Name : Ganesh Gupta **Div : D15C** **RollNo.: 13**

Case Study

Basic Infrastructure Management with Terraform

- **Concepts Used**: Terraform, AWS EC2, and S3.
- **Problem Statement**: "Use Terraform to create an EC2 instance and an S3 bucket. Store the EC2 instance's IP address in the S3 bucket."
- **Tasks**:
  - Write a simple Terraform script to provision an EC2 instance and an S3 bucket.
  - Use Terraform outputs to extract the EC2 instance's IP address.
  - Store the IP address in a text file in the S3 bucket.

Introduction

Case Study Overview:

In this case study, we delve into the realm of Basic Infrastructure Management with Terraform by leveraging Terraform's Infrastructure-as-Code (IaC) capabilities to automate the provisioning and management of cloud resources on AWS. Specifically, the case study focuses on deploying an EC2 instance (virtual machine) and an S3 bucket (storage service) on AWS using Terraform, highlighting the power of automation in modern cloud infrastructure.

The problem statement presents a real-world scenario where cloud resources need to be set up automatically without manual intervention, ensuring that the infrastructure is consistent, scalable, and easy to manage. Terraform serves as the backbone for achieving this by enabling declarative configuration for infrastructure provisioning. This case demonstrates the process of creating an EC2 instance and S3 bucket, extracting the public IP address of the EC2 instance, and storing it as a text file within the S3 bucket, illustrating a common use case in cloud environments for automating cloud resource management.

The goal is to simplify the traditionally complex task of setting up infrastructure by writing a Terraform script that automates this entire workflow, ensuring efficiency, repeatability, and reduced risk of human error.

EC2 (Elastic Compute Cloud)

Amazon EC2 provides scalable computing capacity in the cloud, allowing users to run virtual servers (instances) for a wide range of applications. EC2 instances offer flexibility in selecting operating systems, instance types, and storage options, enabling customized computing environments. It supports automatic scaling, load balancing, and integration with other AWS services, making it ideal for dynamic workloads. EC2 is widely used for hosting websites, running applications, or processing large datasets. The pay-as-you-go pricing model ensures cost efficiency by billing only for the compute resources used.

S3 (Simple Storage Service) Bucket

Amazon S3 is an object storage service designed to store and retrieve large amounts of data reliably and securely. S3 buckets are containers for objects, which can be any type of file, such as documents, images, or backups. It offers virtually unlimited storage with options for different storage classes based on access frequency and durability needs. S3 integrates with a wide range of AWS services, making it a key component in data pipelines, backups, and content distribution. S3's strong security and access control mechanisms ensure that stored data is well-protected against unauthorized access.

Terraform

Terraform is an open-source Infrastructure as Code (IaC) tool that allows users to define, provision, and manage cloud resources through human-readable configuration files. It supports multiple cloud providers, including AWS, Azure, and Google Cloud, enabling users to automate the setup of infrastructure across different environments. Terraform uses a declarative language, meaning users specify the desired state of resources, and Terraform handles the provisioning process to achieve that state. With features like **state management** and **resource dependencies**, Terraform ensures predictable, consistent deployments, making it ideal for automating cloud infrastructure and reducing manual efforts.

Key Feature and Application:

The standout feature of this case study lies in Terraform’s ability to automate cloud resource management and manage outputs dynamically. This includes Terraform’s capability to create an EC2 instance and S3 bucket, extract the EC2 instance's public IP address, and use that data seamlessly in real-time to update the S3 bucket—all in one coherent, automated process.

1. **Automated Infrastructure Provisioning:**
    - Terraform enables the automation of creating cloud resources. In this case, the provisioning of an EC2 instance and S3 bucket happens through Terraform scripts, eliminating the need to manually set up these services via the AWS Management Console.
    - With a simple configuration file, users can describe the desired state of their infrastructure, and Terraform will handle the details, ensuring resources are created, updated, or destroyed accordingly.
2. **Dynamic Outputs with Terraform:**
    - One of Terraform’s key strengths is its ability to dynamically handle and output information from the created resources. In this case, the public IP address of the EC2 instance is dynamically extracted after the instance is provisioned.
    - Terraform’s output functionality captures this data and makes it available for further use, which in this case is to store it as a file in the S3 bucket.
3. **Integration Between AWS Services (EC2 and S3):**
    - This solution highlights the integration between AWS services, with Terraform acting as the intermediary. The EC2 instance and S3 bucket, although different in their purpose (compute vs storage), are seamlessly connected using Terraform.

- The EC2 instance's IP address is stored in the S3 bucket as a text file. This is a practical example of how cloud services can interact to fulfill specific use cases, such as logging IP addresses for future access, audit trails, or further automated workflows.

Practical Use Case:

This automation approach is particularly useful in scenarios where developers or IT teams need to:

- Regularly spin up new infrastructure (EC2 instances) and track their details, such as IP addresses.
- Centralize and store important details (like IP addresses) in S3 for further processing, auditing, or access.
- Maintain a scalable and easily reproducible infrastructure setup, reducing the risk of human error and saving time on manual cloud configuration.

For instance, a company that frequently deploys new EC2 instances as part of its dynamic scaling strategy can use this approach to automate both provisioning and logging IP addresses in S3, ensuring the information is always up to date and accessible.

By using Terraform to automate these tasks, cloud operations become more streamlined, ensuring that infrastructure is managed efficiently, consistently, and with reduced manual oversight.

Prerequisites:

(If Downloaded then skip):

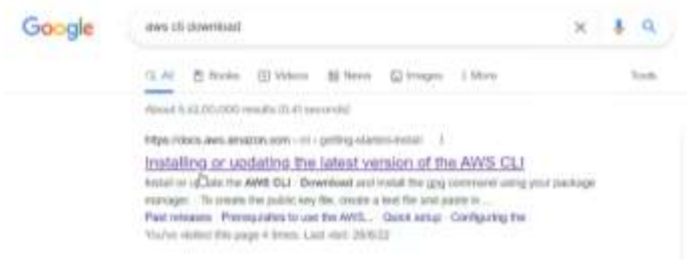
Step 1: Downloading AWS CLI to perform steps:

Download according to your OS as i have windows i downloaded it for windows by clicking the download link

Open the downloaded file and click on next:

Step 1: Install and configure aws CLI

Search for **aws cli download** and click on the link

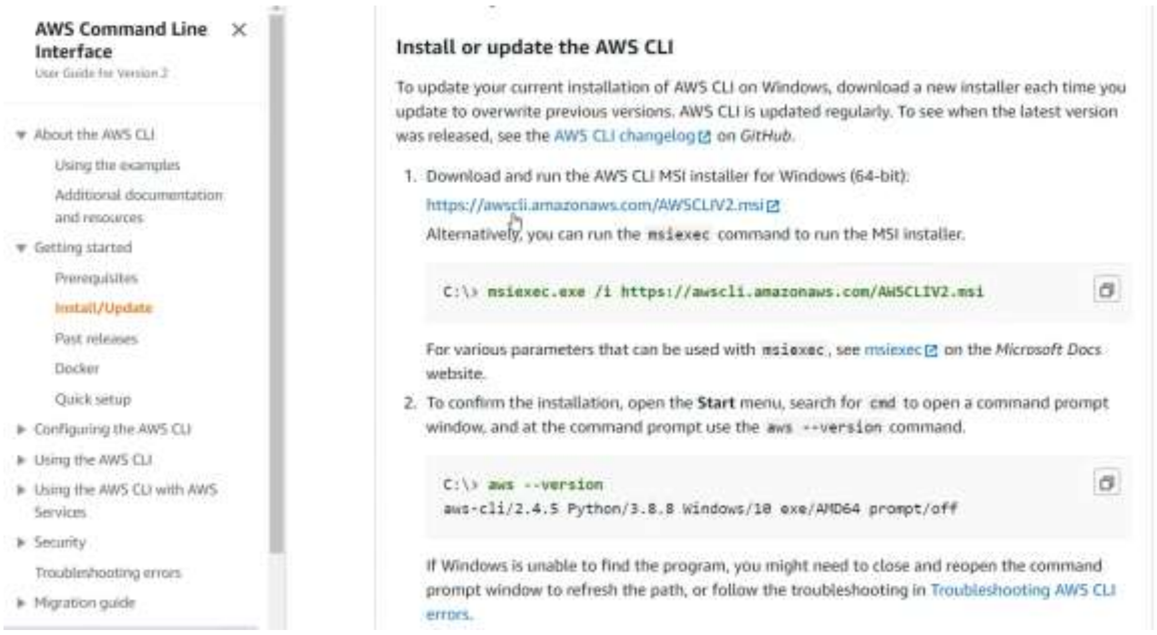

aws cli download

Installing or updating the latest version of the AWS CLI

Install or update the AWS CLI · Download and install the gpg command using your package manager. · To create the public key file, create a text file and paste in ...

Past releases · Prerequisites to use the AWS... · Quick setup · Configuring the

Click on the link for the **msi(Micro Star International) file** the download process of the msi file will start automatically



Install or update the AWS CLI

To update your current installation of AWS CLI on Windows, download a new installer each time you update to overwrite previous versions. AWS CLI is updated regularly. To see when the latest version was released, see the AWS CLI changelog on *GitHub*.

1. Download and run the AWS CLI MSI installer for Windows (64-bit):

   https://awscli.amazonaws.com/AWSCLIV2.msi

   Alternatively, you can run the `msiexec` command to run the MSI installer.

```
C:\> msiexec.exe /i https://awscli.amazonaws.com/AWSCLIV2.msi
```

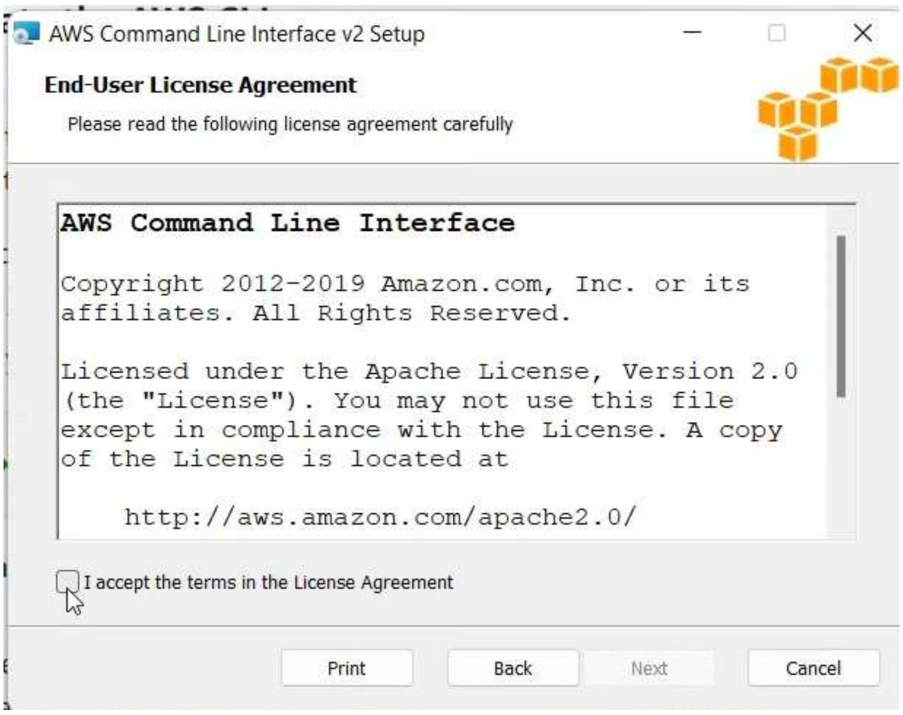
   For various parameters that can be used with `msiexec`, see msiexec on the *Microsoft Docs* website.

2. To confirm the installation, open the **Start** menu, search for `cmd` to open a command prompt window, and at the command prompt use the `aws --version` command.

```
C:\> aws --version
aws-cli/2.4.5 Python/3.8.8 Windows/10 exe/AMD64 prompt/off
```

   If Windows is unable to find the program, you might need to close and reopen the command prompt window to refresh the path, or follow the troubleshooting in Troubleshooting AWS CLI errors.

as you click on the **installed msi** file the following will get displayed click on **accept** the terms in the License Agreement


AWS Command Line Interface v2 Setup
End-User License Agreement
Please read the following license agreement carefully
I accept the terms in the License Agreement
Print
Back
Next
Cancel

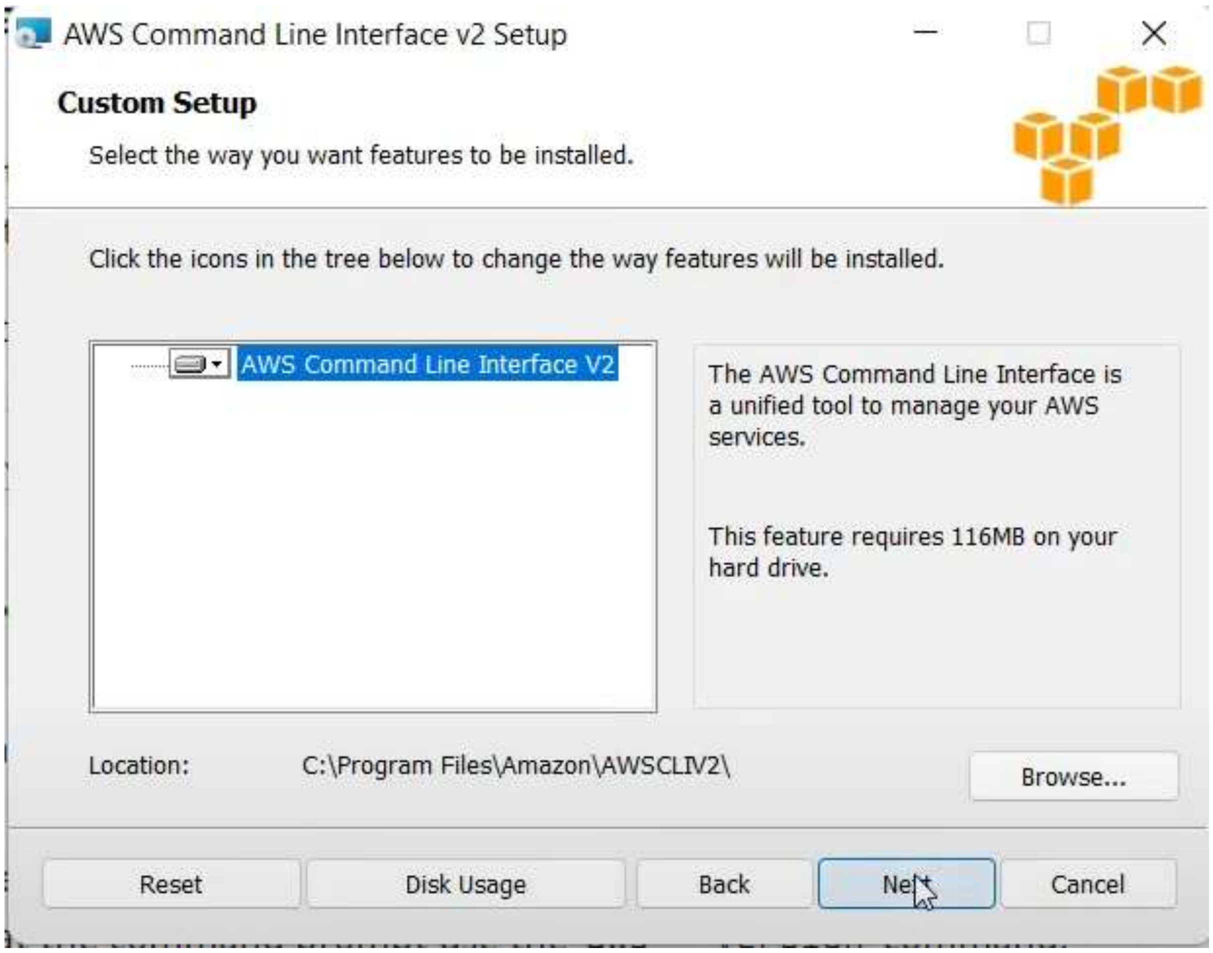
AWS Command Line Interface

Copyright 2012-2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.

