Configuration Manual

MSc Research Project
Cyber Security

WAHAJ RASHID
Student ID: X23197960

School of Computing
National College of Ireland

Supervisor: ROHIT VERMA

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Wahaj Rashid

Student ID: X23197960

Programme: MSc CYB JAN24 **Year:** 2024-2025

Module: MSc Research Practicum

Lecturer: Rohit Verma

Submission Due Date: 12-12-2024 @ 02:00pm

Project Title: ANALYSIS OF AUTOMATED ZERO TRUST AWS HOME NETWORK FOR CONFIDENTIALITY AND AUTHENTICATION ISSUES

Word Count: 1245 **Page Count:** 12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:

Date: 12-12-2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Wahaj Rashid

Student ID: x21397960

1. INTRODUCTION

This configuration manual provides a step-by-step procedure recreate a Zero Trust AWS environment to address confidentiality and authentication issues in a home network.

2. SYSTEM CONFIGURATION

2.1. SYSTEM HARDWARE

MODEL NAME	HP EliteBook 735 G6
SYSTEM TYPE	WINDOWS 11 64-BIT OPERATING SYSTEM, X64-BASED PROCESSOR
PROCESSOR	AMD Ryzen 5 Pro
RAM	16GB
HARD DISK	512GB

2.2 SOFTWARE REQUIREMENT

For Admin - Local System

Software	Version	Type
Visual Studio Code (VSCODE)	v1.85.1	IDE - Code Editor
Visual Studio Code	v1.74	IDE - Code Editor
Terraform CLI	v1.9.7	Infrastructure as Code Tool
Amazon Web Services (AWS CLI)	v2.18.5	Command Line Interface for AWS
Amazon Web Service Management Console	N/A	Web-Based Cloud Management Console
Python	v3.12.3	Programming Language
Pip	v24.1.2	Python Package Installer

Twingate Console for Admin	N/A	Zero Trust Network Management Console

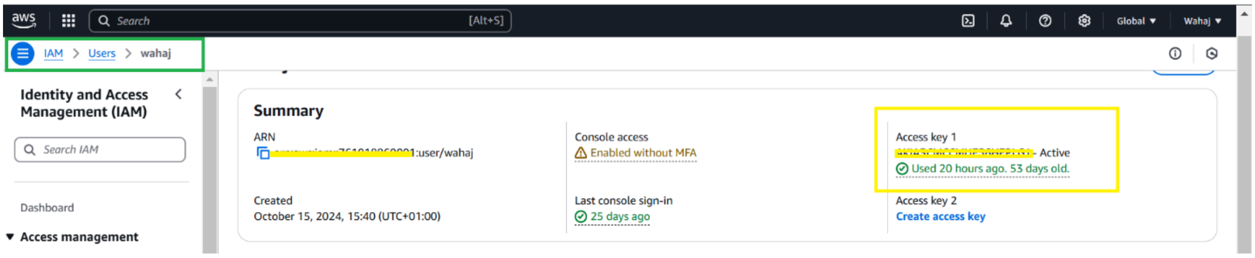
For Users - Mobile devices

Software	**Version**	**Type**
Twingate Client	v2024.311	Zero Trust Network Client
Termius	v5.9.5	SSH and SFTP Client

3. INSTALLATION AND SETUP

For Local System - Admin

- Create AWS Account (We are using Free Tier)
- Any IDE of your choice like VS Code
- Download and Install AWS CLI: v2.18.5. (Installing or updating to the latest version of the AWS cli - AWS command line interface, n.d.)
  - Verify after installation by running **AWS --version**
  - To configure Run **AWS configure**
  - Enter AWS Access Key ID
  - Enter AWS Secret Key ID
    - Retrieve access and secret key ids from AWS IAM user
    - Go to AWS account > IAM > Users > Create User > Create Access Key as shown in Fig. 1



Summary

ARN
[redacted]:user/wahaj

Created
October 15, 2024, 15:40 (UTC+01:00)

Console access
⚠ Enabled without MFA

Last console sign-in
✓ 25 days ago

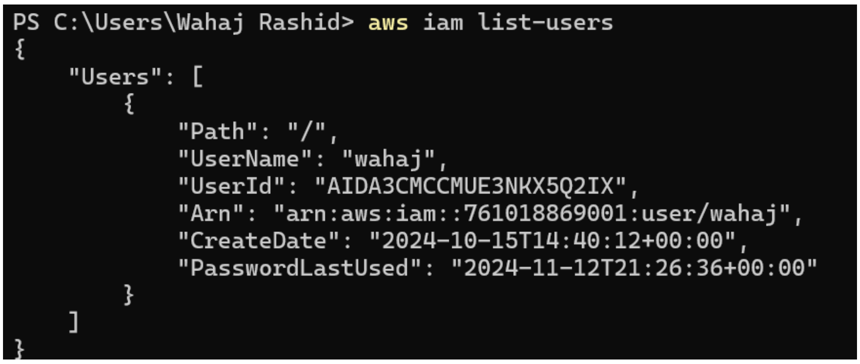
Access key 1
[redacted] - Active
✓ Used 20 hours ago. 53 days old.

Access key 2
Create access key

Fig 1. IAM user with access key

- Enter region **eu-west-1**
- Enter output format **json**

Run AWS IAM list-users to verify if its connected as shown in Fig. 2

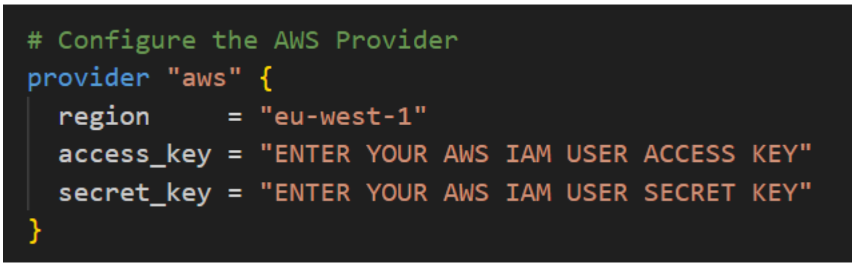


```
PS C:\Users\Wahaj Rashid> aws iam list-users
{
    "Users": [
        {
            "Path": "/",
            "UserName": "wahaj",
            "UserId": "AIDA3CMCCMUE3NKX5Q2IX",
            "Arn": "arn:aws:iam::761018869001:user/wahaj",
            "CreateDate": "2024-10-15T14:40:12+00:00",
            "PasswordLastUsed": "2024-11-12T21:26:36+00:00"
        }
    ]
}
```

