Lab 10 – GH CLI Codespaces + AWS + Terraform: CLI Automation of VPC/Subnet Creation

Name: Emaan Hanif

Reg no: BSE017

Task 1 — GitHub CLI Codespace Setup & Authentication

Goal

The purpose of this task is to install the GitHub CLI, authenticate it with the required permissions, and connect to a GitHub Codespace using SSH. This setup allows us to work inside a cloud-based development environment directly from our terminal.

1. Installing GitHub CLI

GitHub CLI (Command Line Interface) is required so we can interact with GitHub directly from the terminal.
Using GH CLI, we can authenticate, manage repositories, and connect to Codespaces.

Command Used:

winget install --id GitHub.cli

```
PS C:\Users\HP> winget install --id GitHub.cli
Found an existing package already installed. Trying to upgrade the installed packa
No available upgrade found.
No newer package versions are available from the configured sources.
```

After running this command, Winget downloads and installs the latest version of GitHub CLI on the system.

Screenshot Required:
task1_gh_install.png

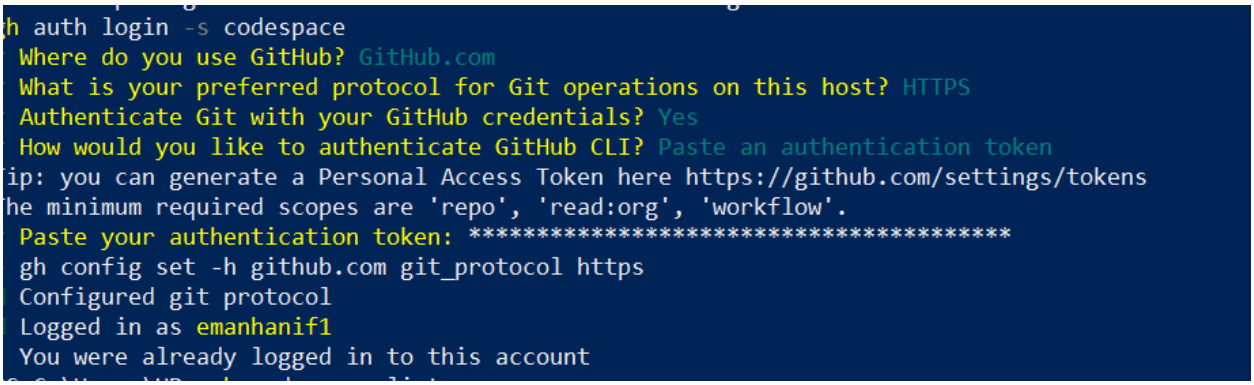
---

2. Authenticating GitHub CLI for Codespaces

In order to access Codespaces through GH CLI, authentication is needed with specific permissions.

We run:

gh auth login -s codespace



Explanation:

- gh auth login starts the login process.
- The -s codespace flag ensures that GH CLI requests the **Codespace scope**, which allows us to list, create, and connect to Codespaces.

After running this command:

- GitHub asks us to generate a **Personal Access Token (classic)**.
- We must select the following scopes:
  - **admin:org** – allows organizational access if needed.
  - **codespace** – required to connect and interact with Codespaces.
  - **repo** – ensures read/write access to the repository inside the Codespace.

Once authenticated, GitHub CLI is ready to communicate with the Codespace environment.

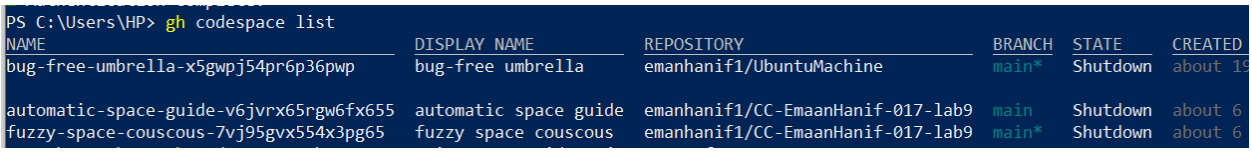
Screenshot Required:
task1_gh_auth_login.png

3. Listing Available Codespaces

To connect to a Codespace, we first need to check what Codespaces exist in our account.

Command Used:

gh codespace list



Explanation:

- This command displays all Codespaces linked to the authenticated GitHub account.
- It shows:
  - Codespace name (important for SSH)
  - Repository
  - Branch
  - State (Available / Shutdown)
  - Created date

From this list, we select the Codespace we want to connect to.