Licensed under the Apache License, Version 2.0 (the "License"). You may not use this file except in compliance with the License. A copy of the License is located at

http://aws.amazon.com/apache2.0/

then cick on next



AWS Command Line Interface v2 Setup

Custom Setup

Select the way you want features to be installed.

Click the icons in the tree below to change the way features will be installed.

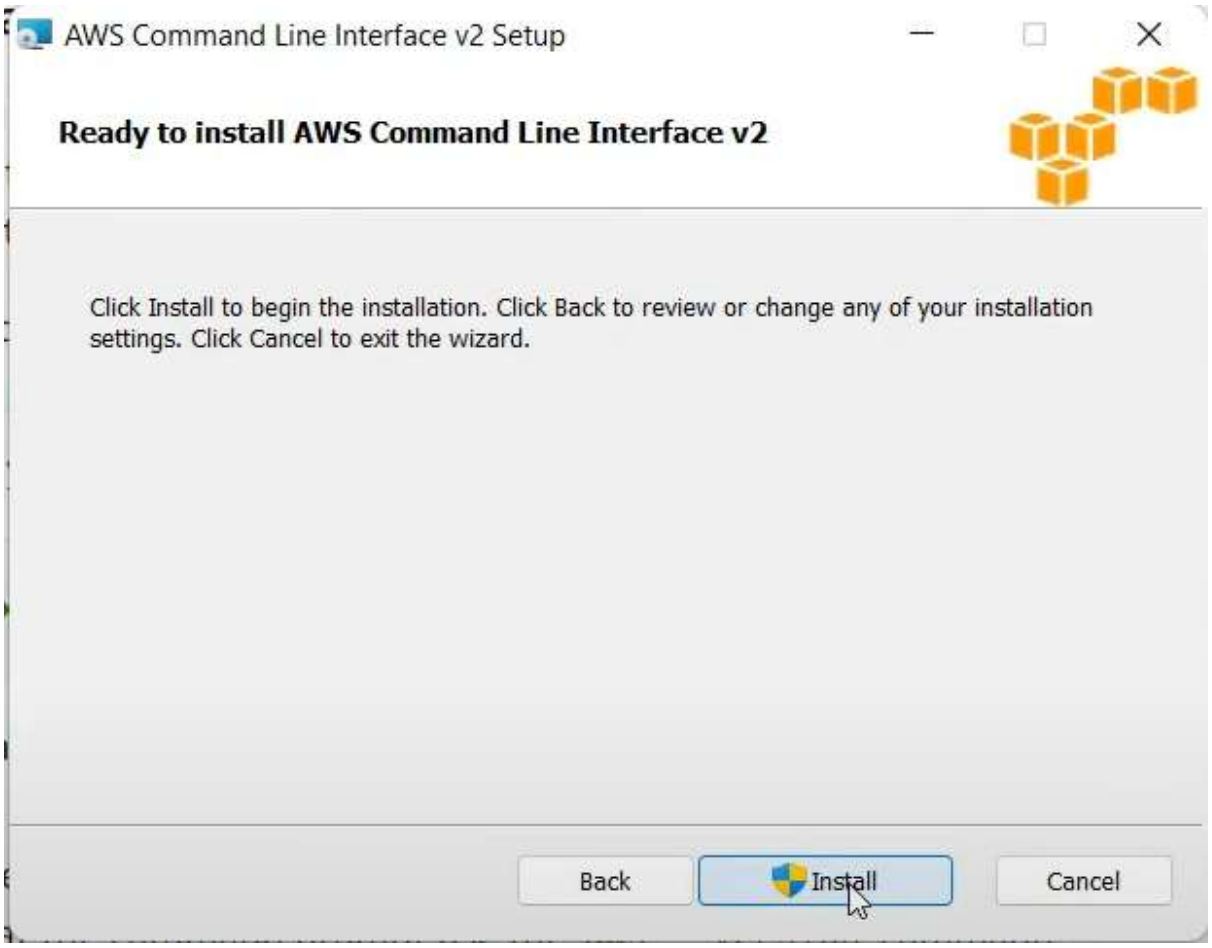
AWS Command Line Interface V2

The AWS Command Line Interface is a unified tool to manage your AWS services.

This feature requires 116MB on your hard drive.

Location: C:\Program Files\Amazon\AWSCLIV2\

then click on **install**

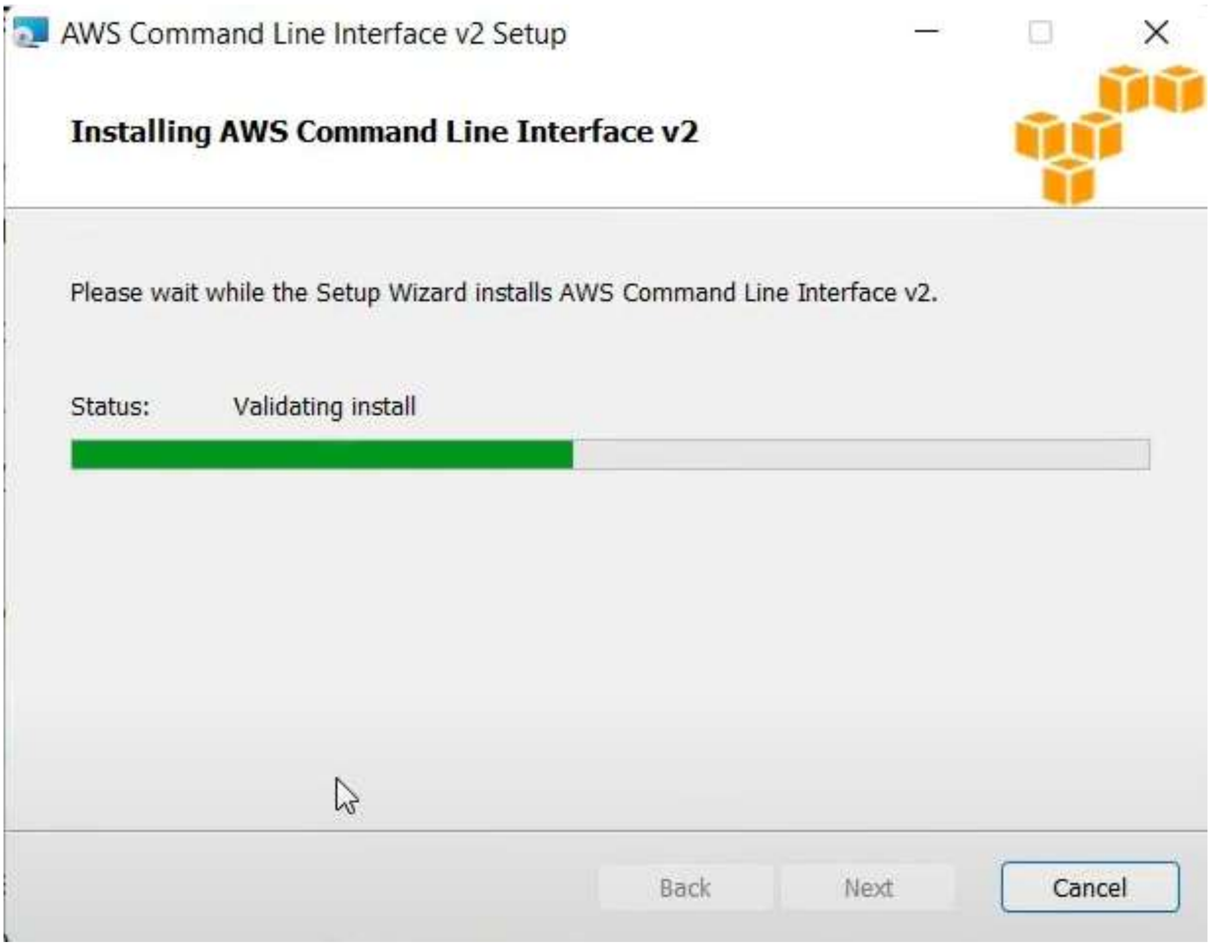


AWS Command Line Interface v2 Setup

Ready to install AWS Command Line Interface v2

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

finally it will start installing **aws cli** on the local machine



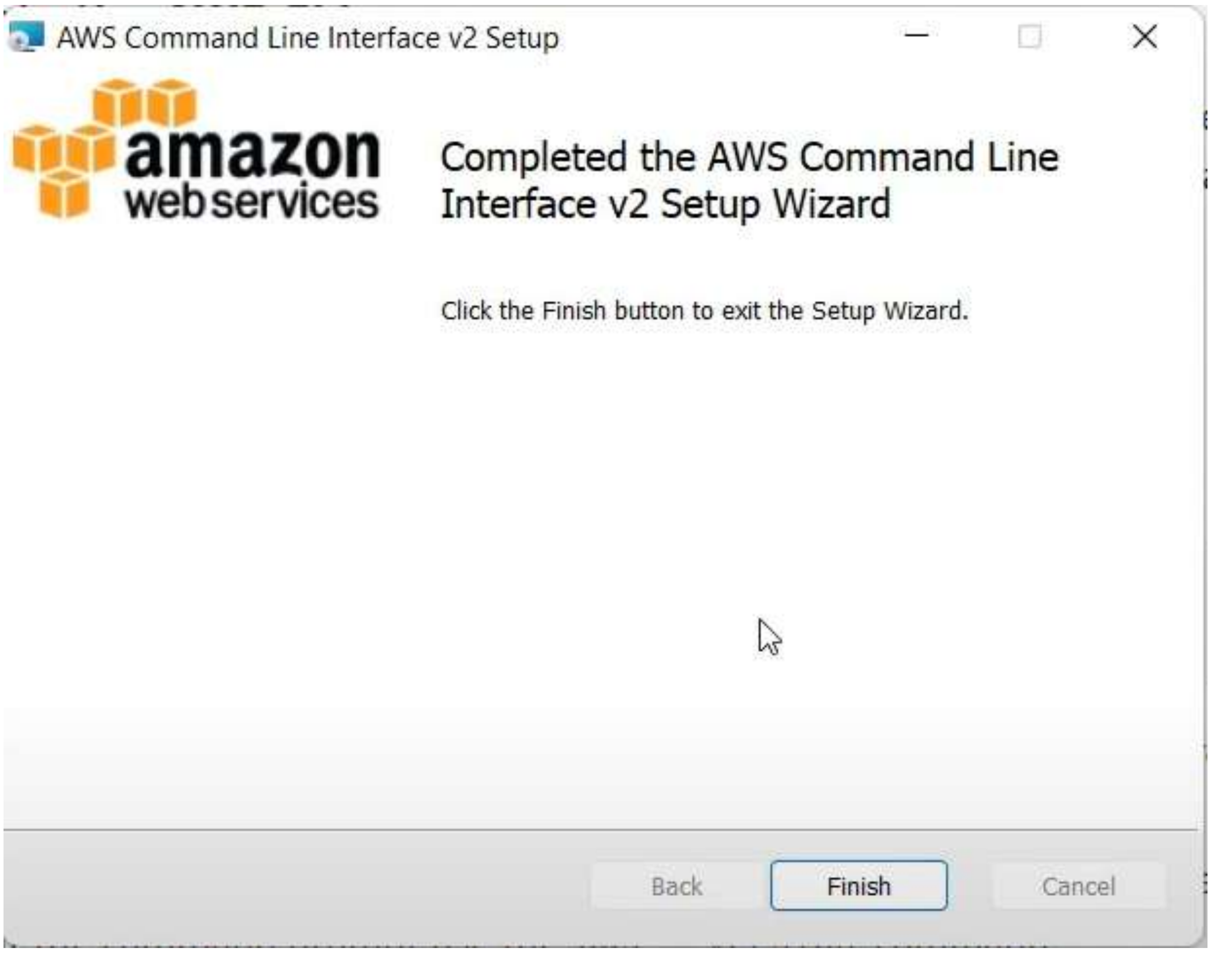
AWS Command Line Interface v2 Setup

Installing AWS Command Line Interface v2

Please wait while the Setup Wizard installs AWS Command Line Interface v2.

Status: Validating install

After installation is finished click on finish

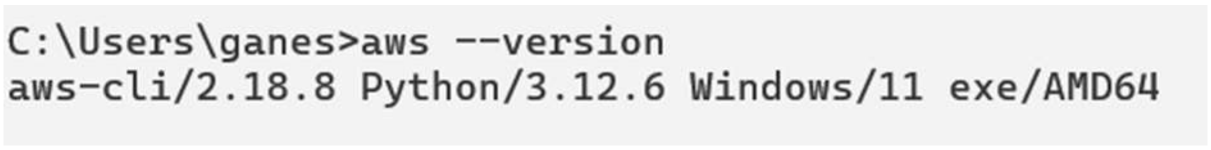


AWS Command Line Interface v2 Setup

Completed the AWS Command Line Interface v2 Setup Wizard

Click the Finish button to exit the Setup Wizard.