Fig.2 Verify if AWS cli is connected to our AWS account

- Download and Install Terraform: v1.9.7 make sure you add its path to environmental variable in after installation(Install | terraform | hashicorp developer, n.d.)
- Verify if its install by running **Terraform --version**

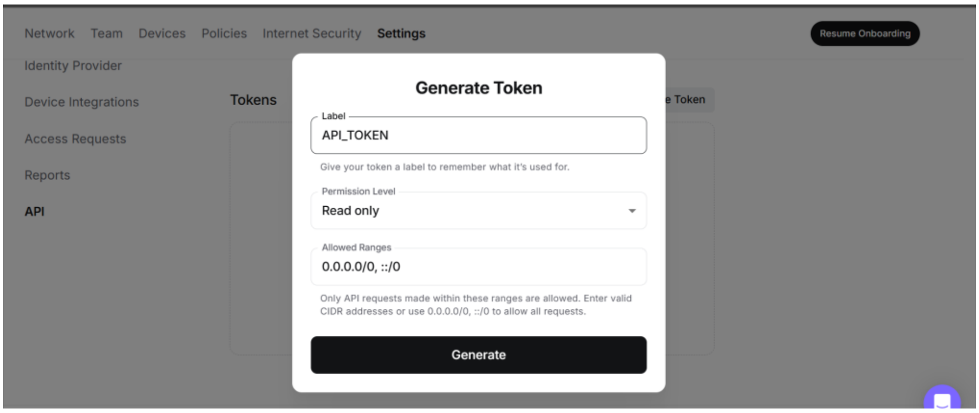
Enter code in main.tf file to create provider as shown In Fig. 3 use credentials from IAM users



```
# Configure the AWS Provider
provider "aws" {
  region     = "eu-west-1"
  access_key = "ENTER YOUR AWS IAM USER ACCESS KEY"
  secret_key = "ENTER YOUR AWS IAM USER SECRET KEY"
}
```

Fig.3 Configuration of AWS Provider

- Create Twingate network on web management console (Twingate, n.d.)
  Go to Admin Console > Settings > Generate API Token and paste it in code with network name to establish a connection between Twingate Admin/Client and configure it in code as shown in Fig 4, Fig 5



Generate Token

Label: API_TOKEN

Give your token a label to remember what it's used for.

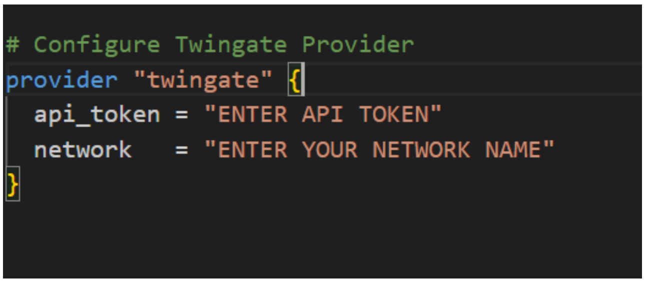
Permission Level: Read only

Allowed Ranges: 0.0.0.0/0, ::/0

Only API requests made within these ranges are allowed. Enter valid CIDR addresses or use 0.0.0.0/0, ::/0 to allow all requests.

Generate

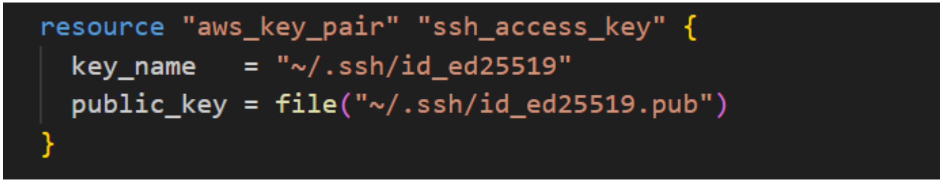
Fig. 4 Generate API Token for Provider



```
# Configure Twingate Provider
provider "twingate" {
  api_token = "ENTER API TOKEN"
  network   = "ENTER YOUR NETWORK NAME"
}
```

Fig. 5 Configure Twingate

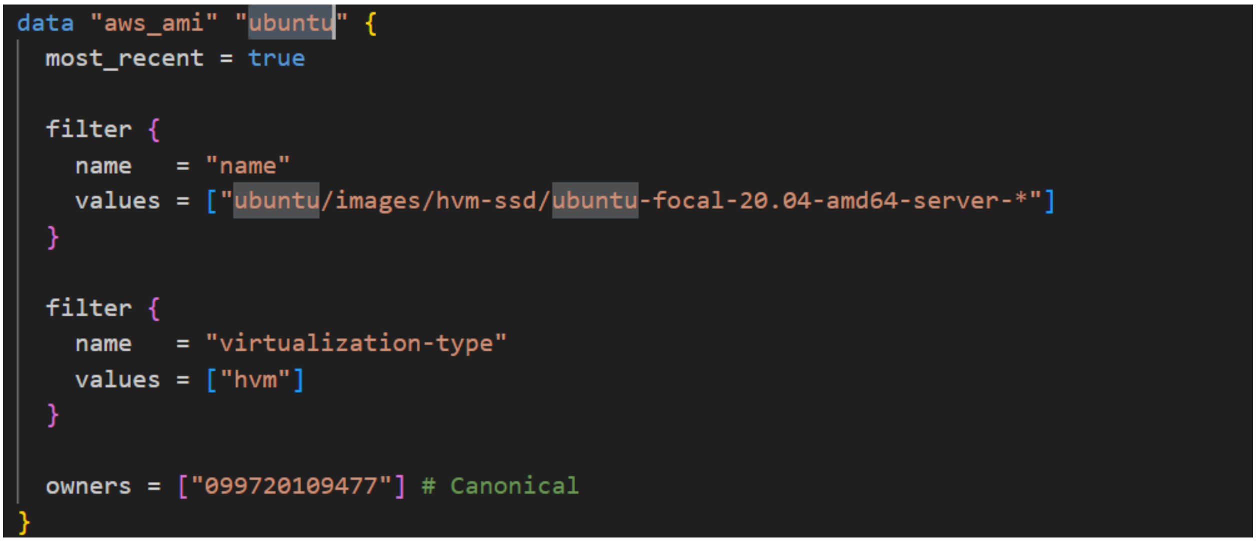
- Create SSH key using **SSH-keygen** and use its location path in code as shown in Fig. 6 we will use it in our EC2 resource as shown in Fig. 7



```
resource "aws_key_pair" "ssh_access_key" {
  key_name   = "~/.ssh/id_ed25519"
  public_key = file("~/.ssh/id_ed25519.pub")
}
```

