pritam㉿kali)-[~]
$ [REDACTED]
```

Step 8:

Tested the AWS environment by running:

Using the command `aws ec2 describe-regions --output table` in your AWS environment retrieves a list of available regions for Amazon EC2 instances and displays them in a tabular format. This command provides us with information about the various regions where we can deploy our EC2 instances, helping us understand the geographical options for hosting our resources. It's a useful way to explore and test your AWS environment's regional availability using the AWS CLI.

```
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
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
ec2.eu-west-1.amazonaws.com	opt-in-not-required	eu-west-1
ec2.ap-northeast-3.amazonaws.com	opt-in-not-required	ap-northeast-3
ec2.ap-northeast-2.amazonaws.com	opt-in-not-required	ap-northeast-2
ec2.ap-northeast-1.amazonaws.com	opt-in-not-required	ap-northeast-1
ec2.ca-central-1.amazonaws.com	opt-in-not-required	ca-central-1
ec2.sa-east-1.amazonaws.com	opt-in-not-required	sa-east-1
ec2.ap-southeast-1.amazonaws.com	opt-in-not-required	ap-southeast-1
ec2.ap-southeast-2.amazonaws.com	opt-in-not-required	ap-southeast-2
ec2.eu-central-1.amazonaws.com	opt-in-not-required	eu-central-1
ec2.us-east-1.amazonaws.com	opt-in-not-required	us-east-1
ec2.us-east-2.amazonaws.com	opt-in-not-required	us-east-2
ec2.us-west-1.amazonaws.com	opt-in-not-required	us-west-1
ec2.us-west-2.amazonaws.com	opt-in-not-required	us-west-2

In AWS, an "endpoint" refers to the URL or network address that we use to access a specific AWS service within a particular region. Each AWS service has its own endpoint in each region.

The endpoint is used to make API requests and interact with that specific service in the chosen region.

A "region" in AWS is a geographic area where AWS has data centers. Each region is composed of multiple availability zones, which are essentially separate data centers within that region. AWS offers a variety of services in each region, and resources (like EC2 instances, S3 buckets, etc.) are provisioned within a chosen region.

Step 9:

Tested python environment by running the following in python interpreter.

```
python3
>>> import boto3
>>> ec2 = boto3.client('ec2')
>>> response = ec2.describe_regions()
>>> print(response)
```

```
ec2 = boto3.client('ec2'):
```

This line creates an EC2 (Elastic Compute Cloud) client using the boto3 library. The ec2 object will be used to make API requests to EC2 services.

```
response = ec2.describe_regions():
```

This line sends an API request to the EC2 service to retrieve information about available regions. The describe_regions() method returns a response containing details about the regions.

This will create an un-tabulated response which can be formatted by writing a Python script. I tabulated the output to have 2 columns with Endpoint and RegionName. I installed the tabulate library for formatting using the following command.

```
pip3 install tabulate
```

```
tabulate-regions.py X __init__.py 1

tabulate-regions.py > ...
1 import boto3
2
3 from tabulate import tabulate
4
5 ec2 = boto3.client('ec2')
6
7 response = ec2.describe_regions()
8
9 # print(response)
10
11 table = []
12
13
14 for region in response['Regions']:
15     table.append([region['Endpoint'], region['RegionName']])
16
17
18 print(tabulate(table, headers=[ "Endpoint", "RegionName"]))
```

Output of the above python script.

```
+ ... zsh + ... zsh
... zsh
(aws-env)-(pritam@kali)-[~/Desktop/aws-labs]
● $ python3 tabulate-regions.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

(aws-env)-(pritam@kali)-[~/Desktop/aws-labs]
○ $
```

Lab 2:

Create an EC2 instance using awscli

Step 1: Create a security group

An AWS security group is a virtual firewall that controls inbound and outbound traffic to and from AWS resources. It operates based on rule-based access control, where you define rules specifying allowed or denied traffic based on IP address ranges and protocols. They are associated with AWS instances and provide instance-level security, allowing us to customize rules for different instances. By default, security groups have no inbound rules, and they employ an implicit deny rule for traffic types not explicitly allowed, helping enhance the security of your AWS resources.

```
aws ec2 create-security-group --group-name <student number>-sg  
--description "security group for development environment"
```

Here's what each part of the command does:

- **aws**: This is the AWS Command Line Interface executable.
- **ec2**: This specifies that we are working with Amazon EC2, which is the Elastic Compute Cloud service for managing virtual servers in the AWS cloud.
- **create-security-group**: This is the command to create a new security group.
- **--group-name <student number>-sg**: This is where we should replace `<student number>` with an actual value, like our student number or any unique identifier. It's used as the name of the security group we're creating. The `-sg` suffix indicates that it's a security group.
- **--description "security group for development environment"**: This is where you provide a description for the security group you're creating. It's a human-readable explanation of what the security group is used for.

Putting it all together, the command is used to create a new security group in Amazon EC2 with a specific name and description. Security groups act as virtual firewalls for your instances, controlling inbound and outbound traffic. They are used to define rules that allow or deny traffic based on protocols, ports, and IP ranges.

Output: The **GroupId** is a unique identifier assigned by AWS to represent a security group, which is a virtual firewall controlling inbound and outbound traffic for Amazon EC2 instances.

```
(aws-env)-(pritam@kali)-[~/Desktop/aws-labs]
$ aws ec2 create-security-group --group-name 23771397-sg --description "security group for development environment"
{
    "GroupId": "sg-0867bc0fa7bf1a395"
```

Step 2: Authorise inbound traffic for ssh

```
aws ec2 authorize-security-group-ingress --group-name <student number>-sg
--protocol tcp --port 22 --cidr 0.0.0.0/0
```

Here's what each part of the command does:

- **authorize-security-group-ingress**: This is the command to add a rule that allows inbound traffic to a security group.
- **--group-name <student number>-sg**: We should replace `<student number>` with our actual identifier. This specifies the name of the security group to which you want to add the rule.
- **--protocol tcp**: Specifies the protocol for which we're allowing traffic. In this case, TCP is used, which is the protocol commonly used for many types of network communication.
- **--port 22**: Specify the port number that you want to allow traffic on. Port 22 is typically used for SSH (Secure Shell) connections, which are used for remote access to instances.
- **--cidr 0.0.0.0/0**: CIDR (Classless Inter-Domain Routing) notation specifies the range of IP addresses that are allowed to access the port. In this case, `0.0.0.0/0` means that traffic from any IP address is allowed.

The command authorizes inbound TCP traffic on port 22 (SSH) from any source IP address to the specified security group. This means that anyone can potentially access instances associated with this security group using SSH, which might be useful for development or administration purposes. However, allowing traffic from any IP address might also pose security risks in a production environment. It's generally recommended to restrict access to only trusted IP addresses or ranges.

Output: It gives the summary of all the things specified in the above command.

```
(aws-env)-(pritam@kali)-[~/Desktop/aws-labs]
$ aws ec2 authorize-security-group-ingress --group-name 23771397-sg --protocol tcp --port 22 --cidr 0.0.0.0/0
{
    "Return": true,
    "SecurityGroupRules": [
        {
            "SecurityGroupId": "sg-0867bc0fa7bf1a395",
            "GroupId": "sg-0867bc0fa7bf1a395",
            "GroupOwnerId": "489389878001",
            "IsEgress": false,
            "IpProtocol": "tcp",
            "FromPort": 22,
            "ToPort": 22,
            "CidrIpv4": "0.0.0.0/0"
        }
    ]
}
```

Step 3: Create a key pair that will allow you to ssh to the EC2 instance

```
aws ec2 create-key-pair --key-name <student number>-key --query
'KeyMaterial' --output text > <student number>-key.pem
```

- **create-key-pair:** This is the command to create a new key pair, which is used for secure authentication to EC2 instances.
- **--key-name <student number>-key:** We should replace `<student number>` with our actual identifier. This specifies the name of the key pair that we want to create. The `'-key` suffix is commonly added to indicate that it's a key pair.
- **--query 'KeyMaterial':** This specifies that we want to retrieve the private key material of the newly created key pair. The query expression `KeyMaterial` is used to extract the private key.
- **--output text:** This specifies the output format. In this case, we're requesting the output to be in plain text format.
- **'> <student number>-key.pem':** This part of the command directs the output (the private key material) to a file named `<student number>-key.pem`. The `>` symbol is used for output redirection.