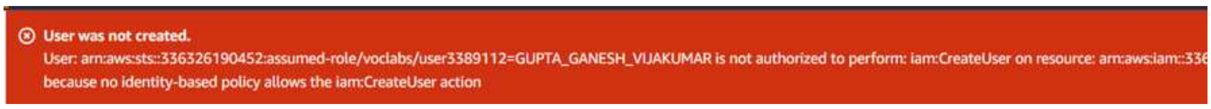
Go to local machine cmd and type command **aws –version** the output should be **aws-cli/2.18.8 Python/3.12.6 Windows/11 exe/AMD64** the versions can be different



```
C:\Users\ganes>aws --version
aws-cli/2.18.8 Python/3.12.6 Windows/11 exe/AMD64
```

In order to **configure aws cli** there are **2 methods** to do so **:**
go on the aws academy site, provided by the college but remember it has only the student access and student doesn't has the access to create new user in IAM if you try to create it the following error will pop up

Method 1(create new user in IAM):

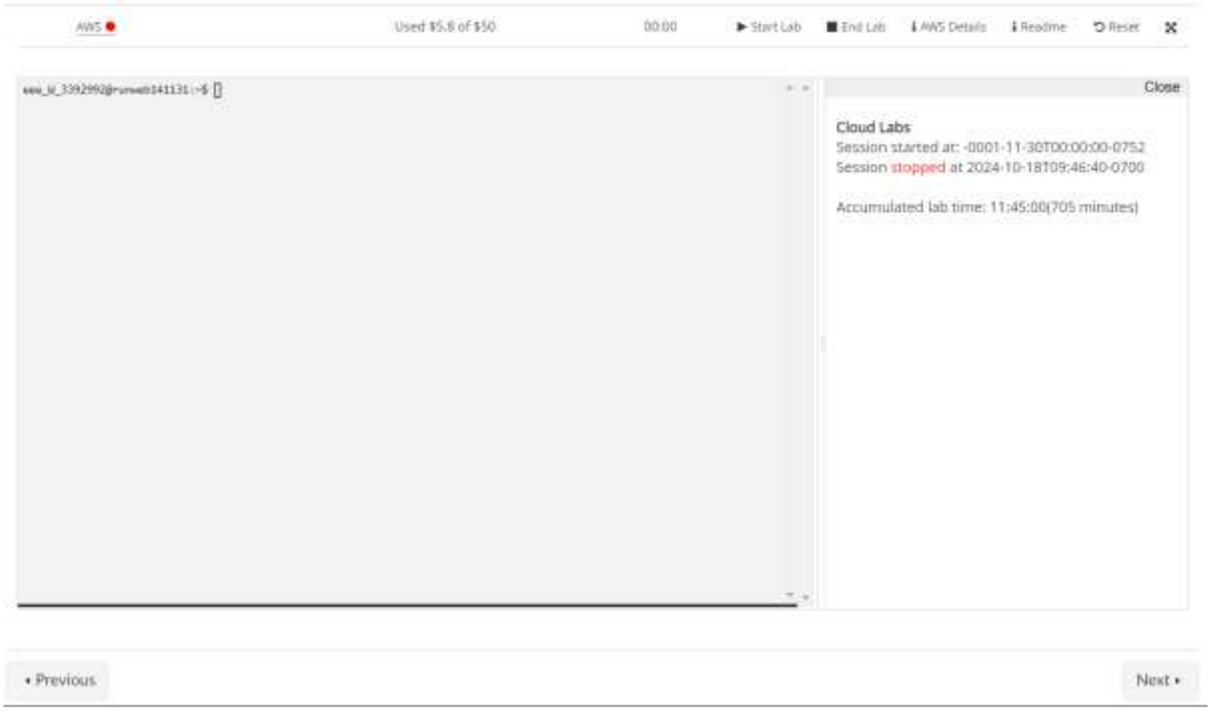


User was not created.
User: arn:aws:sts::336326190452:assumed-role/voclabs/user3389112=GUPTA_GANESH_VIJAKUMAR is not authorized to perform: iam:CreateUser on resource: arn:aws:iam::336
because no identity-based policy allows the iam:CreateUser action

The second method(configure using temporary credentials, note it will be different for each lab session):

a)This is how you can get the credentials

when the lab is not started there are no credentials allotted to the user

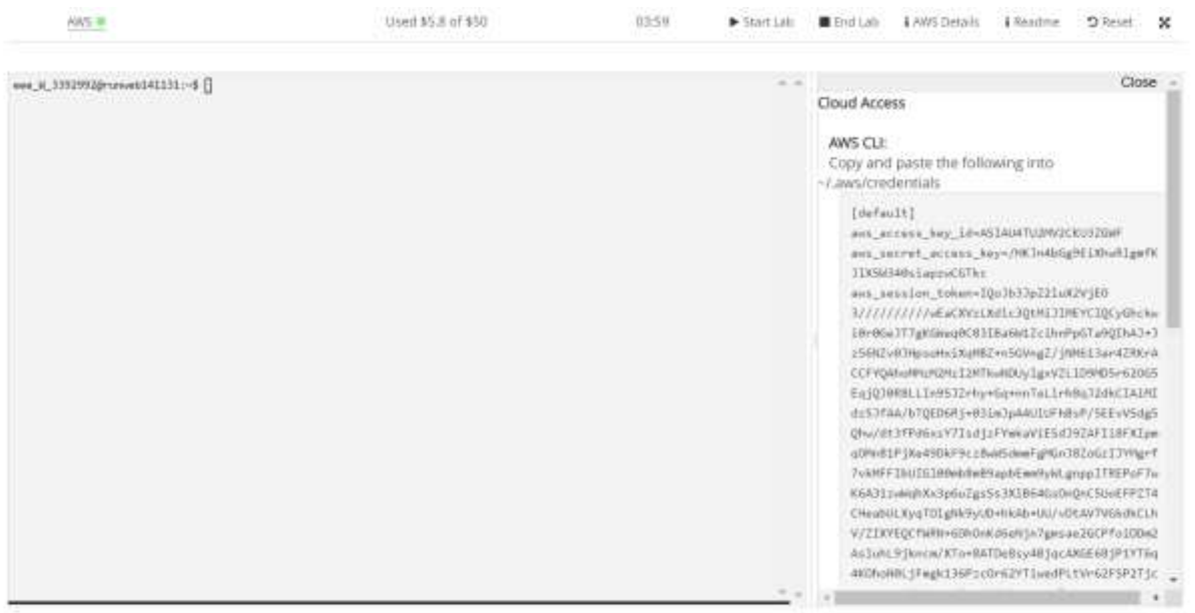


```
eee_W_3392992@runweb141131:~$
```

Cloud Labs
Session started at: -0001-11-30T00:00:00-0752
Session stopped at 2024-10-18T09:46:40-0700

Accumulated lab time: 11:45:00(705 minutes)

b)as we can see there no credentials available here so in order to get the credentials we have to click on start the lab



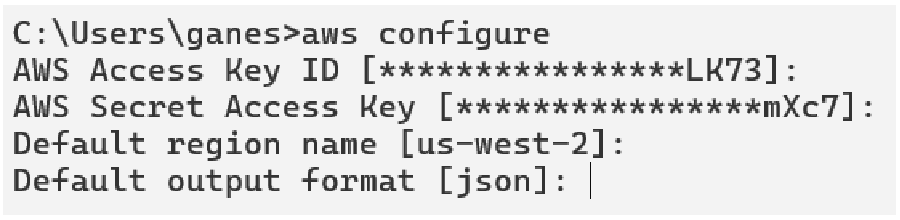
```
aws_ui_3392992@runweb141131:~$
```

Cloud Access

AWS CLI:
Copy and paste the following into ~/.aws/credentials

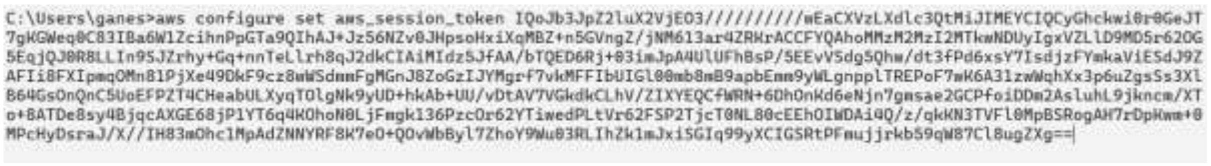
```
[default]
aws_access_key_id=ASIAU4TU2MV2CKU32GWF
aws_secret_access_key=/HKJn4bGg9EiXhu8lgmfK
JIXSW340siapzwC6Tkr
aws_session_token=IQoJb3JpZ2luX2VjEO
3//////////wEaCXVzLXdlc3QtMiJIMEYCIQCyGhckw
i0r0GeJT7gKGWeq0C83IBa6W1ZcihnPpGTa9QIhAJ+J
z56NZv0JHpsoHxiXqMBZ+n5GVngZ/jNM613ar4ZRKrA
CCFYQAhoMMzM2MzI2MTkwNDUyIgxVZLlD9MD5r620G
5EqjQJ0R8LLIn95JZrhy+Gq+nnTeLlrh8qJ2dkCIAiMI
dz5JfAA/bTQED6Rj+03imJpA4UlUFh8sP/5EEvV5dg5
Qhw/dt3fPd6xsY7IsdjzFYmkaViESdJ9ZAFIi8FXIpm
qOMn81PjXe49DkF9cz8wWSdmmFgMGnJ8ZoGzIJYMgrf
7vkMFFIbUIGl00mb8mB9apbEmm9yWLgnpplTREPoF7w
K6A31zwWqhXx3p6uZgsSs3XlB64GsOnQnC5UoEFPZT4
CHeabULXyqTOlgNk9yUD+hkAb+UU/vDtAV7VGkdkCLh
V/ZIXYEQCfWRN+6DhOnKd6eNjn7gmsae2GCPfoiDDm2
AsluhL9jkncm/XTo+8ATDe8sy4BjqcAXGE68jP1YT6q
4KOhoN0LjFmgk136PzcOr62YTiwedPLtVr62FSP2Tjc
```

c)Go to cmd and type the command "aws configure" and enter the details from the aws cli details above



```
C:\Users\ganes>aws configure
AWS Access Key ID [****************LK73]:
AWS Secret Access Key [****************mXc7]:
Default region name [us-west-2]:
Default output format [json]:
```

d)As we know this is session details which is temporary we also have to enter the additional command "aws configure set aws_session_token <Your_Session_Token>"



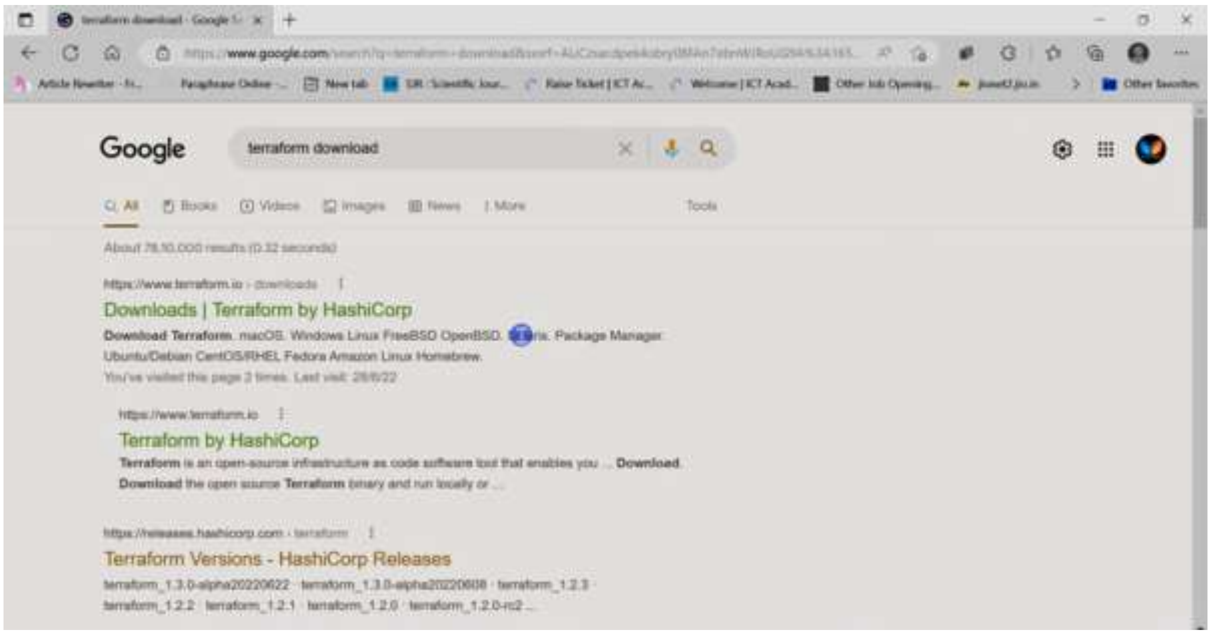
```
C:\Users\ganes>aws configure set aws_session_token IQoJb3JpZ2luX2VjEO3//////////wEaCXVzLXdlc3QtMiJIMEYCIQCyGhckwi0r0GeJT
7gKGWeq0C83IBa6W1ZcihnPpGTa9QIhAJ+Jz56NZv0JHpsoHxiXqMBZ+n5GVngZ/jNM613ar4ZRKrACCFYQAhoMMzM2MzI2MTkwNDUyIgxVZLlD9MD5r620G
5EqjQJ0R8LLIn95JZrhy+Gq+nnTeLlrh8qJ2dkCIAiMIdz5JfAA/bTQED6Rj+03imJpA4UlUFh8sP/5EEvV5dg5Qhw/dt3fPd6xsY7IsdjzFYmkaViESdJ9Z
AFIi8FXIpmqOMn81PjXe49DkF9cz8wWSdmmFgMGnJ8ZoGzIJYMgrf7vkMFFIbUIGl00mb8mB9apbEmm9yWLgnpplTREPoF7wK6A31zwWqhXx3p6uZgsSs3Xl
B64GsOnQnC5UoEFPZT4CHeabULXyqTOlgNk9yUD+hkAb+UU/vDtAV7VGkdkCLhV/ZIXYEQCfWRN+6DhOnKd6eNjn7gmsae2GCPfoiDDm2AsluhL9jkncm/XT
o+8ATDe8sy4BjqcAXGE68jP1YT6q4KOhoN0LjFmgk136PzcOr62YTiwedPLtVr62FSP2TjcT0NL80cEEhOIWDAi4Q/z/qkKN3TVFl0MpBSRogAH7rDpKwm+0
MPcHyDsraJ/X//IH83mOhc1MpAdZNNYRF8K7eO+QOvWbByl7ZhoY9Wu03RLIhZk1mJxiSGIq99yXCIGSRtPFmujjrkb59qW87Cl8ugZXg==
```

Cli will finaly get configured to our aws academy account

And you can confirm that by using command "aws ec2 list-instances"

Step 2: **Install and configure Terraform**

Download the zip file of the terraform for the windows from the official website of the hashicorp



terraform download

About 78,10,000 results (0.32 seconds)

https://www.terraform.io › downloads

Downloads | Terraform by HashiCorp

Download Terraform. macOS. Windows Linux FreeBSD OpenBSD. Solaris. Package Manager. Ubuntu/Debian CentOS/RHEL Fedora Amazon Linux Homebrew.