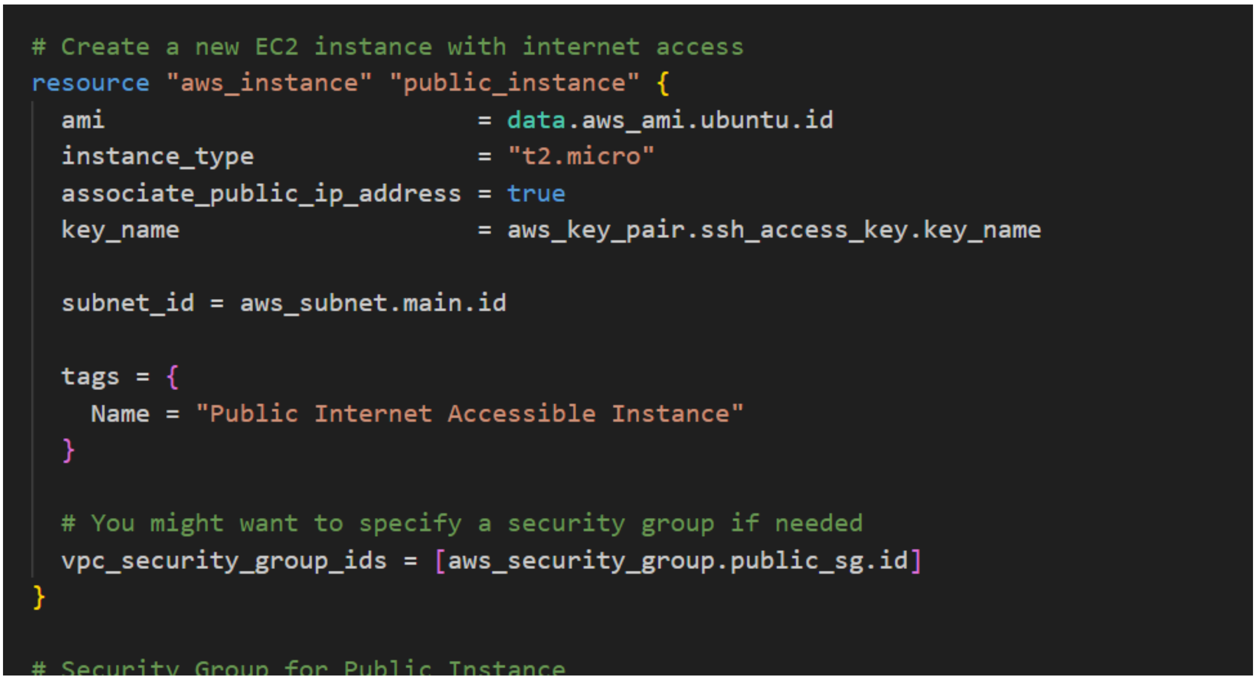
Fig.6 Configure SSH for AWS Instance

Get any free image for AWS instance with AMI Catalog from AWS and paste its ID in code as shown in Fig. 7. AMI Catalog An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance as shown in figure .The installed EC2 instance will be using to deploying our services as shown in Fig. 8



```
data "aws_ami" "ubuntu" {
  most_recent = true

  filter {
    name   = "name"
    values = ["ubuntu/images/hvm-ssd/ubuntu-focal-20.04-amd64-server-*"]
  }

  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }

  owners = ["099720109477"] # Canonical
}
```

Fig.7 Virtual Machine Image for EC2 Instance



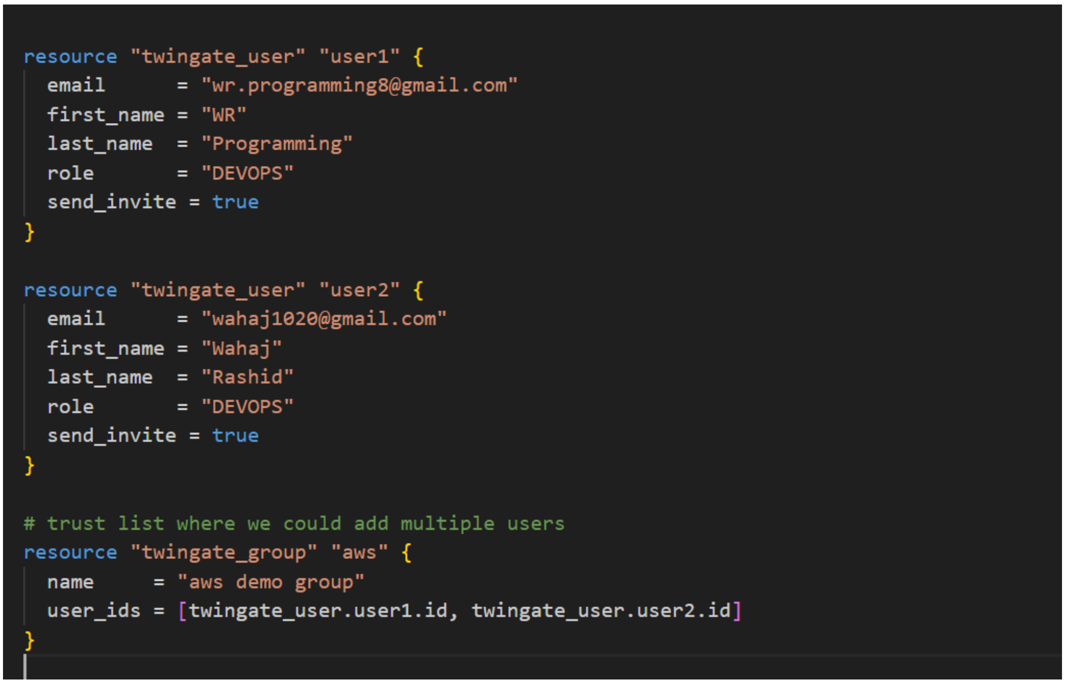
```
# Create a new EC2 instance with internet access
resource "aws_instance" "public_instance" {
  ami                         = data.aws_ami.ubuntu.id
  instance_type               = "t2.micro"
  associate_public_ip_address = true
  key_name                    = aws_key_pair.ssh_access_key.key_name

  subnet_id = aws_subnet.main.id

  tags = {
    Name = "Public Internet Accessible Instance"
  }

  # You might want to specify a security group if needed
  vpc_security_group_ids = [aws_security_group.public_sg.id]
}

# Security Group for Public Instance
```

Fig.8 AWS EC2 Instance configuration

Rest of the code for twingate connection is same as AWS connection only the values will be different basic setup is available at (How to use terraform with AWS and twingate | docs, n.d.)