The command creates a new key pair with the specified name, retrieves the private key material, and saves it to a file named `<student number>-key.pem`. This private key file is used for securely connecting to EC2 instances, particularly when using SSH to authenticate.

Output: The private key should not be shared

```
● [~(aws-env) (pritam㉿kali)] ~/Desktop/aws-labs]
● $ aws ec2 create-key-pair --key-name 23771397-key --query 'KeyMaterial' --output text > 23771397-key.pem
● [~(aws-env) (pritam㉿kali)] ~/Desktop/aws-labs]
● $ ls
23771397-key.pem tabulate-regions.py
● [~(aws-env) (pritam㉿kali)] ~/Desktop/aws-labs]
● $ cat 23771397-key.pem
-----BEGIN RSA PRIVATE KEY-----
[REDACTED]
-----END RSA PRIVATE KEY-----
[~(aws-env) (pritam㉿kali)] ~/Desktop/aws-labs]
```

To use this key on Linux, we should copy the file to a directory `~/.ssh` and change the permissions to:

```
chmod 400 <student number>-key.pem
```

The command **chmod 400 <student number>-key.pem** is used to change the permissions of the private key file `<student number>-key.pem` to make it more secure. The `chmod` command is used to modify the file permissions on Unix-like systems.

- `chmod`: Stands for "change mode," and it's a command to change the permissions of files or directories.
- `400`: This is a numeric value representing the file permissions. In this case, it sets read permission only for the owner of the file (4 corresponds to read permission in octal notation). The file's owner has full read access to the private key.
- `<student number>-key.pem`: This should be replaced with the actual filename.

Step 4: Create the instance and note the instance id

```
aws ec2 run-instances --image-id ami-d38a4ab1 --security-group-ids
<student number>-sg --count 1 --instance-type t2.micro --key-name <student
number>-key --query 'Instances[0].InstanceId'
```

- **aws ec2 run-instances**: This is the AWS CLI command to launch new EC2 instances.

- **--image-id ami-d38a4ab1**: Specifies the ID of the Amazon Machine Image (AMI) we want to use for the instance.
- **--security-group-ids <student number>-sg**: Specifies the security group ID or name (<student number>-sg) to associate with the instance. This controls inbound and outbound traffic.
- **--count 1**: Indicates the number of instances to launch. In this case, it's set to 1.
- **--instance-type t2.micro**: Specifies the instance type, in this case, t2.micro, which is a small and cost-effective instance type.
- **--key-name <student number>-key**: Specifies the name of the key pair to use for SSH access to the instance.
- **--query 'Instances[0].InstanceId'**: Uses JMESPath query language to extract the instance ID of the launched instance. This will be the output of the command, representing the unique identifier of the instance.

Running this command will launch a single EC2 instance with the specified configuration. The instance will be associated with the security group you've specified, allowing incoming traffic based on the security group rules.

Output: It gives the id of the ec2 instance running

```
• [pritam@pritam ~] $ aws ec2 run-instances --image-id ami-d38a4ab1 --security-group-ids 23771397-sg --count 1 --instance-type t2.micro --key-name 23771397-key --query 'Instances[0].InstanceId'
"i-0113960aealif7076"
```

Step 5: Get the public IP address

```
aws ec2 describe-instances --instance-ids i-<instance id from above>
--query 'Reservations[0].Instances[0].PublicIpAddress'
```

- **aws ec2 describe-instances**: This is the AWS CLI command to describe details of EC2 instances.
- **--instance-ids i-<instance id from above>**: Specifies the instance ID of the EC2 instance we want to describe. Replace <instance id from above> with the actual instance ID obtained from the previous command.
- **--query 'Reservations[0].Instances[0].PublicIpAddress'**: Uses JMESPath query language to extract the public IP address of the instance from the response.

Running this command will return the public IP address associated with the specified EC2 instance, allowing you to access the instance remotely using SSH or other protocols.

Output: "3.25.60.149". Gives the public IP address of the ec2 instance running.

```
(aws-env) (pritam@kali) [~/Desktop/aws-labs]
$ aws ec2 describe-instances --instance-ids i-007362e82a3284177 --query 'Reservations[0].Instances[0].PublicIpAddress'
"54.252.172.205"
```

Step 6: Connect to the instance

```
ssh -i <student number>-key.pem ubuntu@<IP Address>
```

- **ssh**: This is the command to initiate an SSH connection.
- **-i <student number>-key.pem**: Specifies the private key file (<student number>-key.pem) to use for authentication.
- **ubuntu**: This is the default username for Ubuntu instances on AWS EC2.
- **@<IP Address>**: Replace <IP Address> with the actual public IP address of the EC2 instance you want to connect to.

By executing this command, we'll be connecting to the EC2 instance using SSH, and the private key will be used for authentication. This allows us to access and manage the instance remotely through the terminal.

```

└─(pritam@pritam) [~/ssh]
$ ssh -i 23771397-key.pem ubuntu@3.25.60.149
The authenticity of host '3.25.60.149 (3.25.60.149)' can't be established.
ED25519 key fingerprint is SHA256:ndj24B3mvvur5F7IQK202lqILpcNCTNy31TMDvJ6LX4.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '3.25.60.149' (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.

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-172-31-12-202:~$ 

```

Create an EC2 instance with Python Boto script

```

import boto3

# Replace these with your actual credentials and region
aws_access_key = 'AKIAJD4PIXXXXXXXX'
aws_secret_key = 'XXXXXX+XXXXXXXXX/Zw0Rt3Bxzo3XXXXX'
region_name = 'ap-southeast-2'

# Create a Boto3 EC2 client
ec2_client = boto3.client(
    'ec2',

```

```

aws_access_key_id=aws_access_key,
aws_secret_access_key=aws_secret_key,
region_name=region_name
)

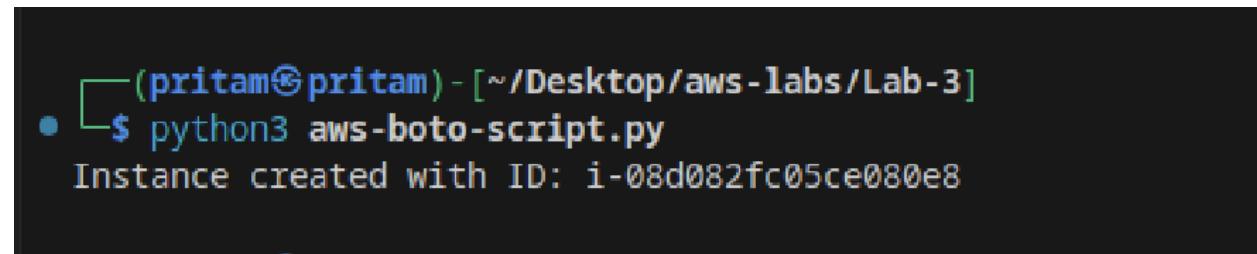
# Specify the parameters for the instance
instance_params = {
    'ImageId': 'ami-d38a4ab1', # Replace with the desired Amazon Machine Image
    (AMI) ID
    'InstanceType': 't2.micro', # Replace with the desired instance
    type
    'KeyName': '23771397-key', # Replace with the key pair name you want to
    use
    'MinCount': 1,
    'MaxCount': 1
}

# Create the instance
response = ec2_client.run_instances(**instance_params)

# Print the instance ID
instance_id = response['Instances'][0]['InstanceId']
print(f"Instance created with ID: {instance_id}")

```

Output: Running the above script creates an instance with the following id. It's easier to make changes with the **boto3** script than writing awscli commands.