You've visited this page 2 times. Last visit: 28/6/22

https://www.terraform.io

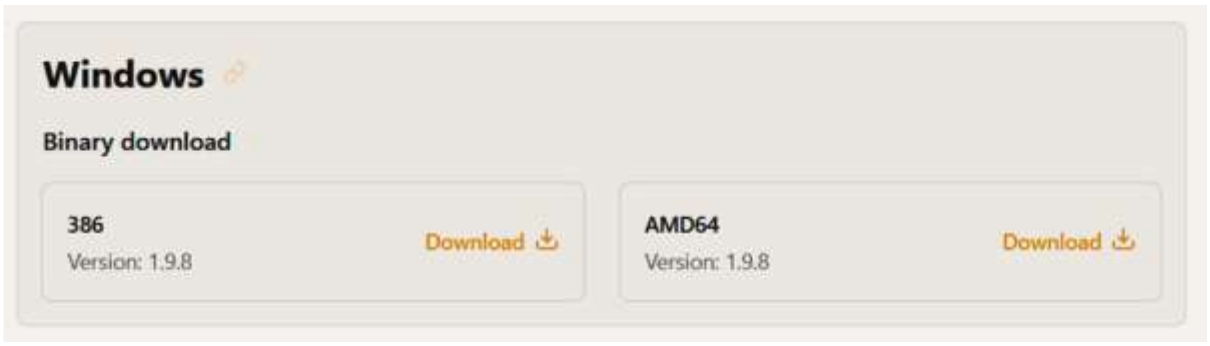
Terraform by HashiCorp

Terraform is an open-source infrastructure as code software tool that enables you ... **Download**. **Download** the open source **Terraform** binary and run locally or ...

https://releases.hashicorp.com › terraform

Terraform Versions - HashiCorp Releases

terraform_1.3.0-alpha20220622 · terraform_1.3.0-alpha20220608 · terraform_1.2.3 · terraform_1.2.2 · terraform_1.2.1 · terraform_1.2.0 · terraform_1.2.0-rc2 ...



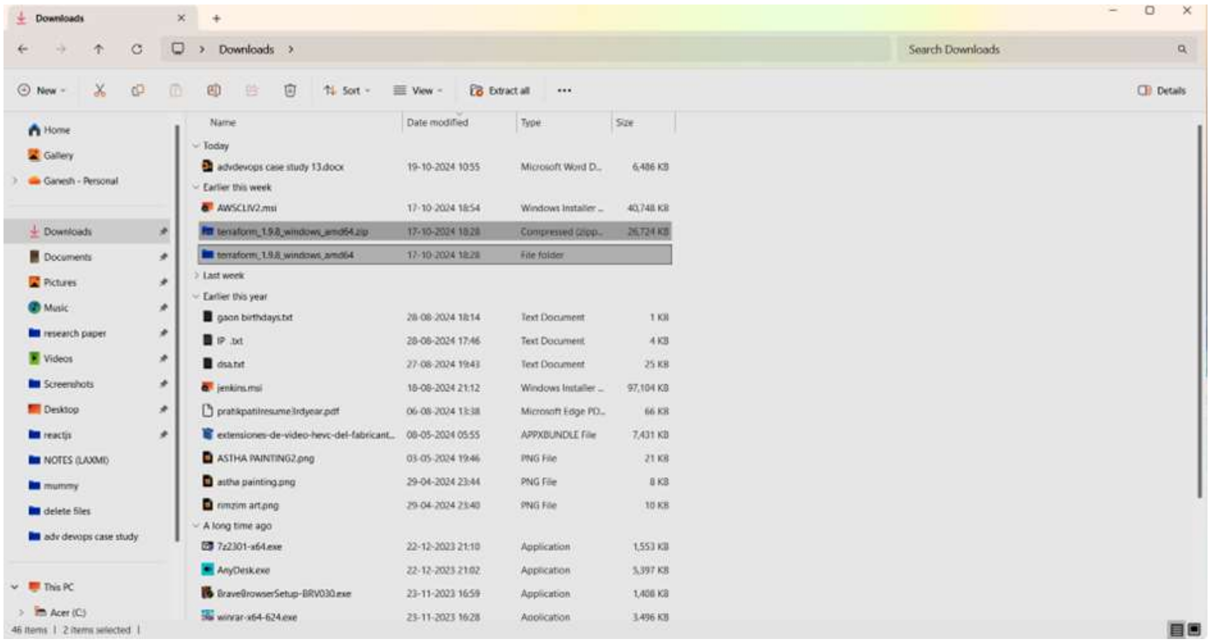
Windows

Binary download

386 Version: 1.9.8	AMD64 Version: 1.9.8

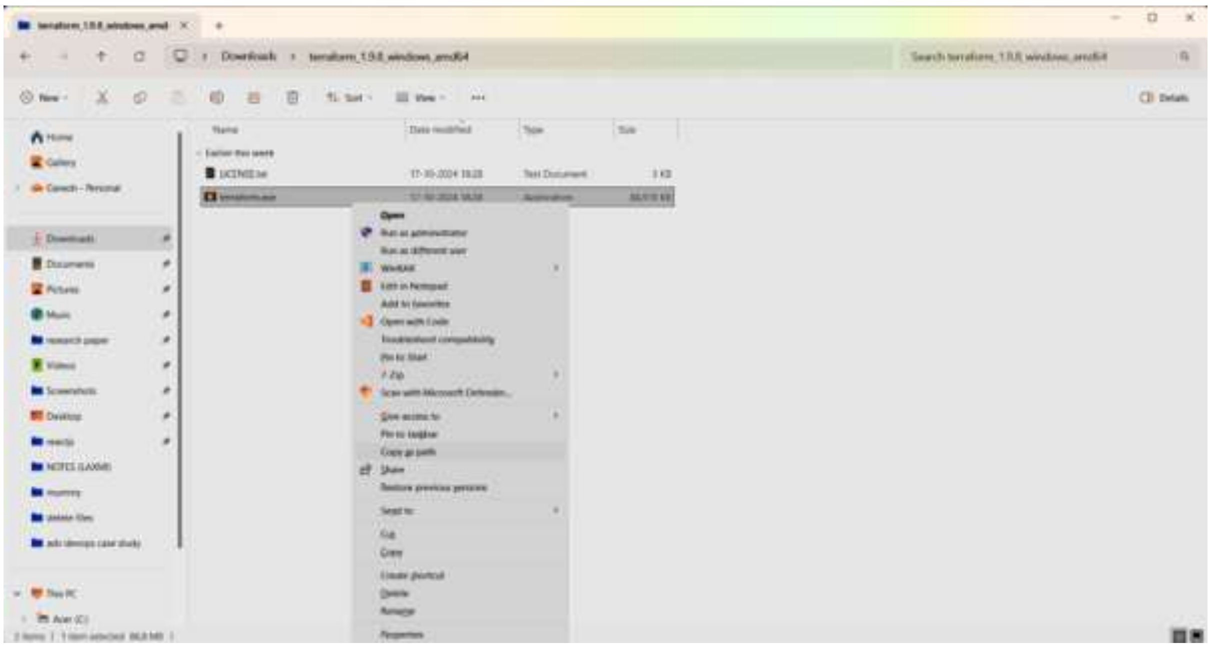
in my case I have downloaded the amd64 you can download any on them whichever supports your machine

after downloading it you have to **unzip** it

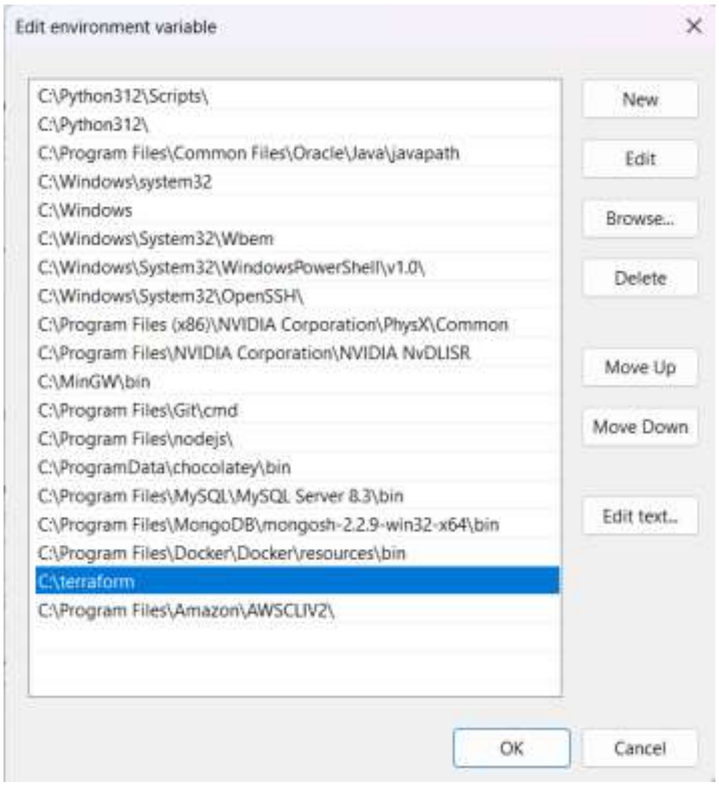


Name	Date modified	Type	Size
Today			
advdevops case study 13.docx	19-10-2024 10:55	Microsoft Word D...	6,486 KB
Earlier this week			
AWSCLIV2.msi	17-10-2024 18:54	Windows Installer ...	40,748 KB
terraform_1.9.8_windows_amd64.zip	17-10-2024 18:28	Compressed (zipp...	26,724 KB
terraform_1.9.8_windows_amd64	17-10-2024 18:28	File folder	
Last week			
Earlier this year			
gaon birthdays.txt	28-08-2024 18:14	Text Document	1 KB
IP .txt	28-08-2024 17:46	Text Document	4 KB
dsa.txt	27-08-2024 19:43	Text Document	25 KB
jenkins.msi	18-08-2024 21:12	Windows Installer ...	97,104 KB
pratikpatilresume3rdyear.pdf	06-08-2024 13:38	Microsoft Edge PD...	66 KB
extensiones-de-video-hevc-del-fabricant...	08-05-2024 05:55	APPXBUNDLE File	7,431 KB
ASTHA PAINTING2.png	03-05-2024 19:46	PNG File	21 KB
astha painting.png	29-04-2024 23:44	PNG File	8 KB
rimzim art.png	29-04-2024 23:40	PNG File	10 KB
A long time ago			
7z2301-x64.exe	22-12-2023 21:10	Application	1,553 KB
AnyDesk.exe	22-12-2023 21:02	Application	5,397 KB
BraveBrowserSetup-BRV030.exe	23-11-2023 16:59	Application	1,408 KB
winrar-x64-624.exe	23-11-2023 16:28	Application	3,496 KB

Navigate inside of the unzip file and **copy it's path**



And add that path to environment variables



Edit environment variable

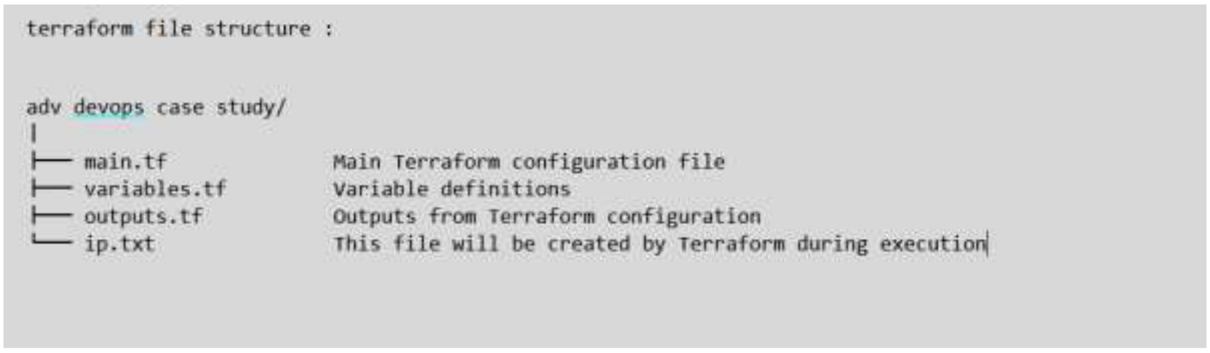
```
C:\Python312\Scripts\
C:\Python312\
C:\Program Files\Common Files\Oracle\Java\javapath
C:\Windows\system32
C:\Windows
C:\Windows\System32\Wbem
C:\Windows\System32\WindowsPowerShell\v1.0\
C:\Windows\System32\OpenSSH\
C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common
C:\Program Files\NVIDIA Corporation\NVIDIA NvDLISR
C:\MinGW\bin
C:\Program Files\Git\cmd
C:\Program Files\nodejs\
C:\ProgramData\chocolatey\bin
C:\Program Files\MySQL\MySQL Server 8.3\bin
C:\Program Files\MongoDB\mongosh-2.2.9-win32-x64\bin
C:\Program Files\Docker\Docker\resources\bin
C:\terraform
C:\Program Files\Amazon\AWSCLIV2\
```

Check the terraform through cmd using **terraform –version**

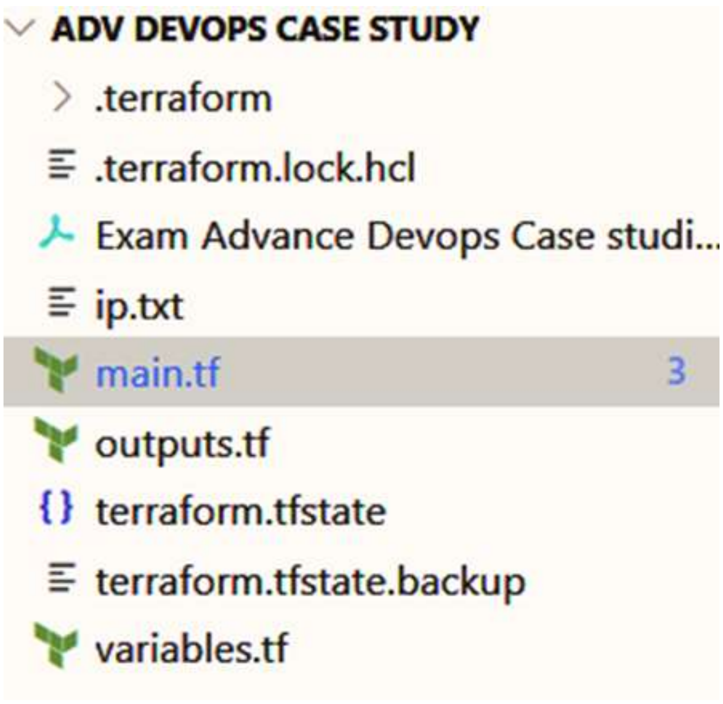
```
C:\Users\ganes>terraform --version
Terraform v1.9.8
on windows_amd64
```

Step 3: **write the terraform code**

make the following directory structure in that new working folder for ex: adv devops case study