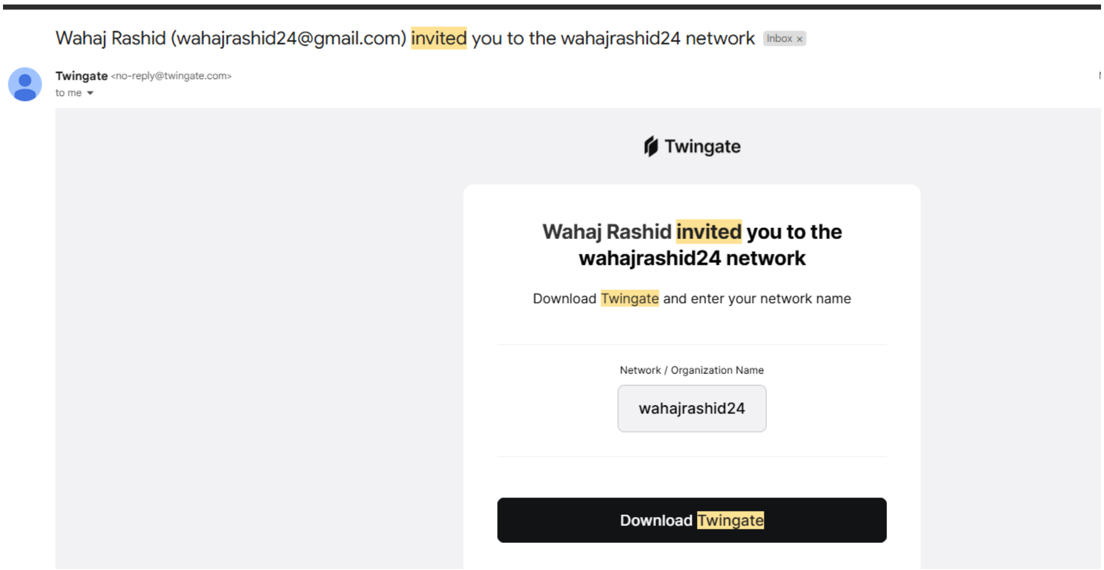
After pasting code and validating that all required elements will be installed and configured after deployment and to deploy user need to run following commands.

- Run **terraform in it** to initializes a working directory that contains installed plugins for required providers and configuration files. Make sure you change the path of SSH key according to your environment before deployment.
- After that run **terraform validate** to spot and verify any error
- After that **terraform plan** to view the changes in infrastructure without deploying the code
- At Last run **terraform apply** to deploy the code and it will configure everything that has mentioned in the code and sent invite to the users with access as shown in Fig. 9 and Fig 10 also you can use the whole group of people to send invite at once through trust list



```
resource "twingate_user" "user1" {
  email      = "wr.programming8@gmail.com"
  first_name = "WR"
  last_name  = "Programming"
  role       = "DEVOPS"
  send_invite = true
}

resource "twingate_user" "user2" {
  email      = "wahaj1020@gmail.com"
  first_name = "Wahaj"
  last_name  = "Rashid"
  role       = "DEVOPS"
  send_invite = true
}

# trust list where we could add multiple users
resource "twingate_group" "aws" {
  name     = "aws demo group"
  user_ids = [twingate_user.user1.id, twingate_user.user2.id]
}
```

Fig. 9 Mentioned Users with their assigned roles



Wahaj Rashid (wahajrashid24@gmail.com) invited you to the wahajrashid24 network

Twingate <no-reply@twingate.com>
to me

Wahaj Rashid invited you to the wahajrashid24 network

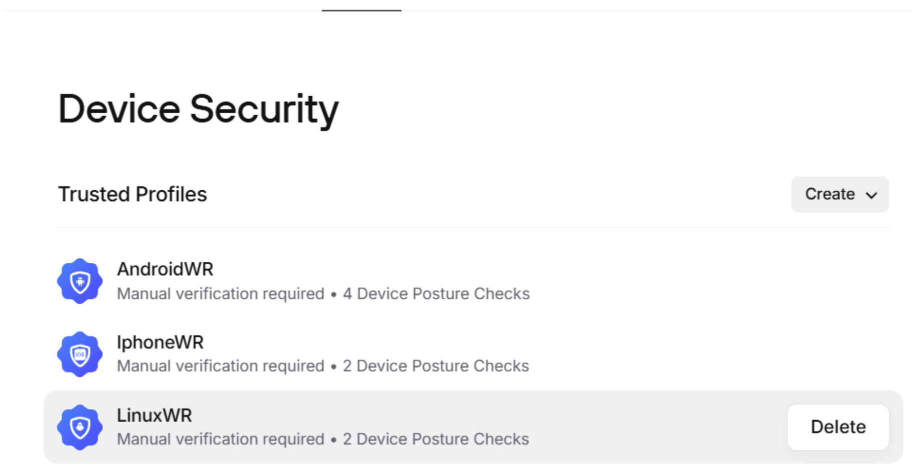
Download Twingate and enter your network name

Network / Organization Name: wahajrashid24

Fig. 10 Accessed user get Invitation as mentioned in code

4. POST DEPLOYMENT STEPS

Admin Needs to Verify the desired environment for the user's device by using Twingate Management Web Console as shown in Fig 11



Device Security

Trusted Profiles

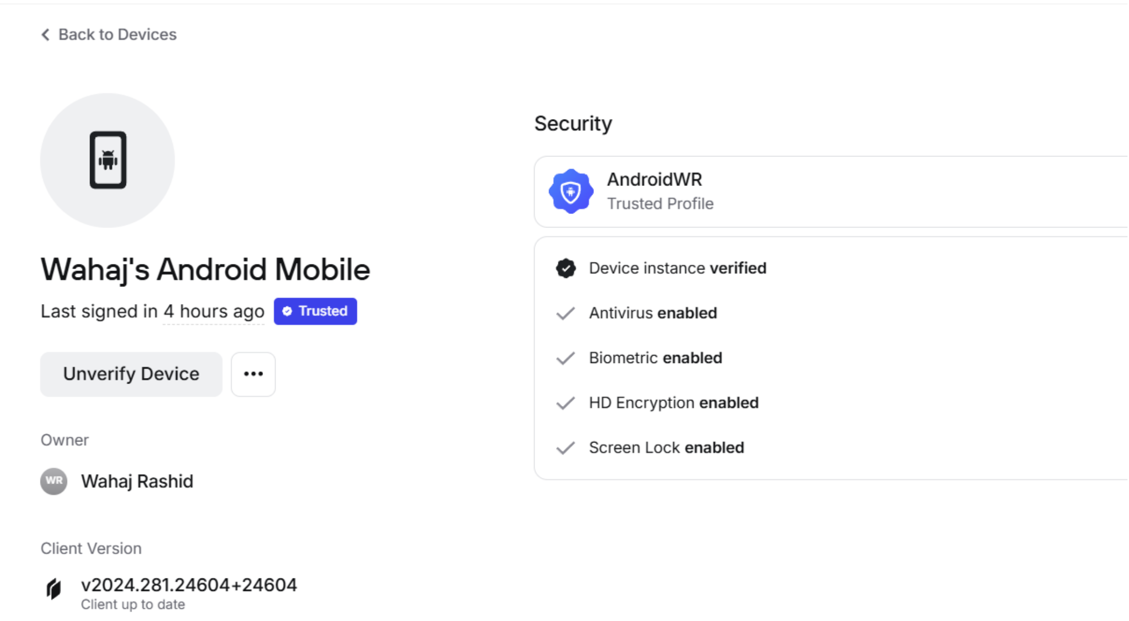
AndroidWR
Manual verification required • 4 Device Posture Checks