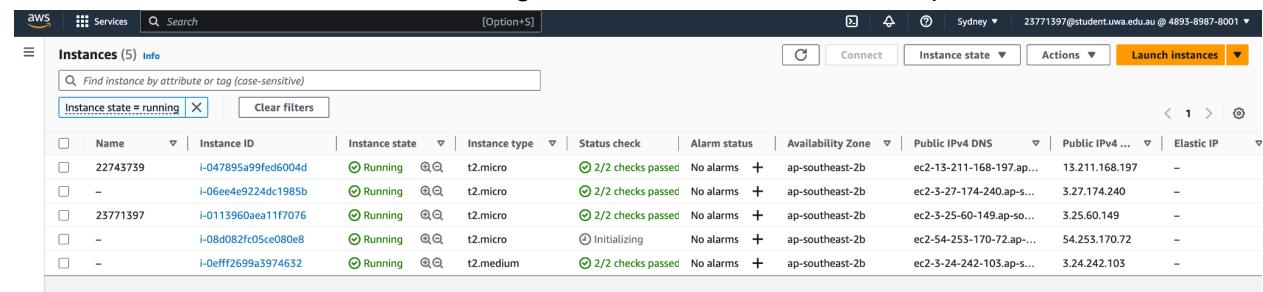


```

(pritam@pritam) - [~/Desktop/aws-labs/Lab-3]
$ python3 aws-boto-script.py
Instance created with ID: i-08d082fc05ce080e8

```

Creation of instance on the AWS Management Console with above script.



Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
22743739	i-047895a99fed6004d	Running	t2.micro	2/2 checks passed	No alarms	ap-southeast-2b	ec2-13-211-168-197.ap...	13.211.168.197	-
-	i-06ee4e9224dc1985b	Running	t2.micro	2/2 checks passed	No alarms	ap-southeast-2b	ec2-3-27-174-240.ap-s...	3.27.174.240	-
23771397	i-0113960aea11f7076	Running	t2.micro	2/2 checks passed	No alarms	ap-southeast-2b	ec2-3-25-60-149.ap-so...	3.25.60.149	-
-	i-08d082fc05ce080e8	Running	t2.micro	Initializing	No alarms	ap-southeast-2b	ec2-54-253-170-72.ap...	54.253.170.72	-
-	i-0efff2699a3974632	Running	t2.medium	2/2 checks passed	No alarms	ap-southeast-2b	ec2-3-24-242-103.ap-s...	3.24.242.103	-

AWS CLI Commands vs Boto3 Scripts (Python SDK):

AWS CLI Commands:

- **Syntax:** AWS CLI commands are entered directly into the command line interface.
- **Ease of Use:** Ideal for quick tasks and one-off operations that can be easily typed in.
- **Interactivity:** Provides a simple and interactive way to interact with AWS services.
- **Scripting:** Suitable for simple scripting and automation of straightforward tasks.

Boto3 Scripts (Python SDK):

- **Syntax:** Boto3 scripts are written in Python and use the Boto3 library.
- **Customization:** Offers extensive customization and control over AWS operations.
- **Complex Operations:** Well-suited for complex automation, handling dynamic conditions, and performing batch operations.
- **Modularity:** Code can be organized into functions and modules for reuse and maintainability.
- **Error Handling:** Provides robust error handling and retries in case of failures.

Termination of EC2 Instance

```
aws ec2 terminate-instances --instance-ids i-<your instance id>
```

- **aws ec2 terminate-instances:** This is the AWS CLI command to terminate EC2 instances.
- **--instance-ids i-<your instance id>:** Specifies the instance ID of the EC2 instance you want to terminate.

Executing this command will initiate the termination process for the specified EC2 instance. The instance will be shut down and eventually removed from the AWS account. We should be cautious when using this command, as terminated instances cannot be recovered, and their associated data may be lost.

```
(pritam@pritam) - [~/Desktop/aws-labs/Lab-3]
$ aws ec2 terminate-instances --instance-ids i-08d082fc05ce080e8
{
    "TerminatingInstances": [
        {
            "CurrentState": {
                "Code": 32,
                "Name": "shutting-down"
            },
            "InstanceId": "i-08d082fc05ce080e8",
            "PreviousState": {
                "Code": 16,
                "Name": "running"
            }
        }
    ]
}
```

The instance is removed from the list of running instances

Instances (3) Info										
C Connect Instance state ▾ Actions ▾ Launch instances ▾										
<input type="text"/> Find instance by attribute or tag (case-sensitive)										
Instance state = running X Clear filters										
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4	Public IPv6	Public IPv6
22743739	i-047895a99fed6004d	Running	t2.micro	2/2 checks passed	No alarms	+ ap-southeast-2b	ec2-13-211-168-197.ap...	13.211.168...		
-	i-06ee4e9224dc1985b	Running	t2.micro	2/2 checks passed	No alarms	+ ap-southeast-2b	ec2-3-27-174-240.ap-s...	3.27.174.2		
-	i-0efff2699a3974632	Running	t2.medium	2/2 checks passed	No alarms	+ ap-southeast-2b	ec2-3-24-242-103.ap-s...	3.24.242.1		

Lab 3

An Amazon S3 (Simple Storage Service) bucket is a scalable and secure cloud storage container provided by Amazon Web Services (AWS). It is designed to store and manage vast amounts of data, including files, images, videos, backups, and more. S3 buckets act as global namespaces, and each bucket has a unique name across the entire AWS platform. Users can configure access controls and permissions to manage who can upload, download, or modify the stored data. S3 buckets offer features like versioning, encryption, lifecycle policies, and integration with other AWS services, making them a versatile and reliable solution for storing, archiving, and distributing data in the cloud.

Step 1: Preparation and Save files to the S3 bucket.

First, we will create a local folder structure as follows:

```
rootdir
└── rootfile.txt
└── subdir
    └── subfile.txt
```

Step 2: Save to S3 bucket

The following script scans a directory and uploads all of the files found in the directory to an S3 bucket, preserving the path information

```
import os
import boto3
import base64

# Constants
ROOT_DIR = '.' # local root directory

bucket_name = '23771397-cloudstorage' # S3 bucket name

s3 = boto3.client("s3")
bucket_config = {'LocationConstraint': 'ap-southeast-2'}

# Create the S3 bucket if it doesn't exist
try:
```

```

s3.create_bucket(Bucket=bucket_name,
CreateBucketConfiguration=bucket_config)
except Exception as error:
    print("Bucket creation error:", error)

def upload_file(local_path, file_name):
    s3_path = f"{local_path}/{file_name}"
    local_file_path = os.path.join(local_path, file_name)
    print("Uploading", local_file_path, "to", s3_path)
    s3.upload_file(local_file_path, bucket_name, s3_path)

# 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(dir_name[2:], fname) # Remove the first two
            characters from dir_name

print("done")

```

The provided Python script uses the Boto3 library to interact with Amazon S3. It creates an S3 bucket named '**23771397-cloudstorage**' in the **ap-southeast-2** region if it doesn't exist. The script then **recursively** traverses a local directory, uploading all files to the S3 bucket. It strips the first two characters from the local directory name before creating an S3 path to maintain the directory structure. Thus, the script automates the process of uploading local files to an S3 bucket while preserving the folder structure.

Output: In the AWS Management Console, we can find the same folder structure being preserved as the local folder structure. Refer to the following screenshots.

Amazon S3 > Buckets > 23771397-cloudstorage

23771397-cloudstorage [Info](#)

Objects Properties Permissions Metrics Management Access Points

Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	rootdir/	Folder	-	-	-

Amazon S3 > Buckets > 23771397-cloudstorage > rootdir/

rootdir/

[Copy S3 URI](#)

Objects Properties

Objects (2)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	rootfile.txt	txt	August 31, 2023, 01:26:19 (UTC+08:00)	22.0 B	Standard
<input type="checkbox"/>	subdir/	Folder	-	-	-

Amazon S3 > Buckets > 23771397-cloudstorage > rootdir/ > subdir/

subdir/

[Copy S3 URI](#)

Objects Properties

Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	subfile.txt	txt	August 31, 2023, 01:26:41 (UTC+08:00)	0 B	Standard

Step 3: Restore from S3

```

import os
import boto3

# Constants
ROOT_DIR = '.' # Local root directory where files will be downloaded
bucket_name = '23771397-cloudstorage' # S3 bucket name

# Initialize the S3 client
s3 = boto3.client("s3")

def download_file(s3_path, local_path):
    """ Download a file from S3 to the local path."""
    s3.download_file(bucket_name, s3_path, local_path)

try:
    # List objects in the S3 bucket
    response = s3.list_objects_v2(Bucket=bucket_name)
    objects = response.get('Contents', [])

    for obj in objects:
        s3_path = obj['Key']
        local_path = os.path.join(ROOT_DIR, s3_path)

        # Create directories if they don't exist
        local_dir = os.path.dirname(local_path)
        os.makedirs(local_dir, exist_ok=True)

        # Download and save the file from S3 to the local path
        print("Downloading", s3_path, "to", local_path)
        download_file(s3_path, local_path)

except Exception as error:
    print("Error:", error)

print("done")

```

Breakdown of the script:

- The script sets the `ROOT_DIR` variable to the local directory where files will be downloaded and define the `bucket_name` for the S3 bucket.
- The S3 client is initialized using the `boto3` library.
- The `download_file` function is defined to download a file from S3 to a local path.