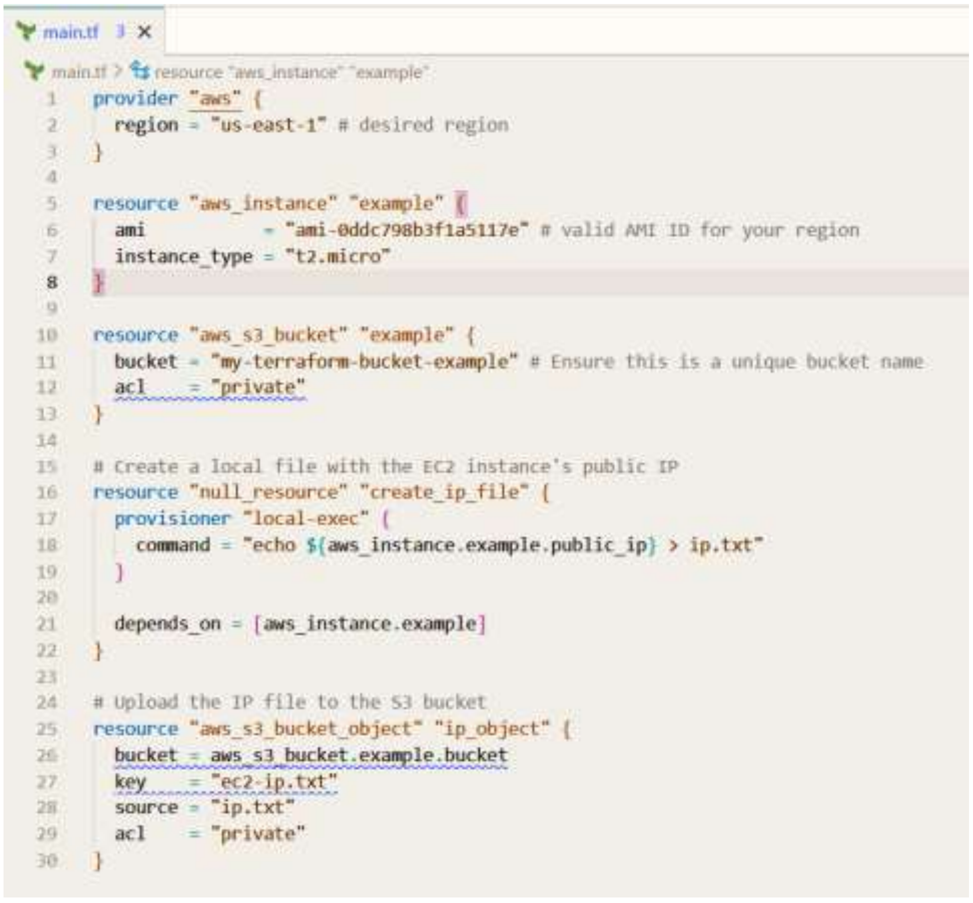


```
terraform file structure :


adv devops case study/
|
├── main.tf              Main Terraform configuration file
├── variables.tf         Variable definitions
├── outputs.tf           Outputs from Terraform configuration
└── ip.txt               This file will be created by Terraform during execution
```



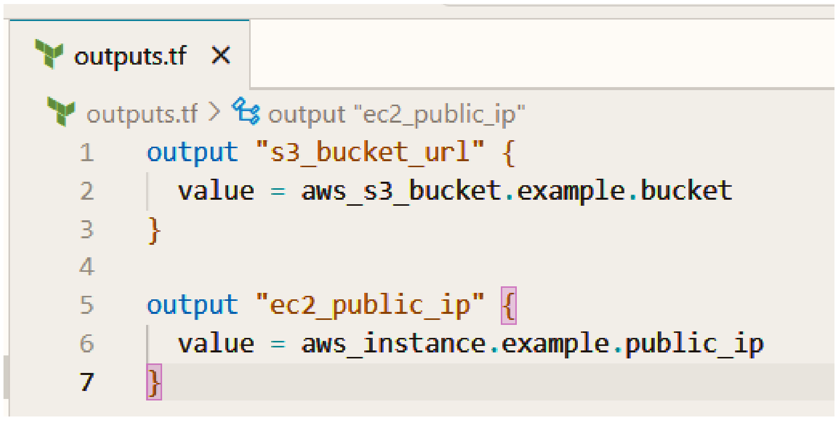
```
ADV DEVOPS CASE STUDY
  .terraform
  .terraform.lock.hcl
  Exam Advance Devops Case studi...
  ip.txt
  main.tf                3
  outputs.tf
  terraform.tfstate
  terraform.tfstate.backup
  variables.tf
```

main.tf terraform code :



```
provider "aws" {
  region = "us-east-1" # desired region
}

resource "aws_instance" "example" {
  ami           = "ami-0ddc798b3f1a5117e" # valid AMI ID for your region
  instance_type = "t2.micro"
}

resource "aws_s3_bucket" "example" {
  bucket = "my-terraform-bucket-example" # Ensure this is a unique bucket name
  acl    = "private"
}

# Create a local file with the EC2 instance's public IP
resource "null_resource" "create_ip_file" {
  provisioner "local-exec" {
    command = "echo ${aws_instance.example.public_ip} > ip.txt"
  }

  depends_on = [aws_instance.example]
}

# Upload the IP file to the S3 bucket
resource "aws_s3_bucket_object" "ip_object" {
  bucket = aws_s3_bucket.example.bucket
  key    = "ec2-ip.txt"
  source = "ip.txt"
  acl    = "private"
}
```

output.tf code :

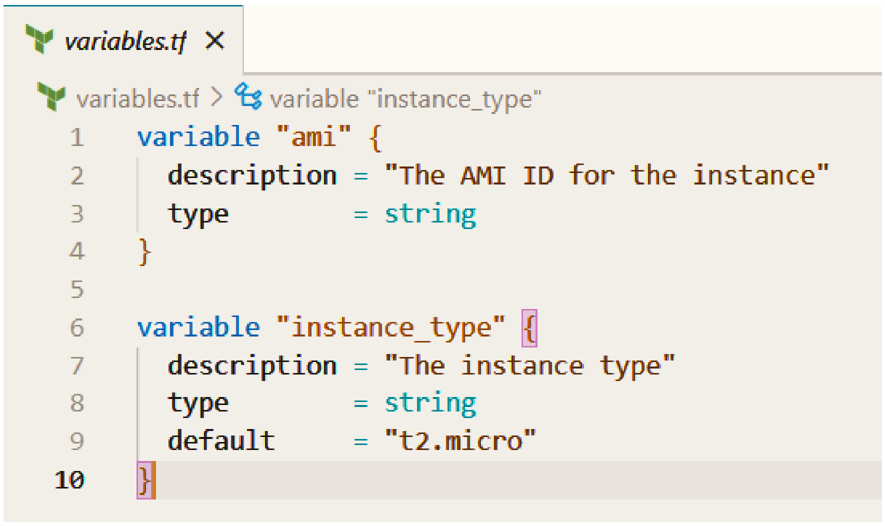


outputs.tf

outputs.tf > output "ec2_public_ip"

```
output "s3_bucket_url" {
  value = aws_s3_bucket.example.bucket
}

output "ec2_public_ip" {
  value = aws_instance.example.public_ip
}
```

variables.tf



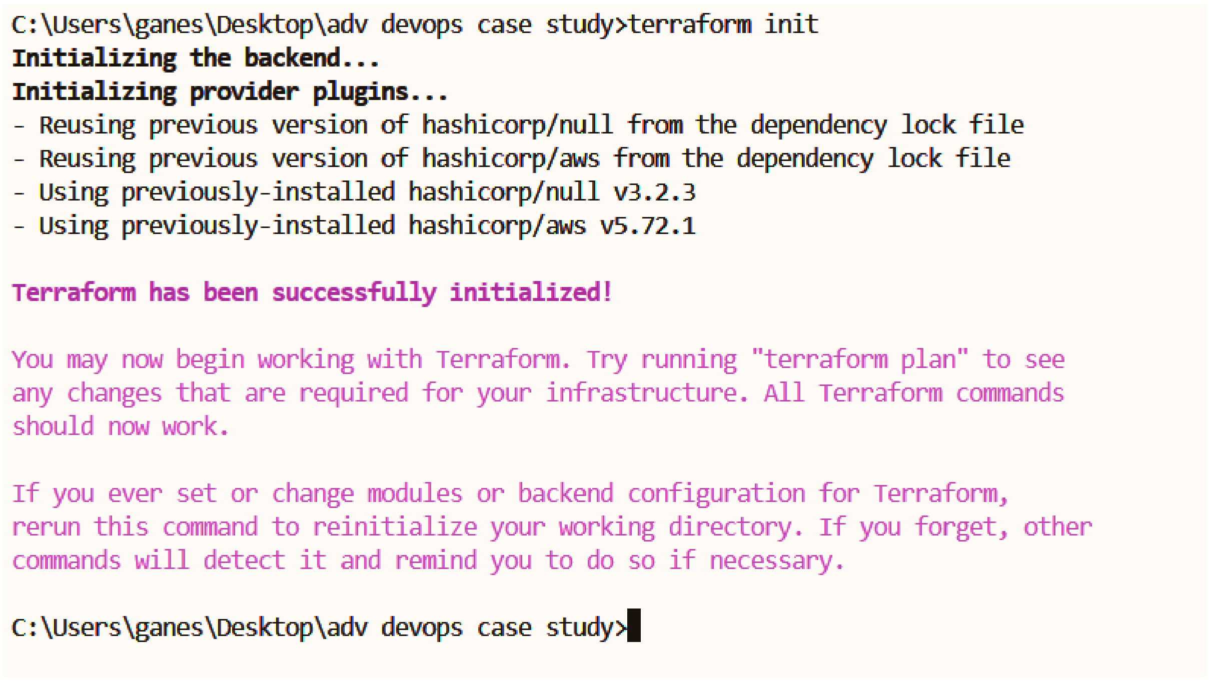
variables.tf

variables.tf > variable "instance_type"

```
variable "ami" {
  description = "The AMI ID for the instance"
  type        = string
}

variable "instance_type" {
  description = "The instance type"
  type        = string
  default     = "t2.micro"
}
```

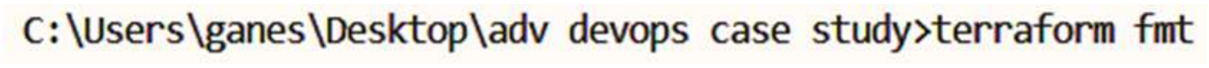
Step 4: **run terraform commands**

a) “terraform init” command



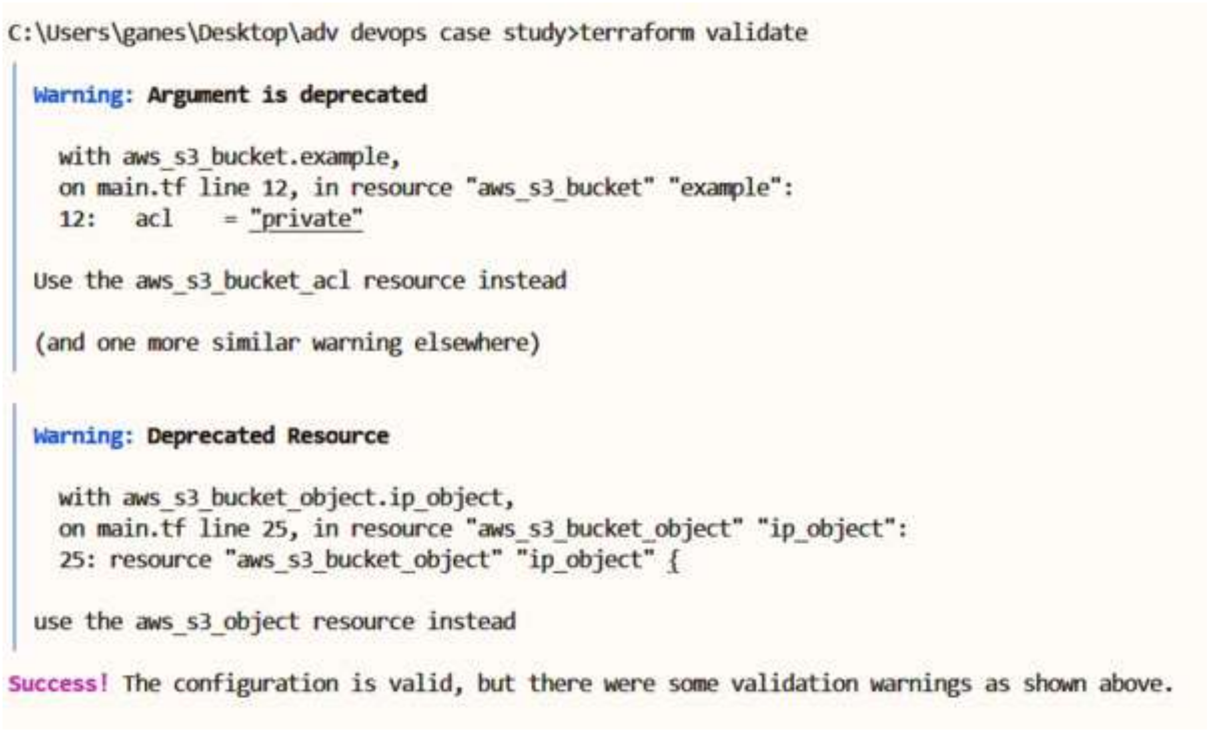
```
C:\Users\ganes\Desktop\adv devops case study>terraform init
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/null from the dependency lock file
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/null v3.2.3
- Using previously-installed hashicorp/aws v5.72.1

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.

C:\Users\ganes\Desktop\adv devops case study>
```

b) ” terraform fmt” command:



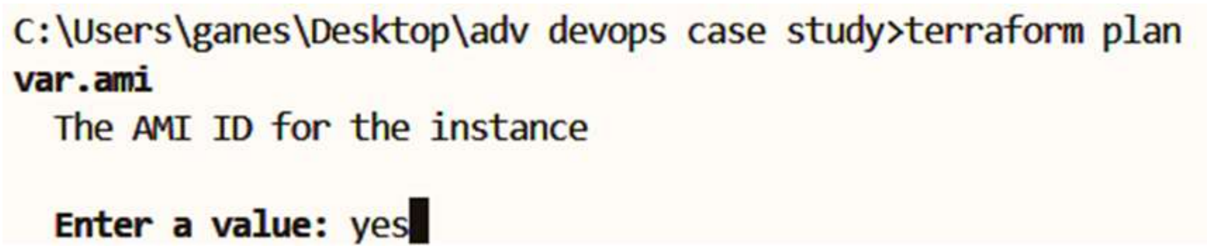
```
C:\Users\ganes\Desktop\adv devops case study>terraform fmt
```

c) “terraform validate” command:



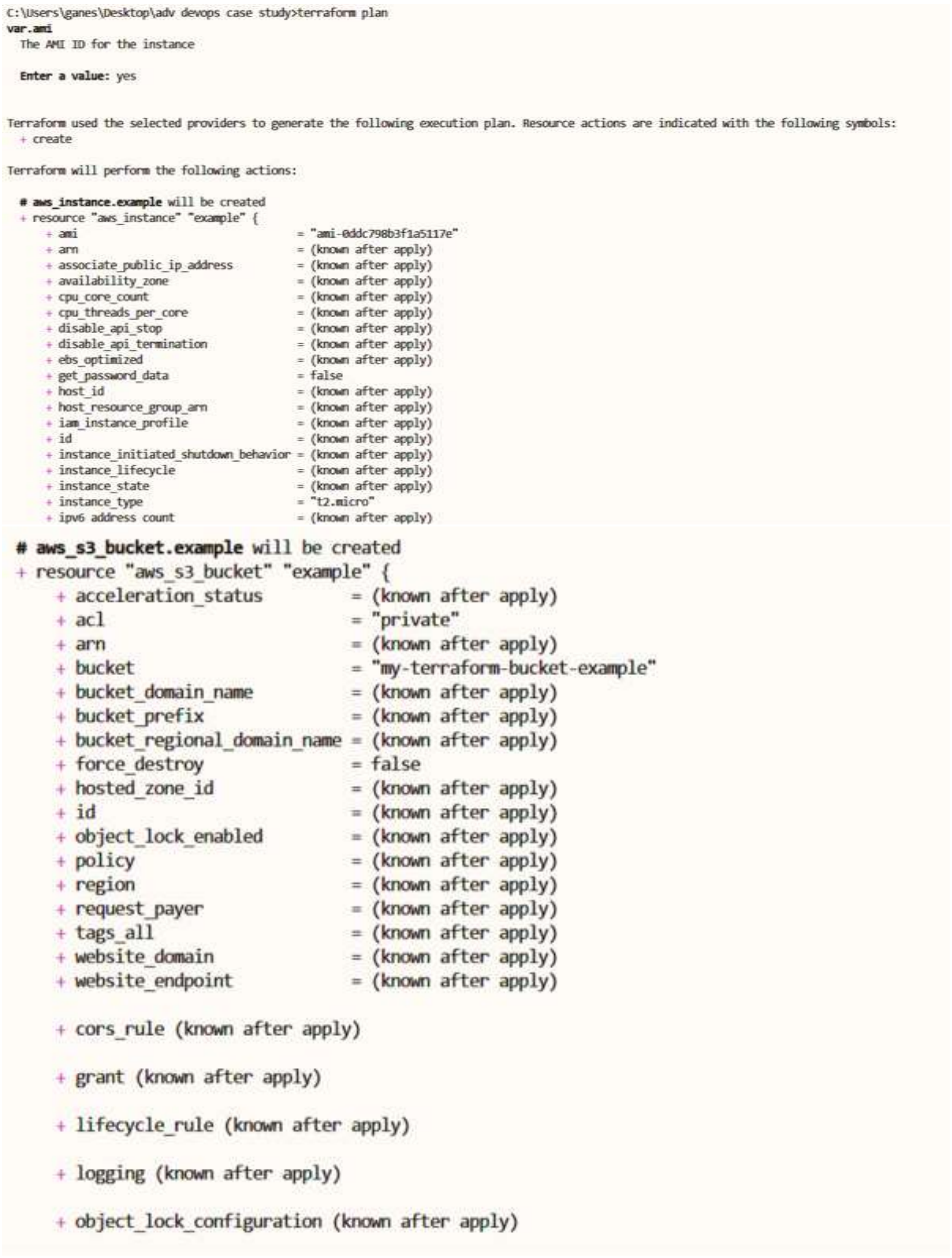
```
C:\Users\ganes\Desktop\adv devops case study>terraform validate

│ Warning: Argument is deprecated
│
│   with aws_s3_bucket.example,
│   on main.tf line 12, in resource "aws_s3_bucket" "example":
│   12:    acl      = "private"
│
│ Use the aws_s3_bucket_acl resource instead
│
│ (and one more similar warning elsewhere)

│ Warning: Deprecated Resource
│
│   with aws_s3_bucket_object.ip_object,
│   on main.tf line 25, in resource "aws_s3_bucket_object" "ip_object":
│   25: resource "aws_s3_bucket_object" "ip_object" {
│
│ use the aws_s3_object resource instead

Success! The configuration is valid, but there were some validation warnings as shown above.
```

d) "terraform plan" command:



```
C:\Users\ganes\Desktop\adv devops case study>terraform plan
var.ami
  The AMI ID for the instance

  Enter a value: yes
```

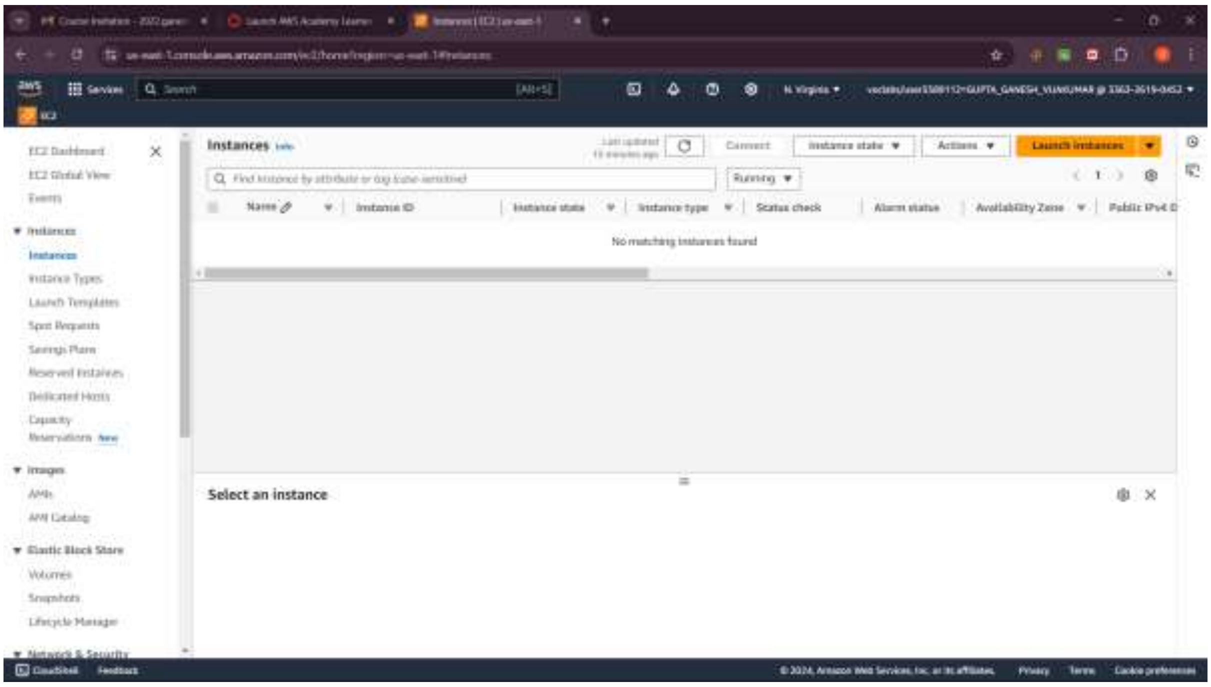
enter yes and enter to see the plan that is going to execute on aws



```
C:\Users\ganes\Desktop\adv devops case study>terraform plan
var.ami
  The AMI ID for the instance

  Enter a value: yes


Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.example will be created
  + resource "aws_instance" "example" {
      + ami                                  = "ami-0ddc798b3f1a5117e"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_stop                     = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + host_resource_group_arn              = (known after apply)
      + iam_instance_profile                 = (known after apply)
      + id                                   = (known after apply)
      + instance_initiated_shutdown_behavior = (known after apply)
      + instance_lifecycle                   = (known after apply)
      + instance_state                       = (known after apply)
      + instance_type                        = "t2.micro"
      + ipv6_address_count                   = (known after apply)
```

```
  # aws_s3_bucket.example will be created
  + resource "aws_s3_bucket" "example" {
      + acceleration_status         = (known after apply)
      + acl                         = "private"
      + arn                         = (known after apply)
      + bucket                      = "my-terraform-bucket-example"
      + bucket_domain_name          = (known after apply)
      + bucket_prefix               = (known after apply)
      + bucket_regional_domain_name = (known after apply)
      + force_destroy               = false
      + hosted_zone_id              = (known after apply)
      + id                          = (known after apply)
      + object_lock_enabled         = (known after apply)
      + policy                      = (known after apply)
      + region                      = (known after apply)
      + request_payer               = (known after apply)
      + tags_all                    = (known after apply)
      + website_domain              = (known after apply)
      + website_endpoint            = (known after apply)

      + cors_rule (known after apply)

      + grant (known after apply)

      + lifecycle_rule (known after apply)

      + logging (known after apply)

      + object_lock_configuration (known after apply)
```

Before executing "terraform apply" command there is no ec2 instance running

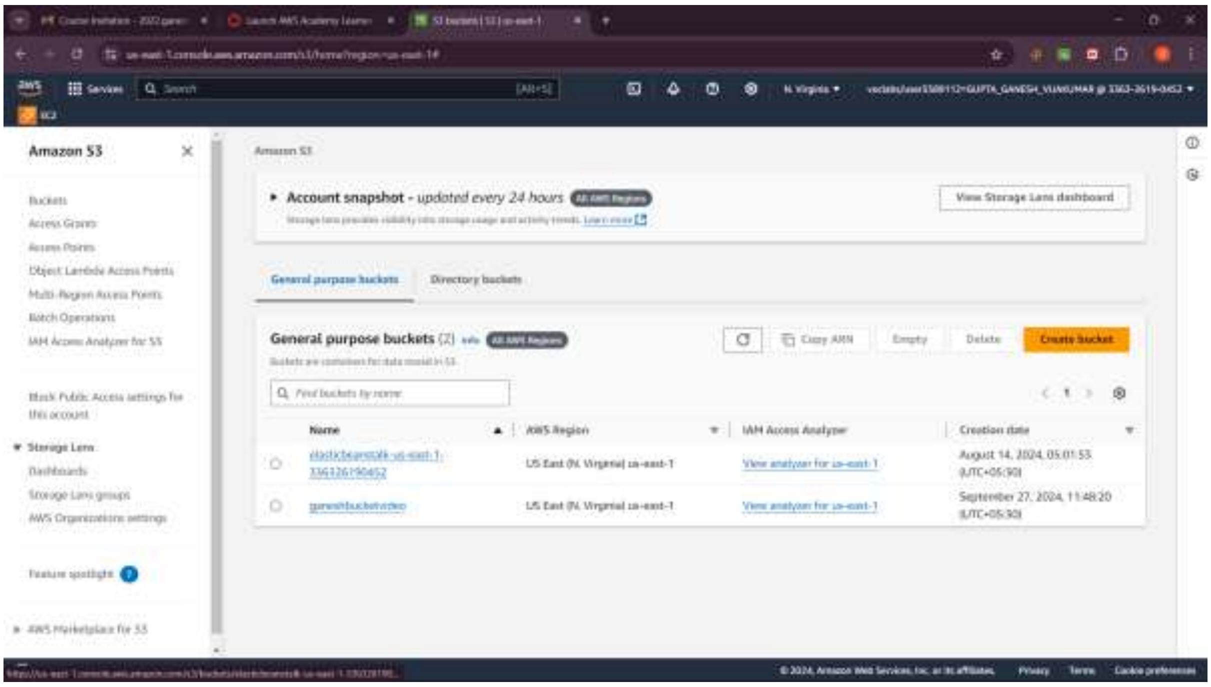


Instances

No matching instances found

Select an instance

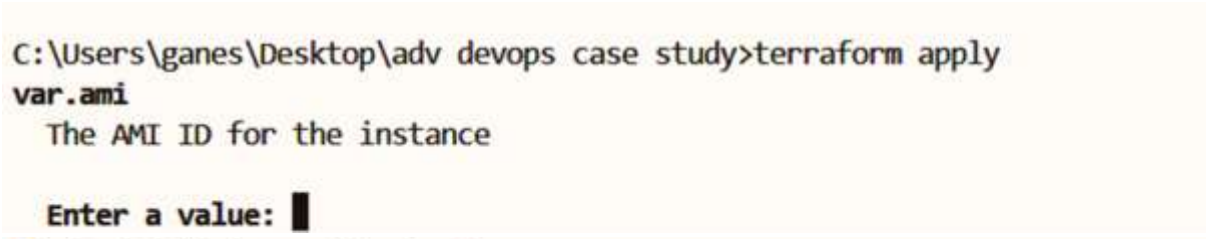
Before executing "terraform apply" command there are only 2 s3 bucket instances and note that every s3 bucket has unique name



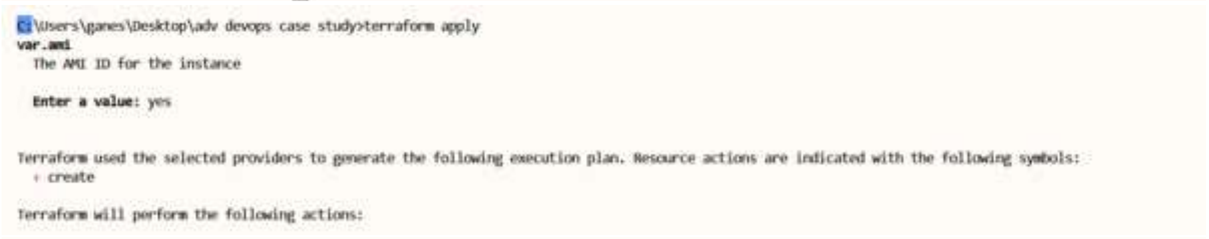
General purpose buckets (2)

	Name	AWS Region	IAM Access Analyzer	Creation date
○	elasticbeanstalk-us-east-1-336126190452	US East (N. Virginia) us-east-1	View analyzer for us-east-1	August 14, 2024, 05:01:53 (UTC+05:30)
○	ganeshbucketvideo	US East (N. Virginia) us-east-1	View analyzer for us-east-1	September 27, 2024, 11:48:20 (UTC+05:30)

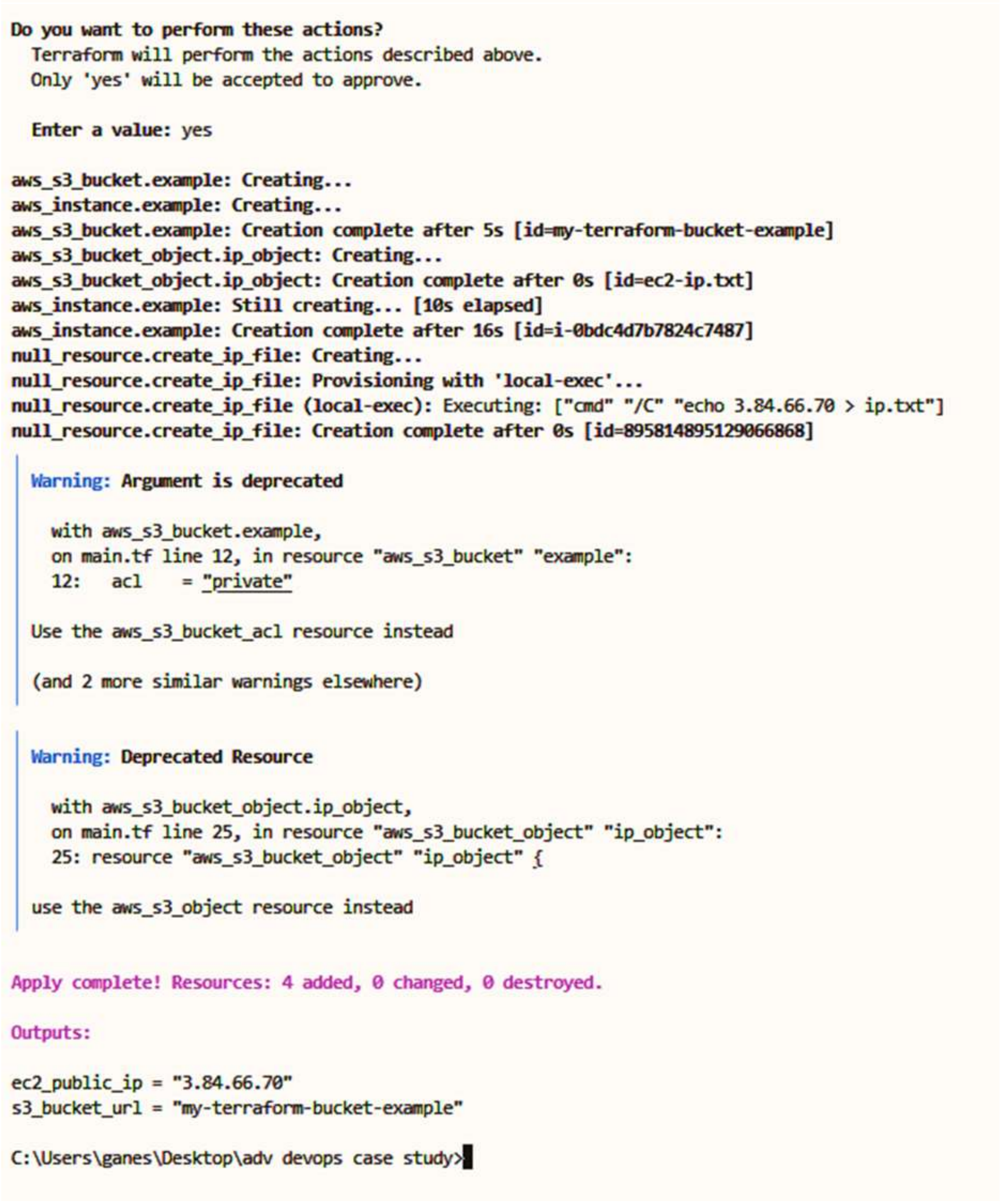
e) Executing "terraform apply" command



```
C:\Users\ganes\Desktop\adv devops case study>terraform apply
var.ami
  The AMI ID for the instance

  Enter a value: 
```