IphoneWR
Manual verification required • 2 Device Posture Checks

LinuxWR
Manual verification required • 2 Device Posture Checks

Fig 11 Desired environment for trusted device

Admin Need to verify user device as well as shown in Fig 12. After verifying devices multiple security elements would be enabled for the user like Biometric, HD Encryption and Anti-Virus scan etc.



Wahaj's Android Mobile

Last signed in 4 hours ago Trusted

Owner

Wahaj Rashid

Client Version

v2024.281.24604+24604
Client up to date

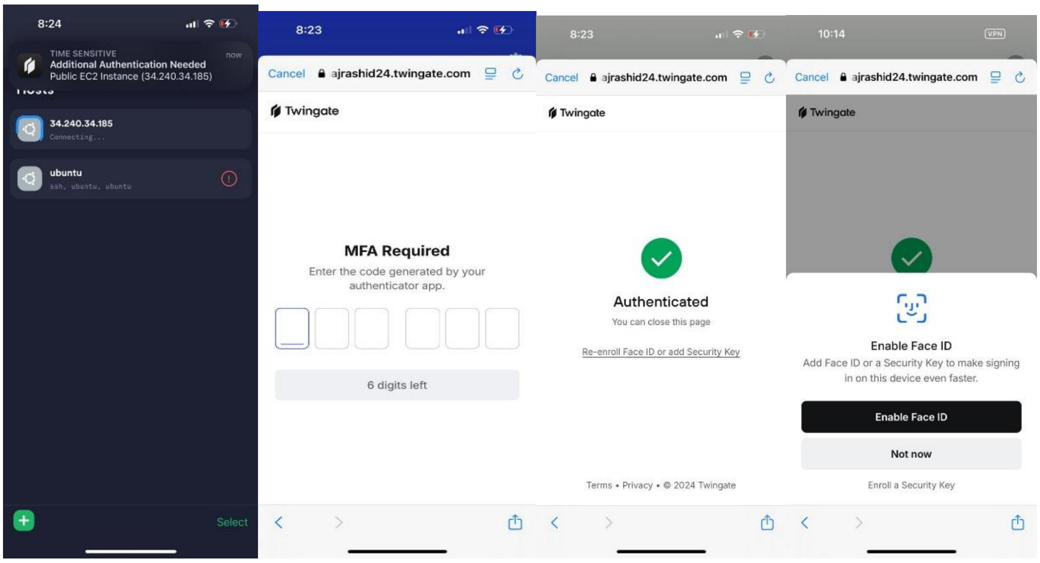
Security

AndroidWR
Trusted Profile

Device instance verified
✓ Antivirus enabled
✓ Biometric enabled
✓ HD Encryption enabled
✓ Screen Lock enabled

Fig 12 Verify/Unverified user with security elements

User open application twingate client and if user and its device is verified by the admin then user will go through MFA first and after that user could set Biometric password less Login for ease. Client copy IP of desired resource Client open Termius (Termius - SSH platform for mobile and desktop, n.d.) and paste IP with SSH key that was shown in Fig. 6 to successfully access the resource through terminal as shown in the Fig 13 and Fig 14



Additional Authentication Needed
Public EC2 Instance (34.240.34.185)

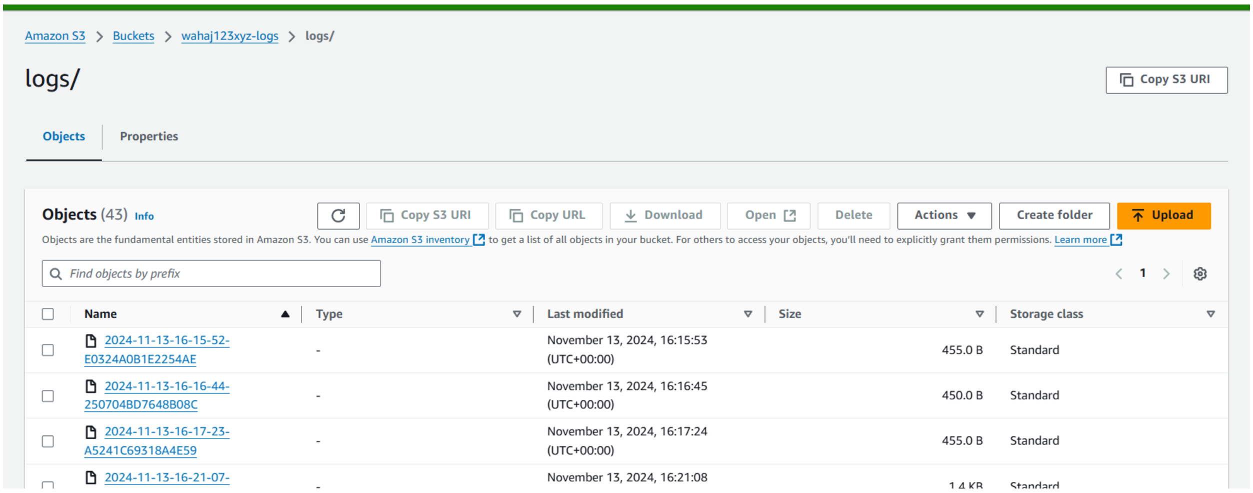
MFA Required
Enter the code generated by your authenticator app.

Authenticated
You can close this page

Enable Face ID
Add Face ID or a Security Key to make signing in on this device even faster.