- The script lists objects in the S3 bucket using the **list_objects_v2** method and iterates through each object.
- For each object, it constructs the **S3 path** and the **corresponding local path**.
- It creates directories on the local machine if they don't exist using **os.makedirs**.
- The script then downloads the file from S3 to the local path using the **download_file** function.
- If any exceptions occur during the process, they are caught and an error message is printed.

Output: After deleting the existing folder structure, the above script will download it again.

The screenshot shows the Visual Studio Code interface with the 'restorefromcloud.py' script open. The script is used to download files from an S3 bucket to a local directory. A red arrow points from the terminal output to the code, indicating the execution of the script. The terminal shows the command being run and the progress of the file download. A message in the terminal indicates that the file was restored after deletion.

```

File Edit Selection View Go Run Terminal Help
File Explorer  Editor  Python Debug Console  Status Bar
LAB-3
  rootdir
    subdir
      subfile.txt
  cloudstorage.py
  restorefromcloud.py
The file and folder were restored again after deletion
zsh: corrupt history file /home/pritam/.zsh_history
pritam@pritam:~/Desktop/aws-labs/Lab-3$ /usr/bin/env /bin/python /home/pritam/.vscode/extensions/ms-python.python-2023.14.0/pythonFiles /lib/python/debugger/adaptar/.../debugpy/launcher 549
61 -- /home/pritam/Desktop/aws-labs/Lab-3/cloudstorage.py
uploading rootdir/rootfile.txt to rootdir/rootfile.txt
uploading rootdir/subdir/subfile.txt to rootdir/subdir/subfile.txt
done
pritam@pritam:~/Desktop/aws-labs/Lab-3$ cd /home/pritam/Desktop/aws-labs/Lab-3 ; /usr/bin/env /bin/python /home/pritam/.vscode/extensions/ms-python.python-2023.14.0/pythonFiles /lib/python/debugger/adaptar/.../debugpy/launcher 49235 ...
/home/pritam/Desktop/aws-labs/Lab-3/restorefromcloud.py
Downloading rootdir/rootfile.txt to ./rootdir/rootfile.txt
Downloading rootdir/subdir/subfile.txt to ./rootdir/subdir/subfile.txt
done
pritam@pritam:~/Desktop/aws-labs/Lab-3$ cd /home/pritam/Desktop/aws-labs/Lab-3 ; /usr/bin/env /bin/python /home/pritam/.vscode/extensions/ms-python.python-2023.14.0/pythonFiles /lib/python/debugger/adaptar/.../debugpy/launcher 50439 ...
/home/pritam/Desktop/aws-labs/Lab-3/restorefromcloud.py
Downloading rootdir/rootfile.txt to ./rootdir/rootfile.txt
Downloading rootdir/subdir/subfile.txt to ./rootdir/subdir/subfile.txt
done
pritam@pritam:~/Desktop/aws-labs/Lab-3$ 

```

Step 4: Write information about files to DynamoDB

Amazon DynamoDB is a fully managed NoSQL database service provided by Amazon Web Services (AWS). It is designed to deliver low-latency and high-performance access to data at any scale. DynamoDB offers features like automatic scaling, data replication, and seamless handling of read and write traffic fluctuations. It's suitable for a wide range of applications, including web and mobile apps, gaming, IoT, and more, where flexible and highly available database solutions are required.

Step 4a: Installing JRE

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

The command `sudo apt-get install default-jre` is used to install the default Java Runtime Environment (JRE) on a Debian-based Linux system. The JRE is necessary to run Java applications without the need for compiling source code.

```
(pritam@pritam) - [~/Desktop/aws-labs/Lab-3/dynamodb]
● $ sudo apt-get install default-jre
[sudo] password for pritam:
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.
The following packages were automatically installed and are no longer required:
  gir1.2-gtksource-3.0 gir1.2-javascriptcoregtk-4.0 gir1.2-soup-2.4 gir1.2-webkit2-4.0
  gobject-introspection king-phisher libavfilter8 libavformat59 libblockdev-crypto2 libblockdev-fs2
  libblockdev-loop2 libblockdev-part-err2 libblockdev-part2 libblockdev-swap2 libblockdev-utils2
  libblockdev2 libcodec2-1.0 libgda32 libgeos3.11.1 liblc3-0 libmongocrypt0 libmuj2 libncurses5
  libplacebo208 libpostproc56 libsoup-gnome2.4-1 libspatialite7 libsuperlu5 libswscale6 libtinfo5
  libwebsockets17 libyara9 pwgen python3-advancedhttpserver python3-boltons python3-cairo-dev
  python3-cryptography37 python3-flask-security python3-geoip2 python3-geojson python3-graphene
  python3-graphene-sqlalchemy python3-graphql-core python3-graphql-relay python3-icalendar
  python3-jaraco.classes python3-maxminddb python3-promise python3-py python3-pytz-deprecation-shim
  python3-requests-file python3-rule-engine python3-rx python3-smoke-zephyr python3-texttable tftp
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 148 not upgraded.

(pritam@pritam) - [~/Desktop/aws-labs/Lab-3/dynamodb]
○ $
```

Step 4b:

The provided commands download the latest version of Amazon DynamoDB Local—a local development and testing version of DynamoDB—using `wget`, and then start the DynamoDB Local server locally with shared in-memory databases for testing and development purposes using the Java Runtime Environment (JRE).

```
wget
https://s3-ap-northeast-1.amazonaws.com/dynamodb-local-tokyo/dynamodb_local
_latest.tar.gz
java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar
-sharedDb
```

- **wget**
https://s3-ap-northeast-1.amazonaws.com/dynamodb-local-tokyo/dynamodb_local
_latest.tar.gz: This command uses the wget utility to download the latest version of the Amazon DynamoDB Local package as a compressed tarball (tar.gz) from the specified

URL. DynamoDB Local is a downloadable version of DynamoDB that runs locally for development and testing purposes.

- **java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar –sharedDb:**
This command starts the DynamoDB Local server using the Java Runtime Environment (JRE). The options and arguments used in this command are as follows:

- **-Djava.library.path=./DynamoDBLocal_lib:** Sets the library path for DynamoDB Local dependencies.
- **-jar DynamoDBLocal.jar:** Launches the DynamoDB Local JAR file.
- **–sharedDb:** Specifies that multiple clients (connections) share the same in-memory database instance. This can be useful for local testing and development.