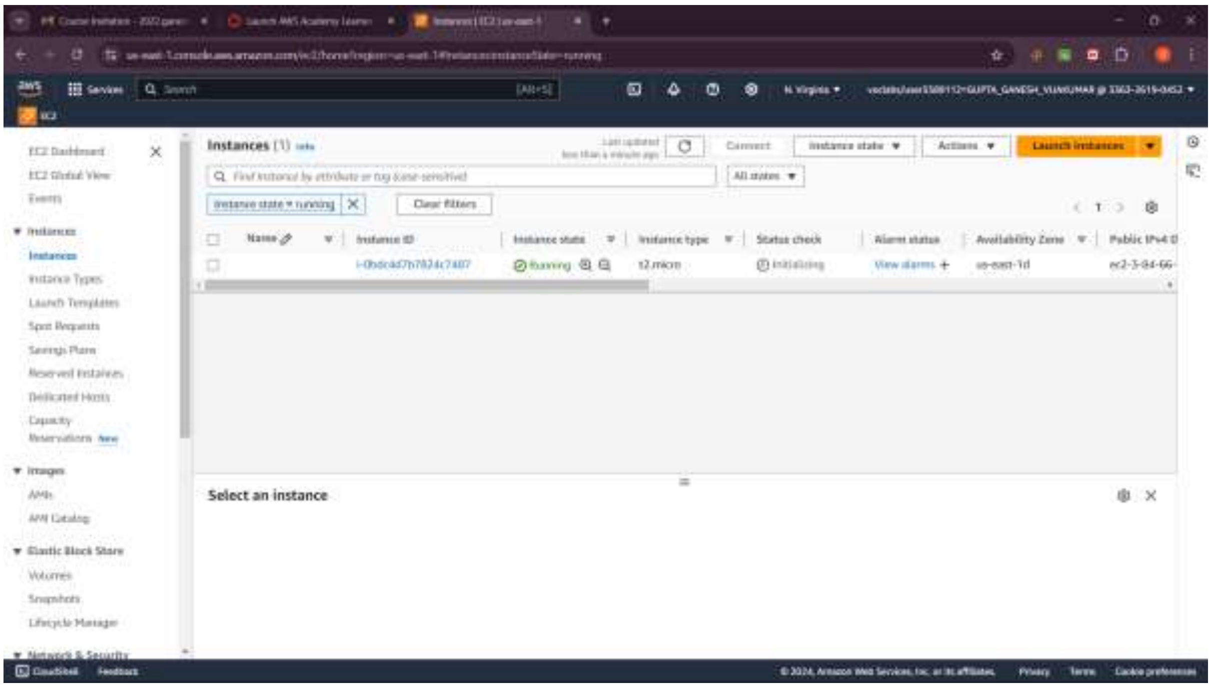
```
C:\Users\ganes\Desktop\adv devops case study>terraform apply
var.ami
  The AMI ID for the instance

  Enter a value: yes


Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:
```



```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_s3_bucket.example: Creating...
aws_instance.example: Creating...
aws_s3_bucket.example: Creation complete after 5s [id=my-terraform-bucket-example]
aws_s3_bucket_object.ip_object: Creating...
aws_s3_bucket_object.ip_object: Creation complete after 0s [id=ec2-ip.txt]
aws_instance.example: Still creating... [10s elapsed]
aws_instance.example: Creation complete after 16s [id=i-0bdc4d7b7824c7487]
null_resource.create_ip_file: Creating...
null_resource.create_ip_file: Provisioning with 'local-exec'...
null_resource.create_ip_file (local-exec): Executing: ["cmd" "/C" "echo 3.84.66.70 > ip.txt"]
null_resource.create_ip_file: Creation complete after 0s [id=895814895129066868]

│ Warning: Argument is deprecated
│
│   with aws_s3_bucket.example,
│   on main.tf line 12, in resource "aws_s3_bucket" "example":
│   12:   acl    = "private"
│
│ Use the aws_s3_bucket_acl resource instead
│
│ (and 2 more similar warnings elsewhere)


│ Warning: Deprecated Resource
│
│   with aws_s3_bucket_object.ip_object,
│   on main.tf line 25, in resource "aws_s3_bucket_object" "ip_object":
│   25: resource "aws_s3_bucket_object" "ip_object" {
│
│ use the aws_s3_object resource instead


Apply complete! Resources: 4 added, 0 changed, 0 destroyed.

Outputs:

ec2_public_ip = "3.84.66.70"
s3_bucket_url = "my-terraform-bucket-example"

C:\Users\ganes\Desktop\adv devops case study>
```

Now check the aws ec2 instances and s3 bucket at academy

1 ec2 instance is created

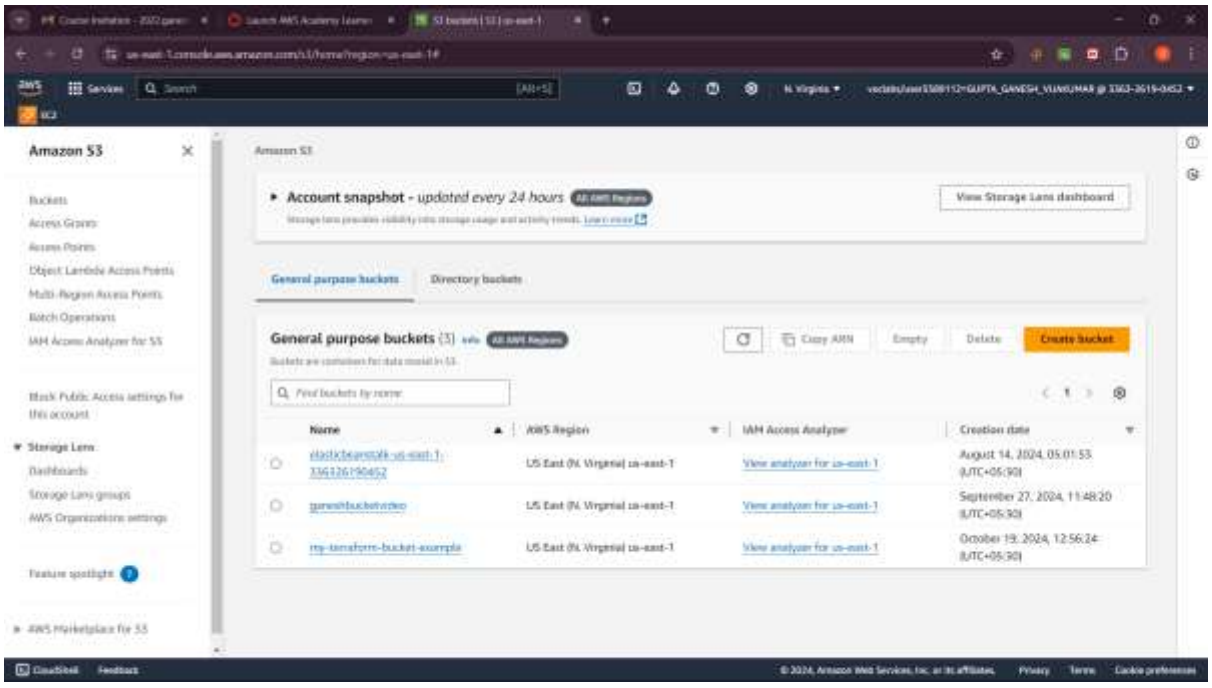


Instances (1)

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 D
	i-0bdcdd7b7824c7407	Running	t2.micro	Initializing	View alarms +	us-east-1d	ec2-3-84-66-

Select an instance

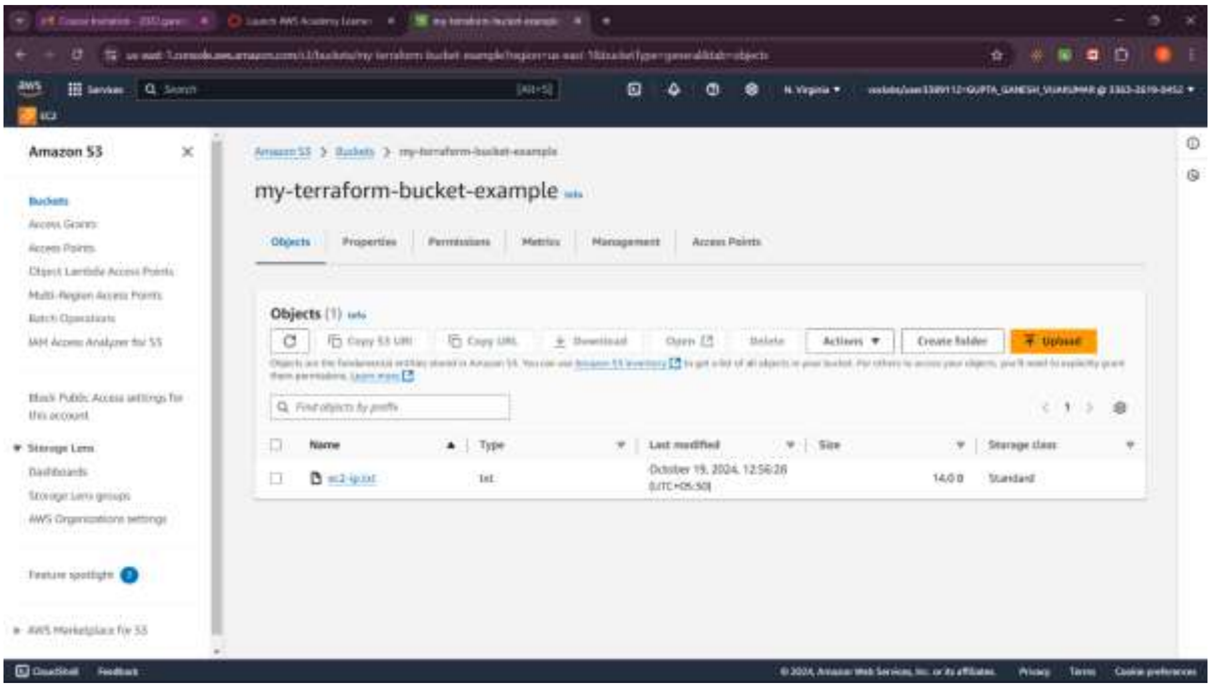
1 s3 bucket is also created



General purpose buckets (3)

Name	AWS Region	IAM Access Analyzer	Creation date
elasticbeanstalk-us-east-1-336126190452	US East (N. Virginia) us-east-1	View analyzer for us-east-1	August 14, 2024, 05:01:53 (UTC+05:30)
ganeshbucketvideo	US East (N. Virginia) us-east-1	View analyzer for us-east-1	September 27, 2024, 11:48:20 (UTC+05:30)
my-terraform-bucket-example	US East (N. Virginia) us-east-1	View analyzer for us-east-1	October 19, 2024, 12:56:24 (UTC+05:30)

Now click on s3 bucket that is newly created

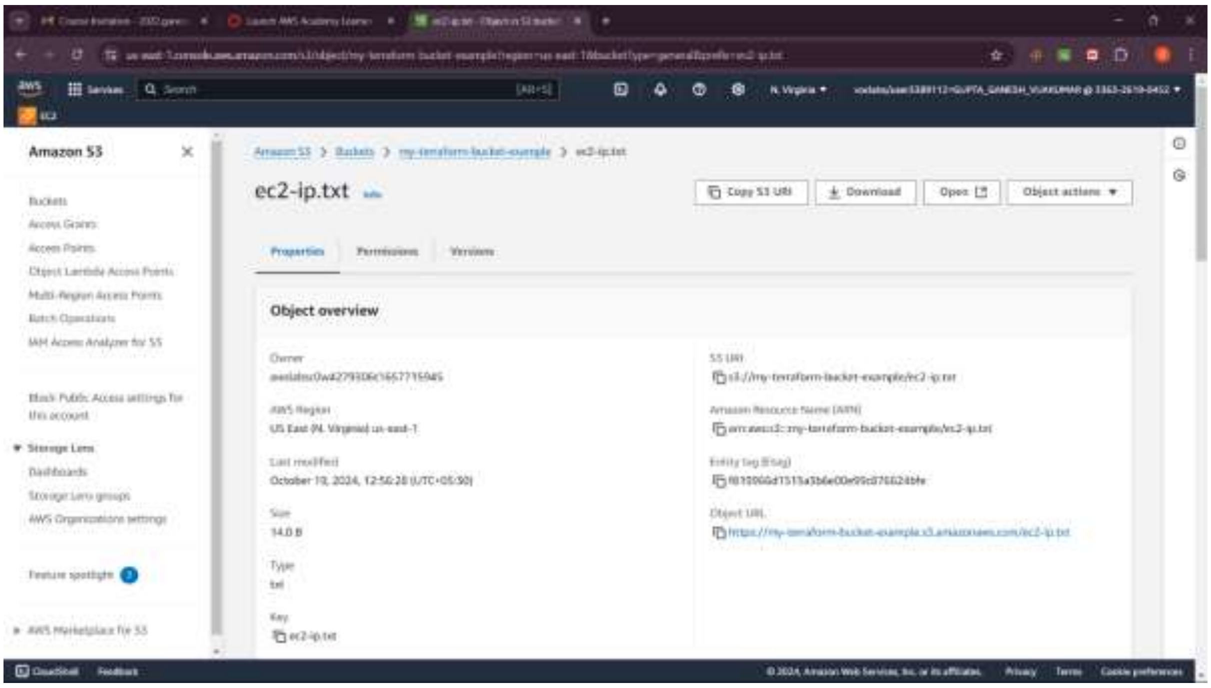

Amazon S3 > Buckets > my-terraform-bucket-example

my-terraform-bucket-example

Objects (1)