Fig 14 User authentication process with biometric

S3 bucket was generated by the time of deployment and a separate log file will generate logs if we perform any action in the bucket like create update or upload files in bucket as shown in Fig 15

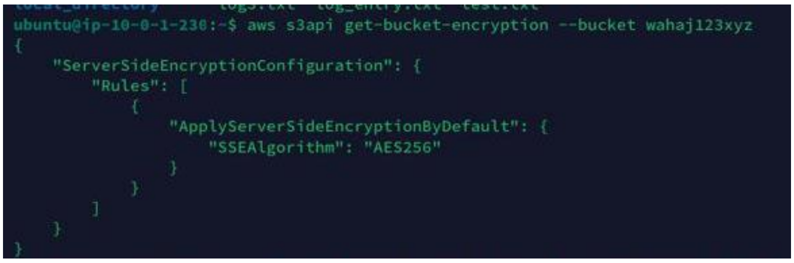


Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

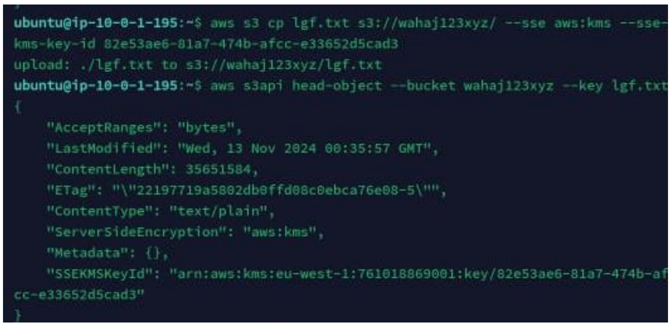
□	Name ▲	Type	Last modified	Size	Storage class
□	2024-11-13-16-15-52-E0324A0B1E2254AE	-	November 13, 2024, 16:15:53 (UTC+00:00)	455.0 B	Standard
□	2024-11-13-16-16-44-250704BD7648B08C	-	November 13, 2024, 16:16:45 (UTC+00:00)	450.0 B	Standard
□	2024-11-13-16-17-23-A5241C69318A4E59	-	November 13, 2024, 16:17:24 (UTC+00:00)	455.0 B	Standard
□	2024-11-13-16-21-07-	-	November 13, 2024, 16:21:08	1.4 KB	Standard

Fig 15 Automated bucket logs to monitor the main S3 Bucket

Data in the file is encrypted by AES256 with Twingate Policies by default as shown in figure and it could be further enhanced by using AWS KMS as shown in Fig 16 just by running AWS s3 cp lgf.txt s3://wahaj123xyz/ --sse AWS:kms



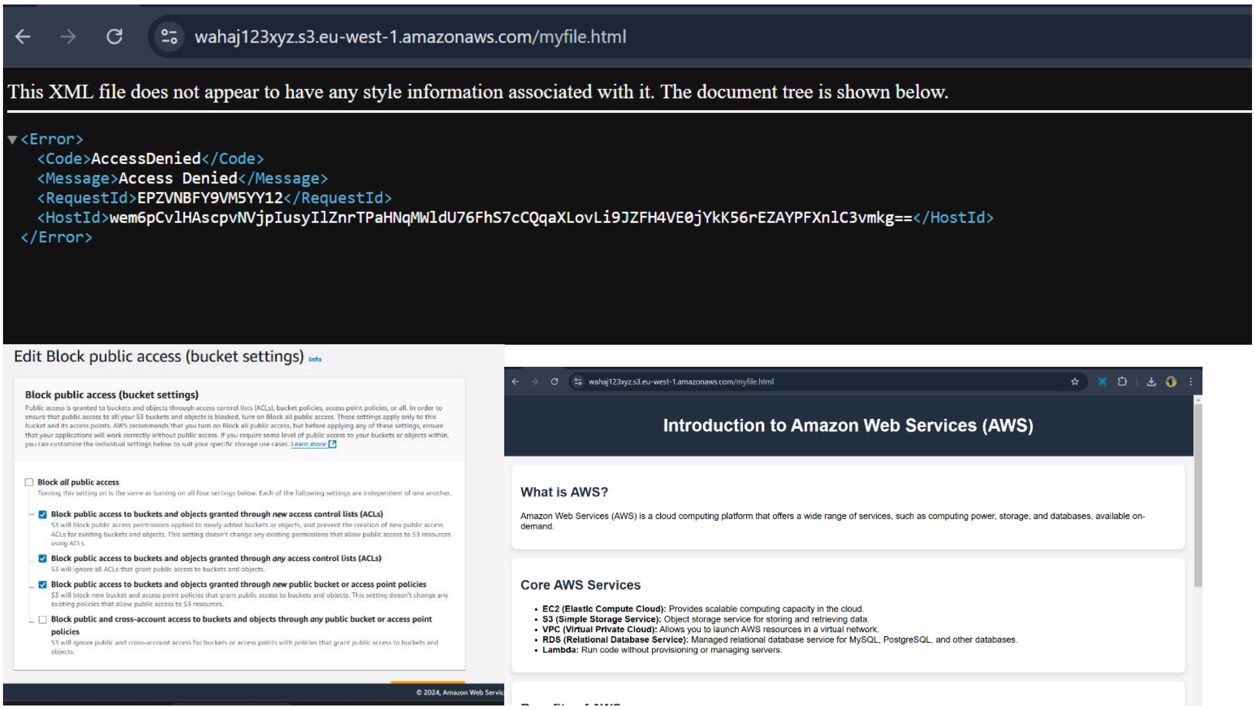
```
ubuntu@ip-10-0-1-238:~$ aws s3api get-bucket-encryption --bucket wahaj123xyz
{
    "ServerSideEncryptionConfiguration": {
        "Rules": [
            {
                "ApplyServerSideEncryptionByDefault": {
                    "SSEAlgorithm": "AES256"
                }
            }
        ]
    }
}
```



```
ubuntu@ip-10-0-1-195:~$ aws s3 cp lgf.txt s3://wahaj123xyz/ --sse aws:kms --sse-
kms-key-id 82e53ae6-81a7-474b-afcc-e33652d5cad3
upload: ./lgf.txt to s3://wahaj123xyz/lgf.txt
ubuntu@ip-10-0-1-195:~$ aws s3api head-object --bucket wahaj123xyz --key lgf.txt
{
    "AcceptRanges": "bytes",
    "LastModified": "Wed, 13 Nov 2024 00:35:57 GMT",
    "ContentLength": 35651584,
    "ETag": "\"22197719a5802db0ffd08c0ebca76e08-5\"",
    "ContentType": "text/plain",
    "ServerSideEncryption": "aws:kms",
    "Metadata": {},
    "SSEKMSKeyId": "arn:aws:kms:eu-west-1:761018869001:key/82e53ae6-81a7-474b-af
cc-e33652d5cad3"
}
```

Fig 16 Encryption type from AES256 to KMS

Defined policies in code for ACL will not let other users to access the file but it could be update according to the user desire as shown in Fig 17.