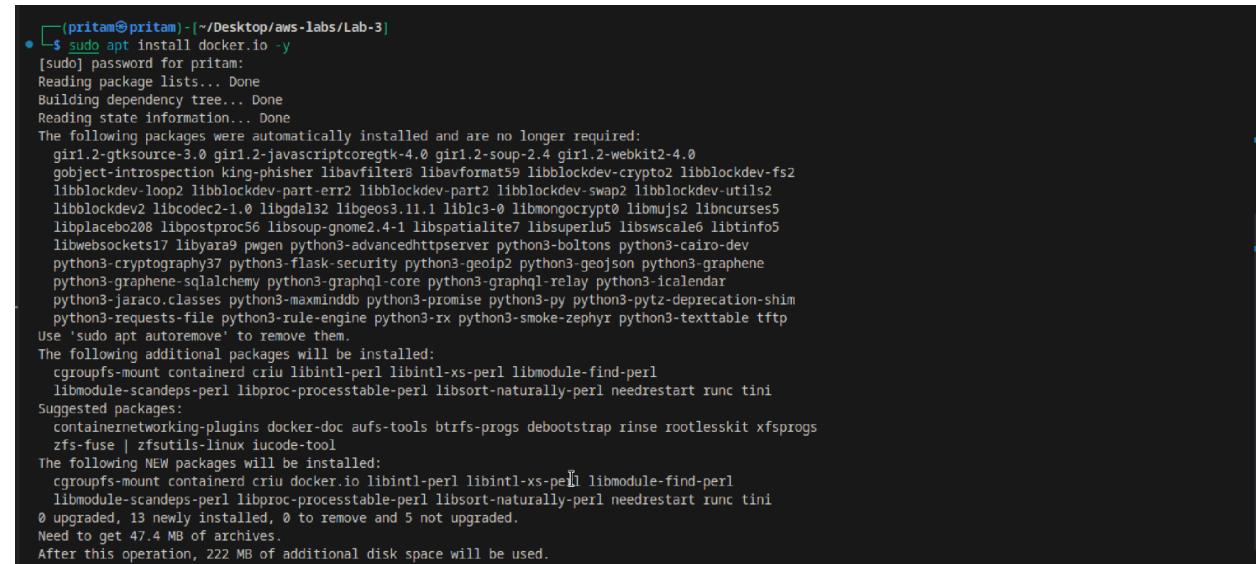
Step 5:

The above code didn't work in my case. So, I used docker instead.

Step 5a: Installing and Managing Docker

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

The command sudo apt install docker.io -y is used to install Docker on a Debian-based Linux system.



```
(pritam@pritam)-[~/Desktop/aws-labs/Lab-3]
$ sudo apt install docker.io -y
[sudo] password for pritam:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  gir1.2-gtksource-3.0 gir1.2-javascriptcoregtk-4.0 gir1.2-soup-2.4 gir1.2-webkit2-4.0
  gobject-introspection king-phisher libavfilter8 libavformat59 libblockdev-crypto2 libblockdev-fs2
  libblockdev-loop2 libblockdev-part-err2 libblockdev-part2 libblockdev-swap2 libblockdev-utils2
  libblockdev2 libcodec2-1.0 libgdal32 libgeo3_11.1 liblc3-0 libmongocrypt0 libmujs2 libncurses5
  libplacebo208 libpostproc56 libsoup-gnome2.4-1 libspatialite7 libsuperlu5 libwscale6 libtinfo5
  libwebsockets17 libyara9 pgen python3-advancedhttpserver python3-boltons python3-cairo-dev
  python3-cryptography37 python3-flask-security python3-geotp2 python3-geojson python3-graphene
  python3-graphene-sqlalchemy python3-graphql-core python3-graphql-relay python3-icalendar
  python3-jaraco.classes python3-maxminddb python3-promise python3-py python3-pytz-deprecation-shim
  python3-requests-file python3-rule-engine python3-rx python3-smoke-zephyr python3-texttable tftp
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  cgroupsfs-mount containerd criu libintl-perl libmodule-find-perl
  libmodule-scandeps-perl libproc-processstable-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:
  cgroupsfs-mount containerd criu docker.io libintl-perl libintl-xs-perl libmodule-find-perl
  libmodule-scandeps-perl libproc-processstable-perl libsort-naturally-perl needrestart runc tini
0 upgraded, 13 newly installed, 0 to remove and 5 not upgraded.
Need to get 47.4 MB of archives.
After this operation, 222 MB of additional disk space will be used.
```

- **sudo systemctl start docker**: This command starts the Docker service immediately. The start action initializes the Docker daemon and allows us to use Docker commands.
- **sudo systemctl enable docker**: This command enables the Docker service to start automatically on system boot. The enable action ensures that Docker starts automatically whenever the system is started or restarted.

```
(pritam@pritam) - [~/Desktop/aws-labs/Lab-3]
$ sudo systemctl start docker

(pritam@pritam) - [~/Desktop/aws-labs/Lab-3]
$ sudo systemctl enable docker
Synchronizing state of docker.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable docker

(pritam@pritam) - [~/Desktop/aws-labs/Lab-3]
$
```

Step 5b: Checking docker version

The command `docker --version` is used to display the version of the Docker software that is currently installed on your system. When you execute this command, it will provide the version information for Docker, which includes the version number and potentially additional details about the build and release. This helps us to verify the installed version and ensure compatibility with other tools and resources.

```
(pritam@pritam) - [~/Desktop/aws-labs/Lab-3]
$ docker --version
Docker version 20.10.25+dfsg1, build b82b9f3
```

Step 6: Running Amazon DynamoDB Local

```
docker run -p 8000:8000 amazon/dynamodb-local -jar DynamoDBLocal.jar
-inMemory -sharedDb
```

The `docker run` command starts a Docker container running Amazon DynamoDB Local, exposing port 8000 on the host machine for communication with the container's port 8000. The `--jar DynamoDBLocal.jar -inMemory -sharedDb` arguments configure DynamoDB Local to run in-memory with a shared database configuration, suitable for local testing and development.

```
File Actions Edit View Help
pritam@pritam: ~ x pritam@pritam: ~ x
951/44278951] Exploit-DB Google Hacking DB OffS # Insert
for obj
file
Error: Unable to access jarfile DynamoDBLocal.jar
└─(pritam@pritam)─[~]
$ docker run -p 8000:8000 amazon/dynamodb-local -jar DynamoDBLocal.jar -inMemory -sharedDb
Unable to find image 'amazon/dynamodb-local:latest' locally
latest: Pulling from amazon/dynamodb-local
0dfb0edac9a7: Pull complete
5ffbcc73bc8b: Pull complete
4f4fb700ef54: Pull complete
96b4d4d353f8: Pull complete
Digest: sha256:c8702bde709520b90930c20ee430d4123cd731da8e544cc4ccae0e2a78ee4fce
Status: Downloaded newer image for amazon/dynamodb-local:latest
Initializing DynamoDB Local with the following configuration:
Port: 8000
InMemory: true
DbPath: null
SharedDb: true
shouldDelayTransientStatuses: false
CorsParams: null
[...]
Make sure
name ''.
values.
```

Step 7: Create a table on local DynamoDB with the key userId

```
import boto3

# Initialize DynamoDB client
dynamodb = boto3.resource('dynamodb')

# Table name
table_name = '23771397-CloudFiles'

# Create the table if it doesn't exist
def create_table():
    # Define the table schema and attributes
    table = dynamodb.create_table(
        TableName=table_name,
        # Define the primary key attributes (userId as HASH, fileName as RANGE)
```

```

KeySchema=[  

    {  

        'AttributeName': 'userId',  

        'KeyType': 'HASH'  

    },  

    {  

        'AttributeName': 'fileName',  

        'KeyType': 'RANGE'  

    }  

],  
  

# Define the data types of the primary key attributes  

AttributeDefinitions=[  

    {  

        'AttributeName': 'userId',  

        'AttributeType': 'S' # String type  

    },  

    {  

        'AttributeName': 'fileName',  

        'AttributeType': 'S' # String type  

    }  

],  
  

# Set the read and write capacity units  

ProvisionedThroughput={  

    'ReadCapacityUnits': 5,  

    'WriteCapacityUnits': 5  

}  

)  
  

# Wait for the table to be created before proceeding  

table.meta.client.get_waiter('table_exists').wait(TableName=table_name)  
  

if __name__ == "__main__":  

    # Call the create_table function when the script is directly executed  

    create_table()  

    print("Table creation done")

```

The script initializes a Boto3 DynamoDB resource client to interact with DynamoDB.

- A constant **table_name** is set to '23771397-CloudFiles'.
- The **create_table** function is defined to create a DynamoDB table with the specified schema and attributes.
- Inside **create_table**, **dynamodb.create_table** is called with:
 - TableName set to **table_name**.