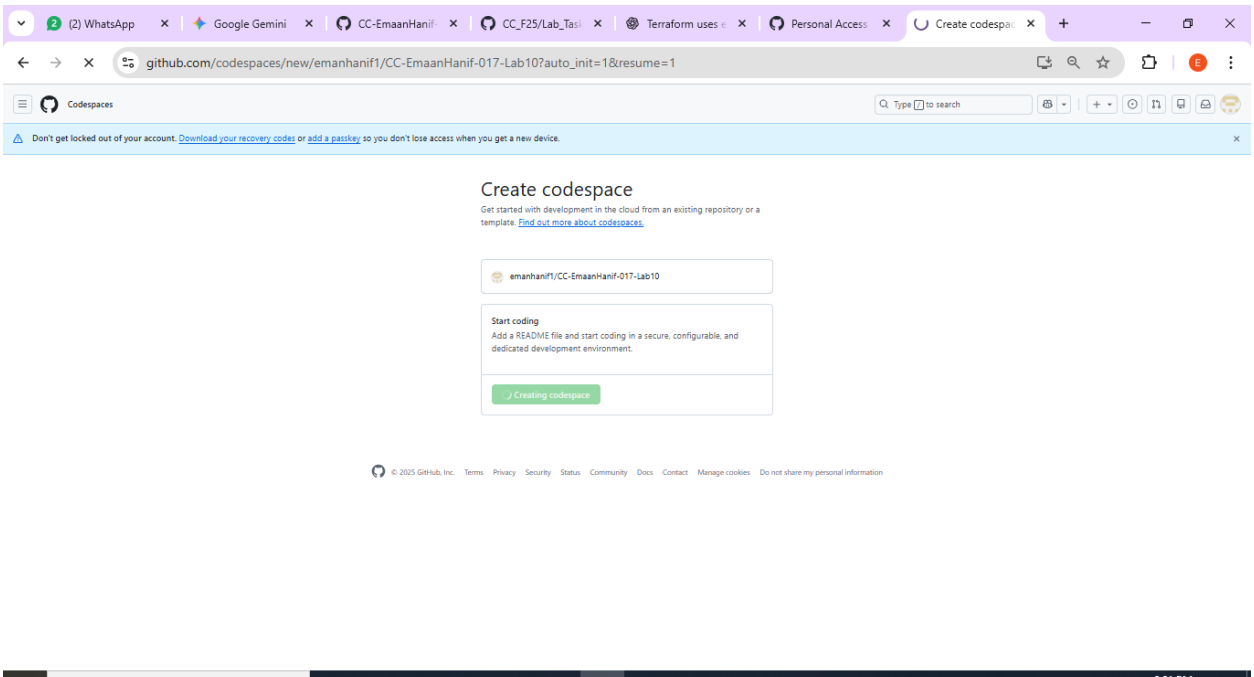
Screenshot Required:
task1_codespace_list.png
(The output must include the codespace name.)