wahaj123xyz.s3.eu-west-1.amazonaws.com/myfile.html

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>EPZVNBFY9VM5YY12</RequestId>
   <HostId>wem6pCvlHAscpvNVjpIusyIlZnrTPaHNqMWldU76FhS7cCQqaXLovLi9JZFH4VE0jYkK56rEZAYPFXnlC3vmkg==</HostId>
 </Error>
```

Edit Block public access (bucket settings) Info

Block public access (bucket settings)

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to all your S3 buckets and objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to your buckets or objects within, you can customize the individual settings below to suit your specific storage use cases. Learn more

□ **Block *all* public access**
Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

- ☑ **Block public access to buckets and objects granted through *new* access control lists (ACLs)**
  S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
- ☑ **Block public access to buckets and objects granted through *any* access control lists (ACLs)**
  S3 will ignore all ACLs that grant public access to buckets and objects.
- ☑ **Block public access to buckets and objects granted through *new* public bucket or access point policies**
  S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.
- □ **Block public and cross-account access to buckets and objects through *any* public bucket or access point policies**
  S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

© 2024, Amazon Web Servic

wahaj123xyz.s3.eu-west-1.amazonaws.com/myfile.html

Introduction to Amazon Web Services (AWS)

What is AWS?

Amazon Web Services (AWS) is a cloud computing platform that offers a wide range of services, such as computing power, storage, and databases, available on-demand.

Core AWS Services

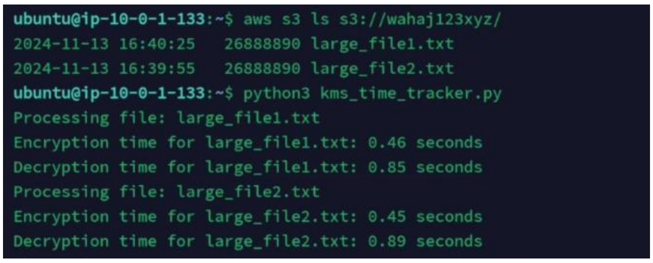
- **EC2 (Elastic Compute Cloud):** Provides scalable computing capacity in the cloud.
- **S3 (Simple Storage Service):** Object storage service for storing and retrieving data.
- **VPC (Virtual Private Cloud):** Allows you to launch AWS resources in a virtual network.
- **RDS (Relational Database Service):** Managed relational database service for MySQL, PostgreSQL, and other databases.
- **Lambda:** Run code without provisioning or managing servers.

Fig 17 Integrated ACL to restrict public access

5. ANALYSIS AND EVALUATION STEPS

To analyze and evaluate we used python scripts to generate some data and compare some of them to find the one with better results .

To track the time taken to encrypt and decrypt kms file use code mentioned below in a file that you will create by using **sudo nano kms_time_tracker.py** and after saving it run **python3 kms_time_tracker.py** to execute as shown in Fig 18.

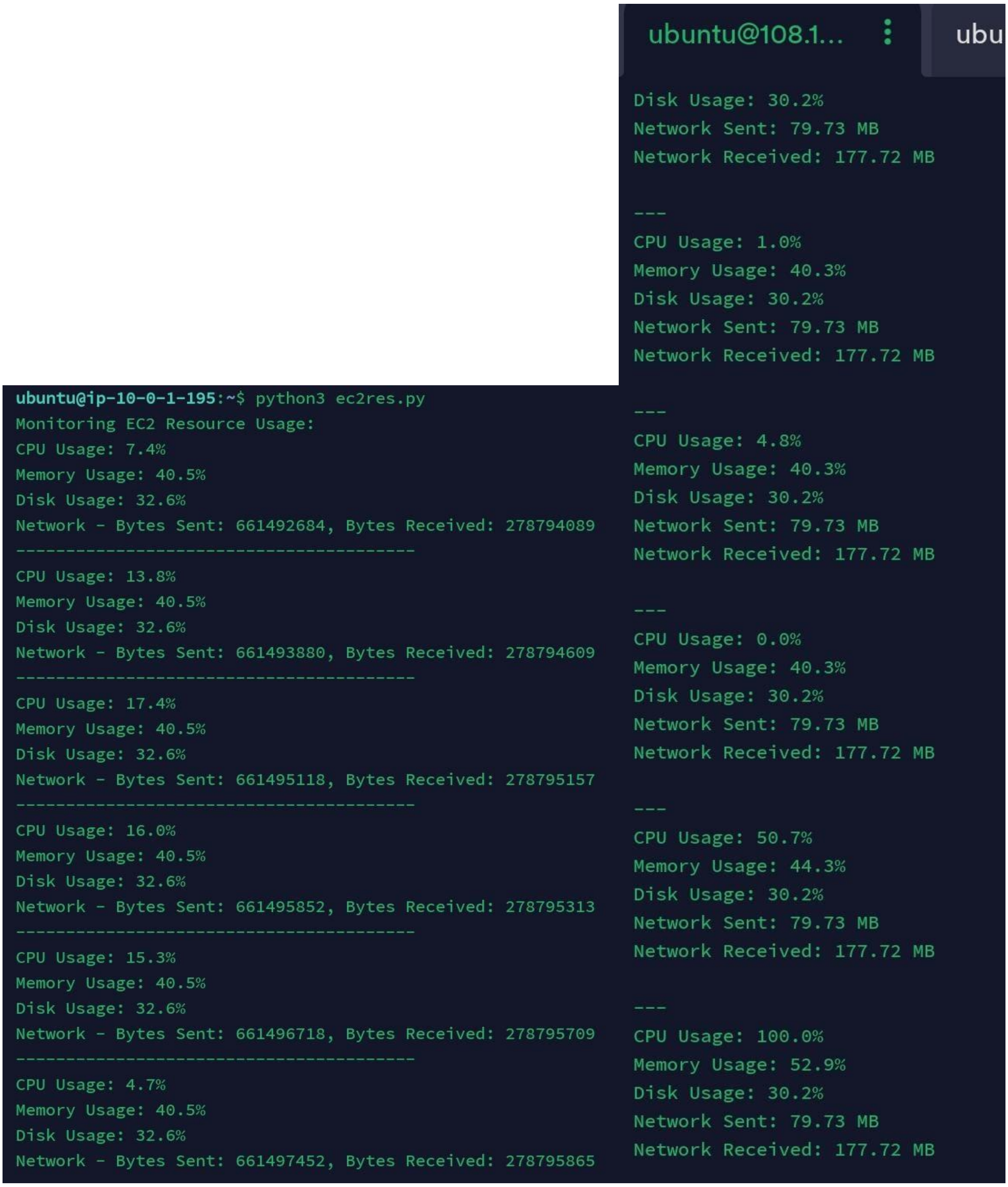