- KeySchema defining the primary key attributes: 'userId' as HASH and 'fileName' as RANGE.
- AttributeDefinitions specifying the data types of the primary key attributes.
- ProvisionedThroughput setting read and write capacity units.
- The script waits for the table to be created using `get_waiter('table_exists').wait()`.
- If the script is directly executed (using `__name__`), the `create_table` function is called, and a message is printed indicating that table creation is done.

Output: The following screenshot shows that the table has been created with zero items initially.

The screenshot shows a terminal window with two main sections of output:

Top Section:

```
zsh: corrupt history file /home/pritam/.zsh_history
(pritam@pritam) - [~/Desktop/aws-labs/Lab-3-dynamodb]
$ docker run -p 8000:8000 amazon/dynamodb-local -jar DynamoDBLocal.jar -inMemory -s
sharedDb
Initializing DynamoDB Local with the following configuration:
Port: 8000
InMemory: true
DbPath: null
SharedDb: true
shouldDelayTransientStatuses: false
CorsParams: null
Local Dynamodb
```

Bottom Section:

```
(pritam@pritam) - [~/Desktop/aws-labs/Lab-3-dynamodb]
$ aws dynamodb scan --table-name CloudFiles
{
    "Items": [],
    "Count": 0,
    "ScannedCount": 0,          No item at start
    "ConsumedCapacity": null
}

(pritam@pritam) - [~/Desktop/aws-labs/Lab-3-dynamodb]
$
```

The output is highlighted with red boxes around the Docker command and the AWS command. The AWS command output includes a note "No item at start".

Step 7b: Created table on AWS Management Console as well

The screenshot shows the AWS DynamoDB 'Explore Items' interface for the table '23771397-CloudFiles'. The left sidebar lists 'Tables (32)'. The main area has a heading '23771397-CloudFiles' with tabs for 'Scan or query items' (selected), 'Autopreview', and 'View table details'. Under 'Scan or query items', there are sections for 'Scan' (selected), 'Query', 'Select a table or index' (set to 'Table - 23771397-CloudFiles'), 'Select attribute projection' (set to 'All attributes'), and 'Filters'. Below these are 'Run' and 'Reset' buttons. A green message bar at the bottom says 'Completed. Read capacity units consumed: 0.5'. The main content area shows a table with a single row: 'Items returned (0)'. To the right of the table are 'Actions' and 'Create item' buttons, along with navigation controls. The status bar at the bottom says 'No items to display.' and 'Create item'.

Step 8: Store information S3 Object information in Dynamodb table

```
import os
import boto3
from datetime import datetime

# Initialize AWS clients
s3 = boto3.client('s3', region_name='ap-southeast-2') # S3 client
dynamodb = boto3.resource('dynamodb') # DynamoDB resource client
table = dynamodb.Table('23771397-CloudFiles') # DynamoDB table
bucket_name = '23771397-cloudstorage' # S3 bucket name
iam = boto3.client('iam') # IAM client

# Get information about the IAM user
response = iam.get_user()
user_id = response['User']['UserId'] # Extract user ID

# List objects in the S3 bucket
objects = s3.list_objects(Bucket=bucket_name)

# Loop through S3 objects
for obj in objects.get('Contents', []):
    file_key = obj['Key']
```

```

file_name = os.path.basename(file_key) # Extract file name from object key
file_path = f"s3://{bucket_name}/{file_key}" # Formulate S3 object path
last_updated = obj['LastModified'].strftime('%Y-%m-%d %H:%M:%S') # Convert
last modified timestamp to formatted string
owner = obj['Owner']['DisplayName'] # Extract owner's display name

# Fetch object ACLs
object_acl = s3.get_object_acl(Bucket=bucket_name, Key=file_key)
permissions = []
for grant in object_acl['Grants']:
    permission = grant['Permission']
    grantee = grant['Grantee']['DisplayName']
    permissions.append(f"{permission}: {grantee}") # Store permission and
grantee information

# Insert data into DynamoDB
item = {
    'userId': user_id,
    'fileName': file_name,
    'path': file_path,
    'lastUpdated': last_updated,
    'owner': owner,
    'permissions': permissions # Store permissions information
}
table.put_item(Item=item) # Put item into DynamoDB table
print(f"Inserted item for '{file_name}' into DynamoDB.")

print("Data insertion complete.")

```

The script interacts with AWS services (S3, DynamoDB, and IAM) to process and insert data about S3 objects into a DynamoDB table.

- It initializes the AWS clients for S3, DynamoDB, and IAM, sets the table name and bucket name, and gets the IAM user's ID.
- The script lists objects in the S3 bucket and loops through each object.
- For each object, it extracts the file name, creates an S3 object path, formats the last modified timestamp, and fetches the object's ACLs.
- Inside the ACL loop, it extracts permission and grantee information and stores them in a permissions list.
- It assembles the data into a dictionary (item) and uses `table.put_item()` to insert the item into the DynamoDB table.
- Information about the inserted item is printed to the console.

- Once all objects are processed, a message is printed indicating the completion of data insertion.

DynamoDB > Explore items > 23771397-CloudFiles

23771397-CloudFiles

Scan or query items

Select a table or index: Table - 23771397-CloudFiles

Select attribute projection: All attributes

Filters

Run Reset

Completed. Read capacity units consumed: 0.5

Items returned (2)

	userid (String)	fileName (String)	lastUpdated	owner	path	permissions
<input type="checkbox"/>	AIDAXD4PISLY4SPF3Y24X	rootfile.txt	2023-08-30 1...	zhi.zhang	s3://23771...	[{"S": "FULL_CONTROL: zhi.zhang"}]
<input type="checkbox"/>	AIDAXD4PISLY4SPF3Y24X	subfile.txt	2023-08-30 1...	zhi.zhang	s3://23771...	[{"S": "FULL_CONTROL: zhi.zhang"}]

Note: Make sure to add sort key as well as partition key .Otherwise, the userid would be same and was being overwritten by new item everytime

Lab 4

A bucket policy in AWS is a JSON-based access control configuration that allows you to define fine-grained permissions for an Amazon S3 bucket. It specifies who can access the bucket and what actions they can perform on the objects within it. Bucket policies can grant or deny permissions to AWS accounts, IAM users, or even make objects publicly accessible. These policies are a powerful way to control access to your S3 resources and are commonly used for tasks like setting up cross-account access, enabling public website hosting, or enforcing data retention policies.

Step 1: Apply policy to restrict permissions on bucket

```
import boto3
import json

# Replace these values with your actual values
bucket_name = "23771397-cloudstorage"
username = "23771397@student.uwa.edu.au"

# Policy dictionary
policy = {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAllS3ActionsInUserFolderForUserOnly",
            "Effect": "DENY",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": f"arn:aws:s3:::{bucket_name}/*",
            "Condition": {
                "StringNotLike": {
                    "aws:username": username
                }
            }
        }
    ]
}

# Convert policy dictionary to JSON
policy_json = json.dumps(policy)

# Initialize the S3 client
s3 = boto3.client('s3')

# Apply the policy to the bucket
```

```

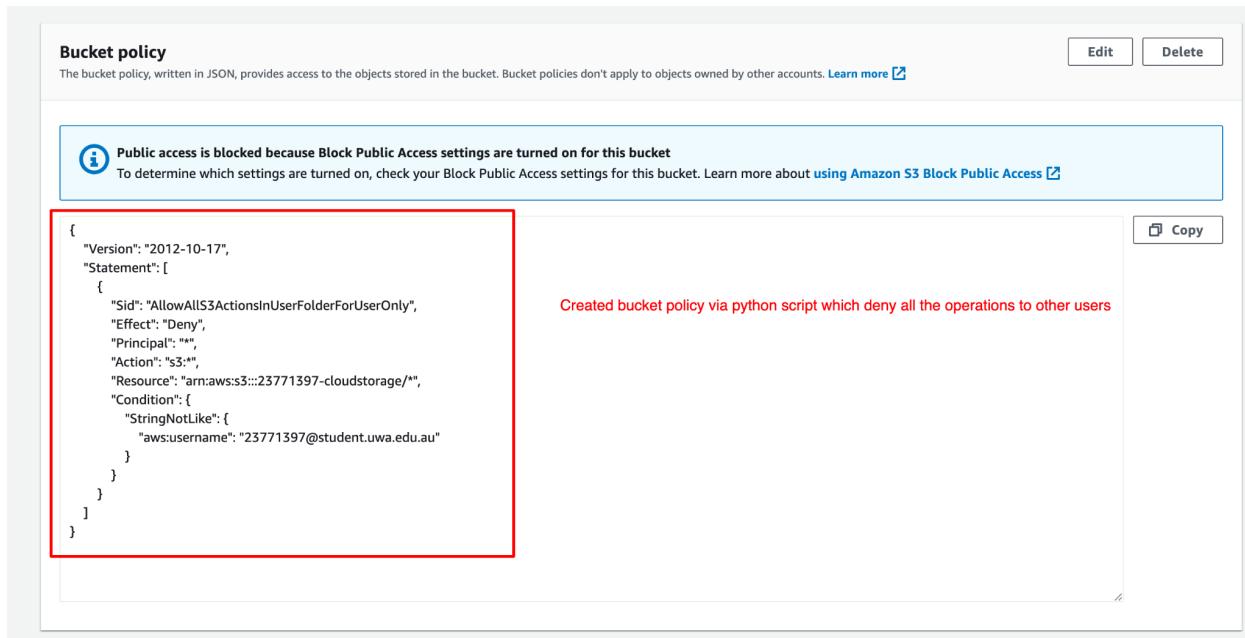
s3.put_bucket_policy(
    Bucket=bucket_name,
    Policy=policy_json
)
print("Policy applied successfully.")