	Name	Type	Last modified	Size	Storage class
☐	ec2-ip.txt	txt	October 19, 2024, 12:56:26 (UTC+05:30)	14.0 B	Standard

Now click on ec2-ip.txt


Amazon S3 > Buckets > my-terraform-bucket-example > ec2-ip.txt

ec2-ip.txt

Object overview

Owner
awslabsc0w4279306c1657715945

AWS Region
US East (N. Virginia) us-east-1

Last modified
October 19, 2024, 12:56:28 (UTC+05:30)

Size
14.0 B

Type
txt

Key
ec2-ip.txt

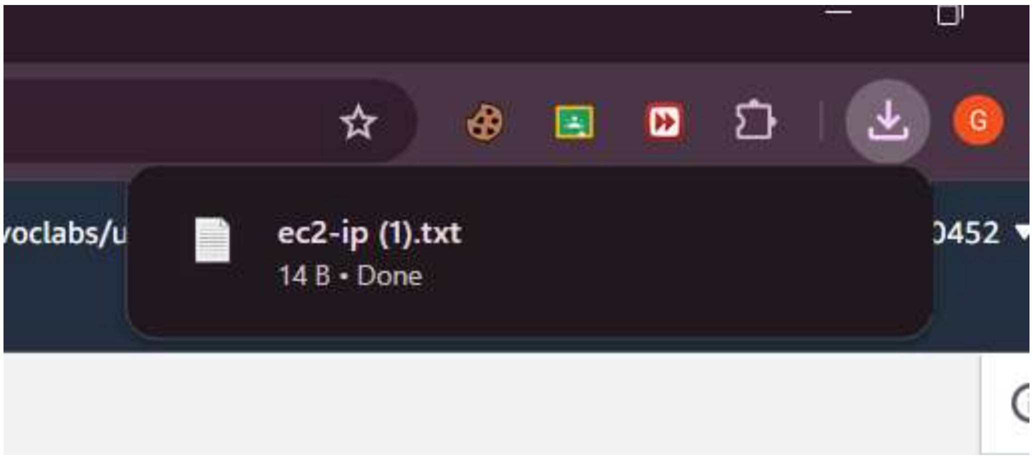
S3 URI
s3://my-terraform-bucket-example/ec2-ip.txt

Amazon Resource Name (ARN)
arn:aws:s3:::my-terraform-bucket-example/ec2-ip.txt

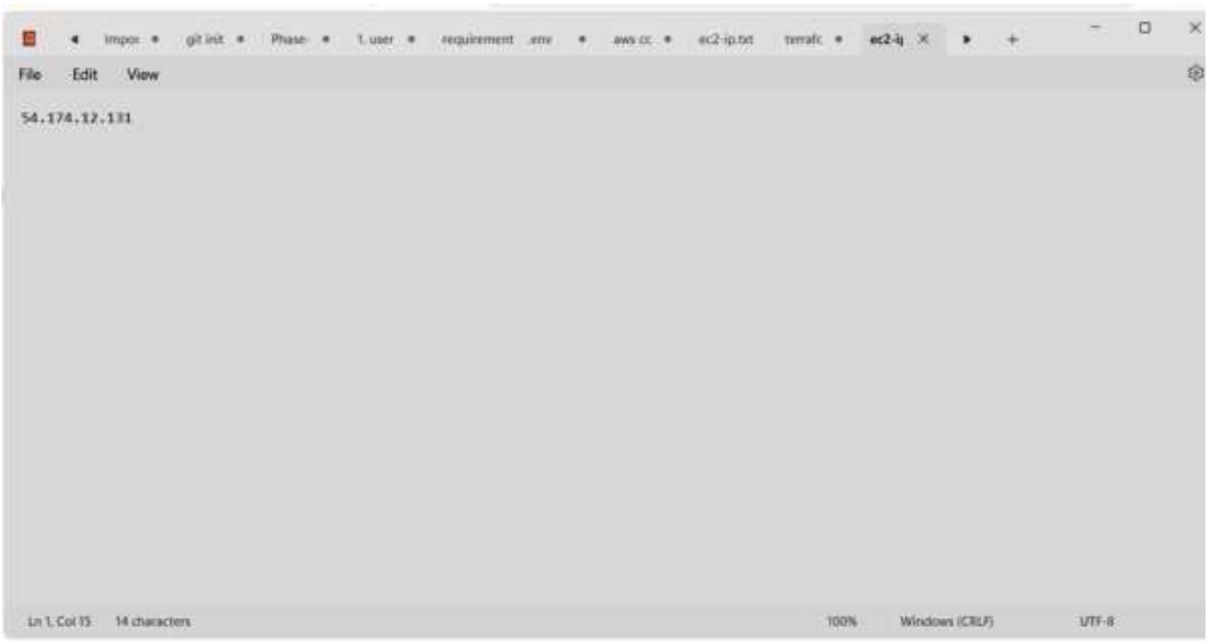
Entity tag (Etag)
f810060d1515a3b6e00e99cd76624bfe

Object URL
https://my-terraform-bucket-example.s3.amazonaws.com/ec2-ip.txt

Click on download


ec2-ip (1).txt
14 B • Done

Open it

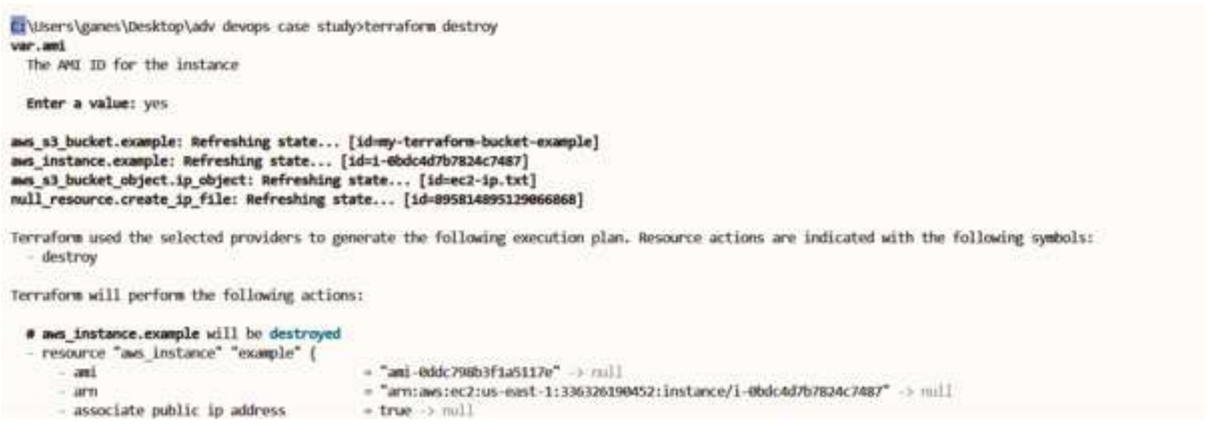


```
54.174.12.131
```

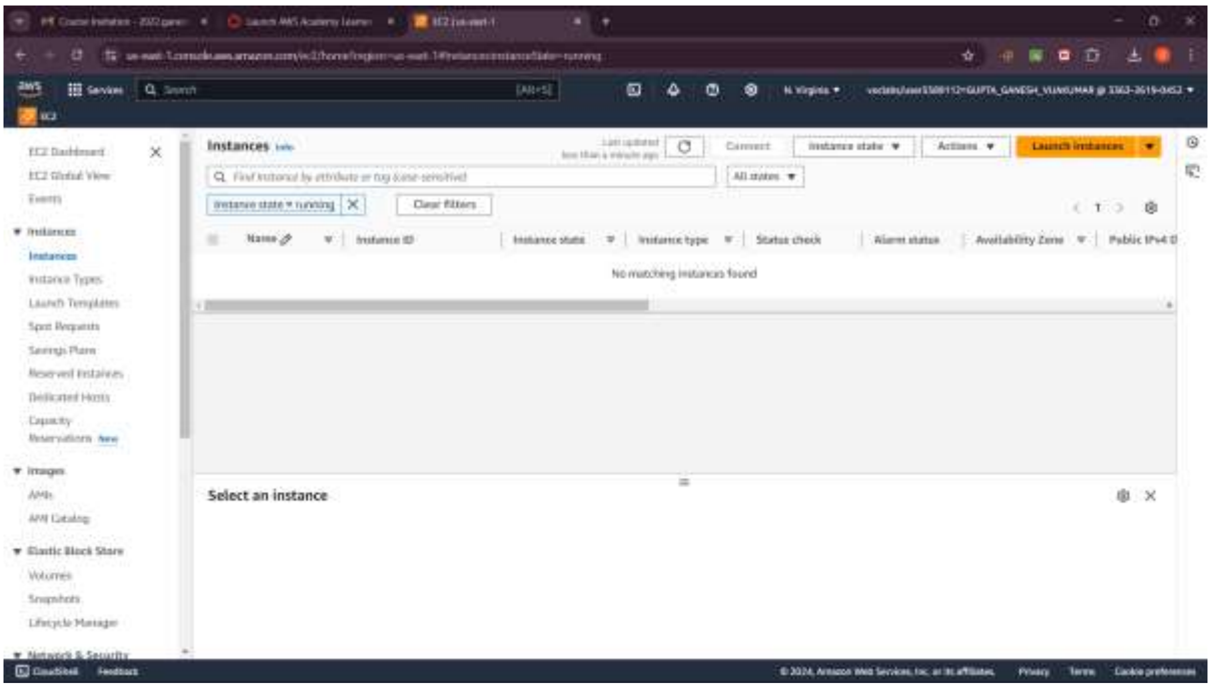
Ln 1, Col 15   14 characters   100%   Windows (CRLF)   UTF-8

The ip address is stored inside the s3 bucket is finally proved

And then use terraform destroy command to destroy all the instances that are made



```
C:\Users\ganes\Desktop\adv devops case study>terraform destroy
var.ami
  The AMI ID for the instance

  Enter a value: yes

aws_s3_bucket.example: Refreshing state... [id=my-terraform-bucket-example]
aws_instance.example: Refreshing state... [id=i-0bdc4d7b7824c7487]
aws_s3_bucket_object.ip_object: Refreshing state... [id=ec2-ip.txt]
null_resource.create_ip_file: Refreshing state... [id=8958148951290668068]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  - destroy

Terraform will perform the following actions:

  # aws_instance.example will be destroyed
  - resource "aws_instance" "example" {
      - ami                                  = "ami-0ddc798b3f1a5117e" -> null
      - arn                                  = "arn:aws:ec2:us-east-1:336326190452:instance/i-0bdc4d7b7824c7487" -> null
      - associate public ip address          = true -> null
```

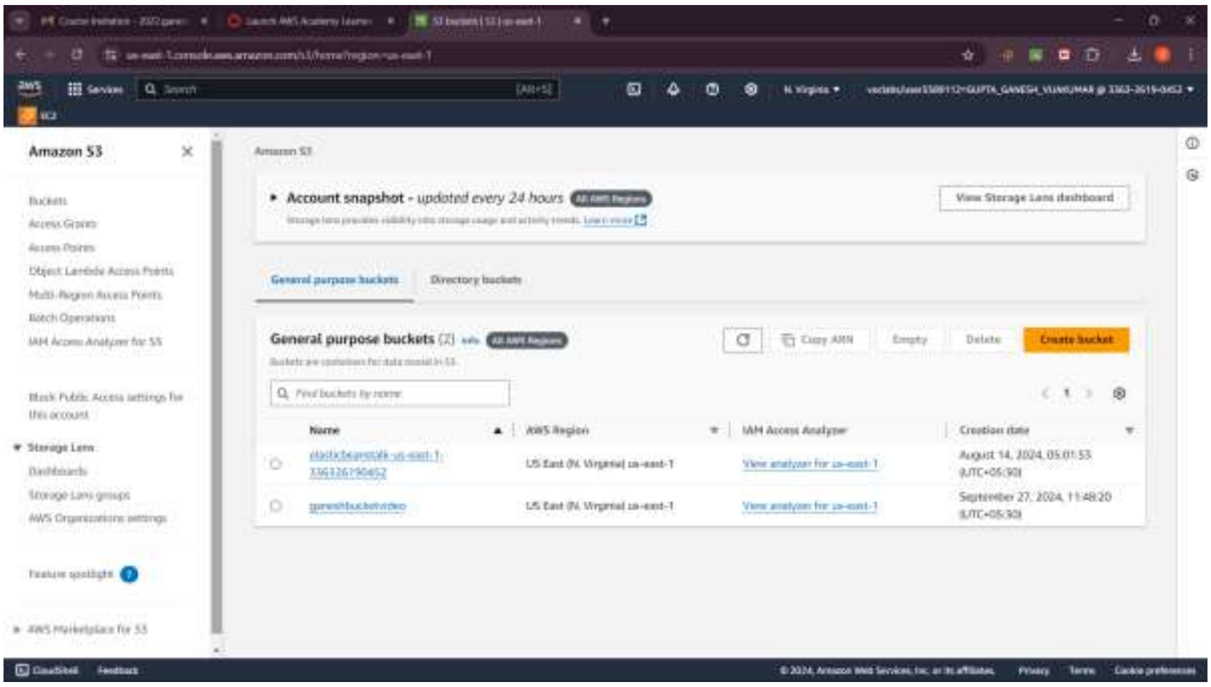


Instances

Instance state = running

No matching instances found

Select an instance



General purpose buckets (2) Info All AWS Regions

Buckets are containers for data stored in S3.

Name	AWS Region	IAM Access Analyzer	Creation date
elasticbeanstalk-us-east-1-336126190452	US East (N. Virginia) us-east-1	View analyzer for us-east-1	August 14, 2024, 05:01:53 (UTC+05:30)
ganeshbucketvideo	US East (N. Virginia) us-east-1	View analyzer for us-east-1	September 27, 2024, 11:48:20 (UTC+05:30)

EC2 instance and S3 bucket is destroyed.

Conclusion:

In this experiment, we used Terraform to automate the deployment of AWS infrastructure by creating an EC2 instance and an S3 bucket, and storing the instance's public IP address in the S3 bucket. We began by configuring AWS CLI with temporary credentials provided by AWS Academy, allowing us to authenticate and interact with AWS services. Using Terraform, we defined the necessary resources in the main.tf file, specifying the AWS provider and creating an EC2 instance with Amazon Linux 2. We also set up an S3 bucket to store a file containing the EC2 instance's public IP address. The Terraform script utilized outputs to extract the public IP and the local_file resource to save it locally, followed by uploading it to the S3 bucket. This experiment demonstrated how Terraform simplifies infrastructure management by automating the provisioning, configuration, and management of cloud resources, making it highly useful for maintaining scalable and repeatable infrastructure setups across multiple environments.