```
ubuntu@ip-10-0-1-133:~$ aws s3 ls s3://wahaj123xyz/
2024-11-13 16:40:25   26888890 large_file1.txt
2024-11-13 16:39:55   26888890 large_file2.txt
ubuntu@ip-10-0-1-133:~$ python3 kms_time_tracker.py
Processing file: large_file1.txt
Encryption time for large_file1.txt: 0.46 seconds
Decryption time for large_file1.txt: 0.85 seconds
Processing file: large_file2.txt
Encryption time for large_file2.txt: 0.45 seconds
Decryption time for large_file2.txt: 0.89 seconds
```

Fig 18 Evaluating time to encrypt and decrypt S3 file using KMS

```
import subprocess
import time
import os

# S3 Bucket Name
bucket_name = 'wahaj123xyz'

# Function to get the list of files in the S3 bucket
def list_files():
    result = subprocess.run(
        ["AWS", "s3", "ls", f"s3://{bucket_name}/", "--recursive"],
        stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True
    )
    files = result.stdout.strip().splitlines()
    return [file.split()[-1] for file in files]  # Extract file paths

# Function to measure the time it takes to upload (encrypt) a file
def measure_encryption_time(file_path):
    start_time = time.time()
    subprocess.run(
        ["AWS", "s3", "cp", f"{file_path}", f"s3://{bucket_name}/", "--sse", "AWS:kms", "--sse-kms-key-id", "82e53ae6-81a7-474b-afcc-
e33652d5cad3"],
        stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True
    )
    end_time = time.time()
    return end_time - start_time

# Function to measure the time it takes to download (decrypt) a file
def measure_decryption_time(file_path):
    start_time = time.time()
    subprocess.run(
        ["AWS", "s3", "cp", f"s3://{bucket_name}/{file_path}", f"{file_path}", "--sse", "AWS:kms", "--sse-kms-key-id", "82e53ae6-81a7-474b-afcc-
e33652d5cad3"],
        stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True
    )
    end_time = time.time()
    return end_time - start_time

# Main function to iterate over all files and measure encryption/decryption times
def main():
    files = list_files()
    for file in files:
        print(f"Processing file: {file}")

        # Measure encryption time
        encryption_time = measure_encryption_time(file)
        print(f"Encryption time for {file}: {encryption_time:.2f} seconds")

        # Measure decryption time
        decryption_time = measure_decryption_time(file)
        print(f"Decryption time for {file}: {decryption_time:.2f} seconds")

if __name__ == "__main__":
    main()
```

In the same way paste the code mentioned below to monitor the resources in EC2 to compare the public and private instance to measure performance while uploading big files as shown in Fig 19.



```
ubuntu@ip-10-0-1-195:~$ python3 ec2res.py
Monitoring EC2 Resource Usage:
CPU Usage: 7.4%
Memory Usage: 40.5%
Disk Usage: 32.6%
Network - Bytes Sent: 661492684, Bytes Received: 278794089
----------------------------------------
CPU Usage: 13.8%
Memory Usage: 40.5%
Disk Usage: 32.6%
Network - Bytes Sent: 661493880, Bytes Received: 278794609
----------------------------------------
CPU Usage: 17.4%
Memory Usage: 40.5%
Disk Usage: 32.6%
Network - Bytes Sent: 661495118, Bytes Received: 278795157
----------------------------------------
CPU Usage: 16.0%
Memory Usage: 40.5%
Disk Usage: 32.6%
Network - Bytes Sent: 661495852, Bytes Received: 278795313
----------------------------------------
CPU Usage: 15.3%
Memory Usage: 40.5%
Disk Usage: 32.6%
Network - Bytes Sent: 661496718, Bytes Received: 278795709
----------------------------------------
CPU Usage: 4.7%
Memory Usage: 40.5%
Disk Usage: 32.6%
Network - Bytes Sent: 661497452, Bytes Received: 278795865
```

```
Disk Usage: 30.2%
Network Sent: 79.73 MB
Network Received: 177.72 MB

---
CPU Usage: 1.0%
Memory Usage: 40.3%
Disk Usage: 30.2%
Network Sent: 79.73 MB
Network Received: 177.72 MB

---
CPU Usage: 4.8%
Memory Usage: 40.3%
Disk Usage: 30.2%
Network Sent: 79.73 MB
Network Received: 177.72 MB

---
CPU Usage: 0.0%
Memory Usage: 40.3%
Disk Usage: 30.2%
Network Sent: 79.73 MB
Network Received: 177.72 MB

---
CPU Usage: 50.7%
Memory Usage: 44.3%
Disk Usage: 30.2%
Network Sent: 79.73 MB
Network Received: 177.72 MB

---
CPU Usage: 100.0%
Memory Usage: 52.9%
Disk Usage: 30.2%
Network Sent: 79.73 MB
Network Received: 177.72 MB
```

Fig 19 Public VS Private EC2 instance while uploading big files in S3 bucket

```
import psutil
import time

def get_cpu_usage():
    cpu_usage = psutil.cpu_percent(interval=1)
    return cpu_usage

def get_memory_usage():
    memory_info = psutil.virtual_memory()
    return memory_info.percent

def get_disk_usage():
    disk_info = psutil.disk_usage('/')
    return disk_info.percent

def get_network_usage():
    network_info = psutil.net_io_counters()
    return network_info.bytes_sent, network_info.bytes_recv

def display_usage():
    print("Checking EC2 resource usage...\n")
    while True:
        cpu = get_cpu_usage()
        memory = get_memory_usage()
        disk = get_disk_usage()
        network_sent, network_recv = get_network_usage()

        print(f"CPU Usage: {cpu}%")
        print(f"Memory Usage: {memory}%")
        print(f"Disk Usage: {disk}%")
        print(f"Network Sent: {network_sent / (1024 * 1024):.2f} MB")
        print(f"Network Received: {network_recv / (1024 * 1024):.2f} MB")
        print("\n---")

        time.sleep(5)  # Update every 5 seconds

if __name__ == "__main__":
    display_usage()
```

REFERENCES

How to use terraform with AWS and twingate | docs (n.d.). Available at: https://www.twingate.com/docs/terraform-AWS (Accessed: 11 December 2024).

Install | terraform | hashicorp developer (n.d.) Install | Terraform | HashiCorp Developer. Available at: https://developer.hashicorp.com/terraform/install (Accessed: 11 December 2024).

Installing or updating to the latest version of the AWS cli - AWS command line interface (n.d.). Available at: https://docs.AWS.amazon.com/cli/latest/userguide/getting-started-install.html#getting-started-install-instructions (Accessed: 11 December 2024).

Termius - SSH platform for mobile and desktop (n.d.). Available at: https://termius.com/ (Accessed: 11 December 2024).

Twingate (n.d.). Available at: https://auth.twingate.com/signup-v2 (Accessed: 11 December 2024).