```

The script uses Boto3 to apply a bucket policy to an S3 bucket.

- It defines a policy dictionary with a policy statement that **denies** all S3 actions for users other than **the specified username**.
- The S3 client is initialized, and **s3.put_bucket_policy()** is used to apply the policy to the specified bucket.

In S3 bucket, the above code will add the following policy as attached on screenshot.



Bucket policy
The bucket policy, written in JSON, provides access to the objects stored in the bucket. Bucket policies don't apply to objects owned by other accounts. [Learn more](#)

Public access is blocked because Block Public Access settings are turned on for this bucket
To determine which settings are turned on, check your Block Public Access settings for this bucket. Learn more about [using Amazon S3 Block Public Access](#)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAllS3ActionsInUserFolderForUserOnly",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::23771397-cloudstorage/*",
      "Condition": {
        "StringNotLike": {
          "aws:username": "23771397@student.uwa.edu.au"
        }
      }
    }
  ]
}
```

Created bucket policy via python script which deny all the operations to other users [Copy](#)

When other IAM user try to access the bucket, they will get following error message.



23771397-cloudstorage.s3.ap-southeast-2.amazonaws.com/rootdir/rootfile.txt?response-content-disposition=attachment&X-Amz-Security-Token=lQoJb3Jp...

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```

<Error>
  <Code>AccessDenied</Code>
  <Message>Access Denied</Message>
  <RequestId>JYAY4B6AV4SW0988</RequestId>
  <HostId>reTCdaus8D5AtUsa43AT+i2jh+os5f6K/XK10QJE6dPLzzlnndwSilXoJ7r1Yg9f56Hx0swBZcw=</HostId>
</Error>
```

Step 2: AES Encryption using KMS

AWS Key Management Service (KMS) is a managed service that enables the creation and control of cryptographic keys for securing sensitive data. It provides centralized management of encryption keys used to encrypt and decrypt data within AWS services and applications. KMS offers features like key rotation, audit logging, and integration with various AWS services, making it easier to enforce encryption best practices. It helps ensure the security and privacy of data by allowing users to manage encryption keys without the complexity of manual key management.

Step 2a: Creating KMS Policy

```
import boto3
import json

# Replace these with your actual values
alias_name = "23771397-t"
username = "23771397@student.uwa.edu.au"

kms_policy = {
    "Version": "2012-10-17",
    "Id": "key-consolepolicy-3",
    "Statement": [
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::489389878001:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allow access for Key Administrators",
            "Effect": "Allow",
            "Principal": {
                "AWS": f"arn:aws:iam::489389878001:user/{username}"
            },
            "Action": [
                "kms>Create*",
                "kms>Describe*",
                "kms>Enable*",
                "kms>List*",
                "kms>Update*"
            ]
        }
    ]
}
```

```
        "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": f"arn:aws:iam::489389878001:user/{username}"
    },
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
    ],
    "Resource": "*"
},
{
    "Sid": "Allow attachment of persistent resources",
    "Effect": "Allow",
    "Principal": {
        "AWS": f"arn:aws:iam::489389878001:user/{username}"
    },
    "Action": [
        "kms>CreateGrant",
        "kms>ListGrants",
        "kms:RevokeGrant"
    ],
    "Resource": "*",
    "Condition": {
        "Bool": {
            "kms:GrantIsForAWSResource": "true"
        }
    }
}
```

```
        }
    ]
}

print(kms_policy)

# Create KMS client
kms_client = boto3.client('kms')

# Create KMS key
key_response = kms_client.create_key()

key_id = key_response['KeyMetadata']['KeyId']

# Create alias for the key
kms_client.create_alias(
    AliasName=f'alias/{alias_name}',
    TargetKeyId=key_id
)

# Attach policy to the key
# policy = json.loads(policy_json)
kms_client.put_key_policy(
    KeyId=key_id,
    PolicyName='default',
    Policy=json.dumps(kms_policy)
)

print(f"KMS Key created with ID: {key_id}")
print(f"Alias '{alias_name}' created for the key")
print("Policy attached to the key")
```

The script uses Boto3 to automate the creation and configuration of an AWS KMS key.

- The **kms_policy** dictionary defines a comprehensive policy specifying permissions and actions for the KMS key.
- It initializes a KMS client using **boto3.client('kms')**.
- A KMS key is created using **kms_client.create_key()**, and the **key_id** is extracted from the response.

- An alias for the key is created using `kms_client.create_alias()`, associating the alias with the generated key.
- The script attaches the defined kms_policy to the key using `kms_client.put_key_policy()`.

Step 3: Encrypt and decrypt the files using the KMS Client

```

import boto3

# Create an S3 client
s3_client = boto3.client('s3')

# Define your S3 bucket name
bucket_name = '23771397-cloudstorage'

def encrypt_file(file_path, kms_alias):
    with open(file_path, 'rb') as file:
        file_contents = file.read()

    kms_client = boto3.client('kms')
    response = kms_client.describe_key(KeyId=kms_alias)
    kms_key_id = response['KeyMetadata']['KeyId']

    encrypted_data = kms_client.encrypt(
        KeyId=kms_key_id,
        Plaintext=file_contents
    )

    return encrypted_data['CiphertextBlob']

def decrypt_data(encrypted_data, kms_alias):
    kms_client = boto3.client('kms')
    response = kms_client.describe_key(KeyId=kms_alias)
    kms_key_id = response['KeyMetadata']['KeyId']

    decrypted_data = kms_client.decrypt(
        KeyId=kms_key_id,
        CiphertextBlob=encrypted_data
    )

    return decrypted_data['Plaintext']

# Upload a file to S3

```

```

def upload_file_to_s3(file_path, s3_key):
    with open(file_path, 'rb') as file:
        s3_client.upload_fileobj(file, bucket_name, s3_key)

# Download a file from S3
def download_file_from_s3(s3_key, local_path):
    with open(local_path, 'wb') as file:
        s3_client.download_fileobj(bucket_name, s3_key, file)

# Usage
file_path = 'test.txt'
encrypted_file_path = 'encrypted_test.txt'
decrypted_file_path = 'decrypted_test.txt'
kms_alias = 'alias/23771397-t'

# Encrypt and upload the file
encrypted_data = encrypt_file(file_path, kms_alias)
s3_client.put_object(
    Bucket=bucket_name,
    Key=encrypted_file_path,
    Body=encrypted_data
)
print(f"Encrypted file uploaded to S3:
{s3://{{bucket_name}}/{{encrypted_file_path}}")

# Download and decrypt the file
download_file_from_s3(encrypted_file_path, encrypted_file_path)
decrypted_data = decrypt_data(encrypted_data, kms_alias)

with open(decrypted_file_path, 'wb') as file:
    file.write(decrypted_data)

print(f"Decrypted file saved locally: {{decrypted_file_path}}")

```

The script initializes an S3 client and defines the bucket_name where files will be stored.

- Two functions, **encrypt_file** and **decrypt_data**, are defined to perform encryption and decryption using KMS.
- Additional functions, **upload_file_to_s3** and **download_file_from_s3**, are created to handle file uploads and downloads using the S3 client.
- The **encrypt_file** function is called to encrypt the contents of a file.
- The encrypted data is uploaded to the S3 bucket using **put_object**.
- The **download_file_from_s3** function is called to download the encrypted file.
- The **decrypt_data** function is used to decrypt the downloaded data.
- The decrypted data is written to a local file.

This script demonstrates how to securely upload, store, and manage encrypted files in an S3 bucket using KMS for encryption and decryption operations.

Output: Encrypted and decrypted file using KMS Client.

Encrypted file that has been uploaded to S3 bucket.

Amazon S3 > Buckets > 23771397-cloudstorage

23771397-cloudstorage Info

Objects | Properties | Permissions | Metrics | Management | Access Points

Objects (2)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	 encrypted_test.txt	txt	August 31, 2023, 14:35:08 (UTC+08:00)	163.0 B	Standard
<input type="checkbox"/>	 rootdir/	Folder	-	-	-

The contents of encrypted file.

```
xx"Ã†{HÜI03]<ZÌ}÷„ÿs  
°fv"◊ô$ÀYçùâéÓx°·æi0g  *ÜHÜ~  
†ZØXØS  *ÜHÜ~  
0  `ÜHe.Ø  
Öw◊]YK8øY4øÄ&@äʃ#æ‡·Ω©ð†c©\ÃfidN. `opÎ/oEÖFWYüÎ≥YYs|
```

Step 4: AES Encryption using pycryptodome

```
import os, struct  
from Crypto.Cipher import AES  
from Crypto import Random
```

```
import boto3
import hashlib

# Initialize the S3 client
s3_client = boto3.client('s3')

# Define your S3 bucket name
bucket_name = '23771397-cloudstorage'

# Define the chunk size for reading and writing files
CHUNK_SIZE = 64 * 1024

# Function to encrypt a file before uploading to S3 using AES
def encrypt_file(password, local_path, encrypted_path):
    # Generate a key from the password using SHA-256
    key = hashlib.sha256(password.encode("utf-8")).digest()

    # Generate a random initialization vector (IV)
    iv = Random.new().read(AES.block_size)

    with open(local_path, 'rb') as infile:
        with open(encrypted_path, 'wb') as outfile:
            # Write the original file size and IV to the encrypted file
            outfile.write(struct.pack('<Q', os.path.getsize(local_path)))
            outfile.write(iv)

            # Create an AES encryptor
            encryptor = AES.new(key, AES.MODE_CBC, iv)

            while True:
                chunk = infile.read(CHUNK_SIZE)
                if len(chunk) == 0:
                    break
                elif len(chunk) % 16 != 0:
                    # Pad the chunk if its size is not a multiple of 16 bytes
                    chunk += b' ' * (16 - len(chunk) % 16)

                outfile.write(encryptor.encrypt(chunk))

# Function to decrypt a file after downloading from S3 using AES
def decrypt_file(password, encrypted_path, decrypted_path):
    # Generate a key from the password using SHA-256
    key = hashlib.sha256(password.encode("utf-8")).digest()
```

```

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

    # Create an AES decryptor
    decryptor = AES.new(key, AES.MODE_CBC, iv)

    with open(decrypted_path, 'wb') as outfile:
        while True:
            chunk = infile.read(CHUNK_SIZE)
            if len(chunk) == 0:
                break
            outfile.write(decryptor.decrypt(chunk))

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

# Example usage
password = 'kitty and the kat'
local_file_path = './test.txt'
s3_file_path = 'rootdir/test.txt.encrypted'

# Encrypt and upload the file to S3
encrypted_path = local_file_path + '.encrypted'
encrypt_file(password, local_file_path, encrypted_path)
s3_client.upload_file(encrypted_path, bucket_name, s3_file_path)
os.remove(encrypted_path) # Remove the encrypted temporary file

# Download and decrypt the file from S3
downloaded_file_path = local_file_path + '.decrypted'
local_encrypted_path = local_file_path + '.encrypted'
s3_client.download_file(bucket_name, s3_file_path, local_encrypted_path)
decrypt_file(password, local_encrypted_path, downloaded_file_path)
os.remove(local_encrypted_path) # Remove the downloaded encrypted file

print("File Uploaded")

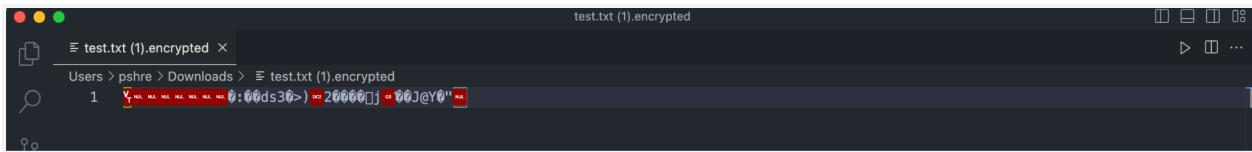
```

Above is a Python script that uses the Crypto library for AES encryption and decryption, along with Boto3 for interacting with AWS S3. Here's an explanation of what this script does:

- It initializes the S3 client and defines the `bucket_name` where files will be stored in Amazon S3.
- The script defines a `CHUNK_SIZE` constant for reading and writing files in chunks.

- **encrypt_file**: This function takes a password, the local path of a file, and the desired path for the encrypted file. It encrypts the file using AES encryption with CBC mode and writes the encrypted content to the specified file path. It also prepends the original file size and an initialization vector (IV) to the encrypted file.
 - **decrypt_file**: This function takes the same password, the path to the encrypted file, and the desired path for the decrypted file. It reads the encrypted file, extracts the original file size and IV, decrypts the content, and writes it to the specified decrypted file path.
 - The example usage section demonstrates how to use these functions:
 - A password is provided for encryption and decryption.
 - A **local_file_path** points to the file we want to encrypt and upload to S3.
 - An **s3_file_path** specifies the S3 destination for the encrypted file.
 - The script encrypts the file using the `encrypt_file` function, uploads it to S3 using `s3_client.upload_file`, and then removes the temporary encrypted file.
 - It also demonstrates downloading the encrypted file from S3, decrypting it using the `decrypt_file` function, and removing the downloaded encrypted file.

Encrypted File downloaded from AWS Management Console.



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

The performance difference between using AWS Key Management Service (KMS) and a custom encryption solution can vary depending on several factors, including the complexity of the encryption algorithm, the size of the data being encrypted, and the specific use case. Here are some considerations:

KMS Performance:

- **Managed Service:** KMS is a fully managed service provided by AWS. It is designed for high availability, scalability, and low latency. AWS manages the infrastructure and operations, ensuring consistent and reliable performance.

- **Hardware Acceleration:** AWS KMS uses hardware security modules (HSMs) for key storage and cryptographic operations. These HSMs are optimized for encryption and decryption tasks, which can result in high performance.
- **Integration:** KMS seamlessly integrates with various AWS services, making it easy to encrypt and decrypt data within the AWS ecosystem.

Custom Encryption Solution Performance:

- **Algorithm Choice:** The performance of a custom encryption solution depends on the encryption algorithm used. Some algorithms are computationally more intensive than others. AES (Advanced Encryption Standard) is a commonly used and efficient encryption algorithm.
- **Implementation:** The quality of the custom encryption implementation can significantly impact performance. Well-optimized code and parallelization techniques can improve speed.
- **Key Management:** Managing encryption keys securely in a custom solution can add complexity. If not handled properly, it can impact both security and performance.

Performance Comparison:

- For small to moderately-sized files, the performance difference between KMS and a well-optimized custom solution may not be significant.
- However, as the size of data and the number of encryption/decryption operations increase, KMS can often provide more consistent and predictable performance due to its optimized hardware and infrastructure.
- Custom solutions may offer more flexibility but require careful optimization to match KMS performance, especially at scale.