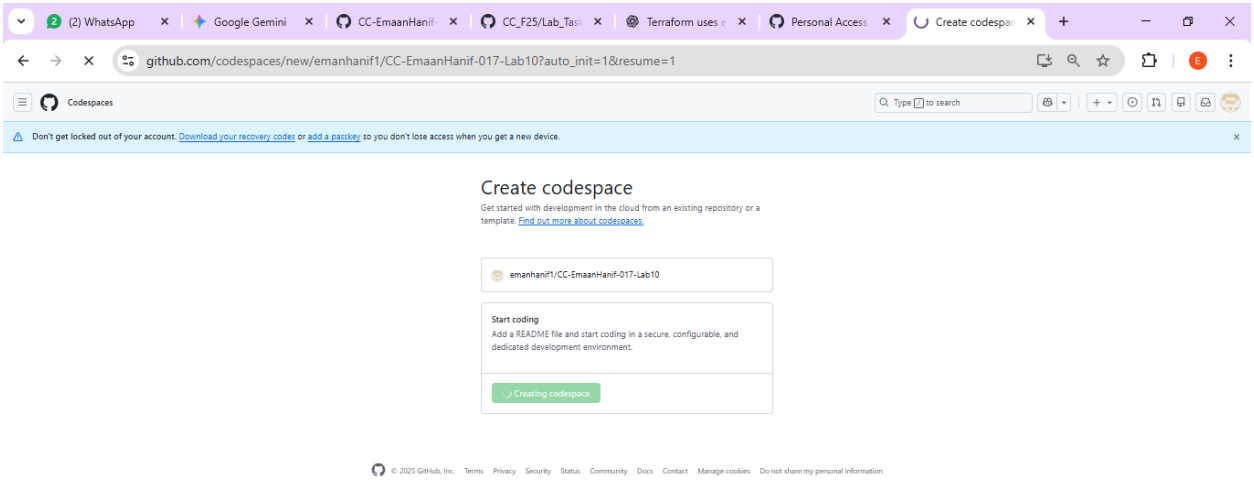
---

4. Connecting to a Codespace via SSH

- First, create a **new repository** CC-EmaanHanif-017-lab10 on GitHub.

- Open the repository and manually **create a Codespace** from the **Code → Codespaces → Create Codespace** option.

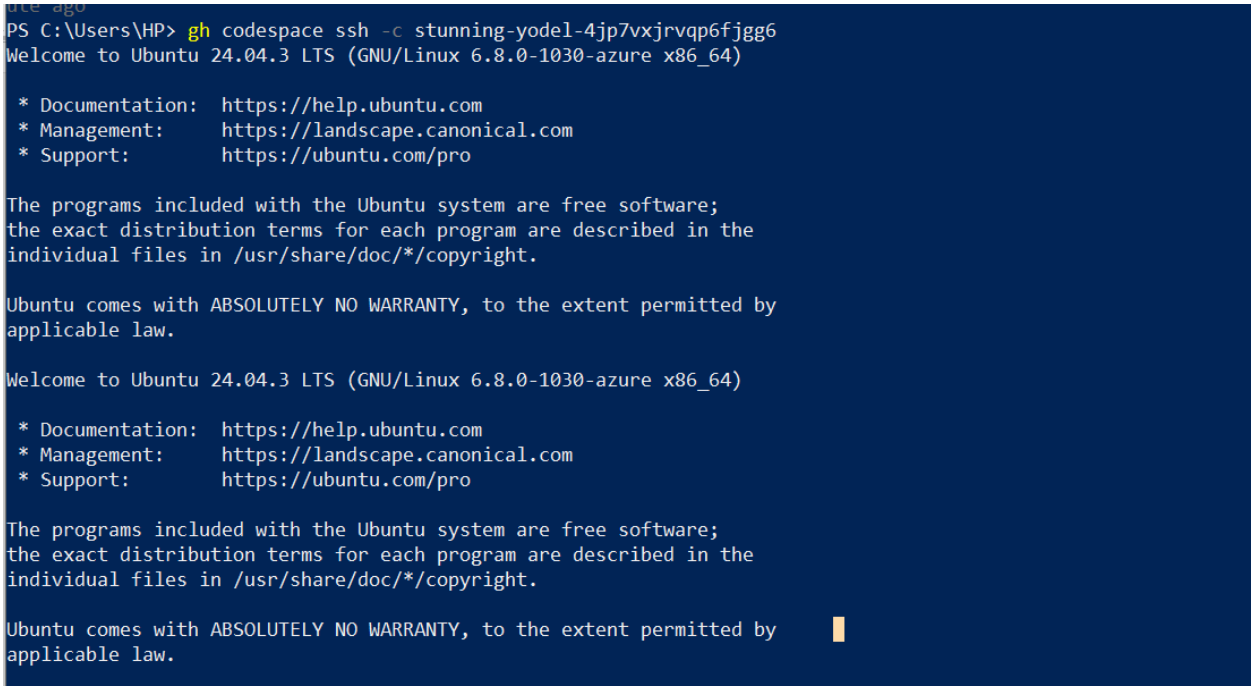




After identifying the Codespace name, we connect using SSH:

Command:

gh codespace ssh -c <name_of_codespace>



Explanation:

- gh codespace ssh initiates an SSH session to a GitHub Codespace.
- The -c option specifies the name of the Codespace to connect to.
- After a successful connection, the terminal prompt changes to :

codespace ➜ /workspaces/CC-EmaanHanif-017-lab9

- This prompt indicates that you are now inside the **remote development environment** of the Codespace and can execute commands directly within it.

Screenshot Required:
task1_codespace_ssh_connected.png

Task 2 — Install AWS CLI, Terraform CLI, and Provider Setup

Goal

The purpose of this task is to install **AWS CLI** inside the Codespace, configure it with proper credentials, and verify connectivity. This ensures the environment is ready to interact with AWS resources using Terraform or CLI commands.

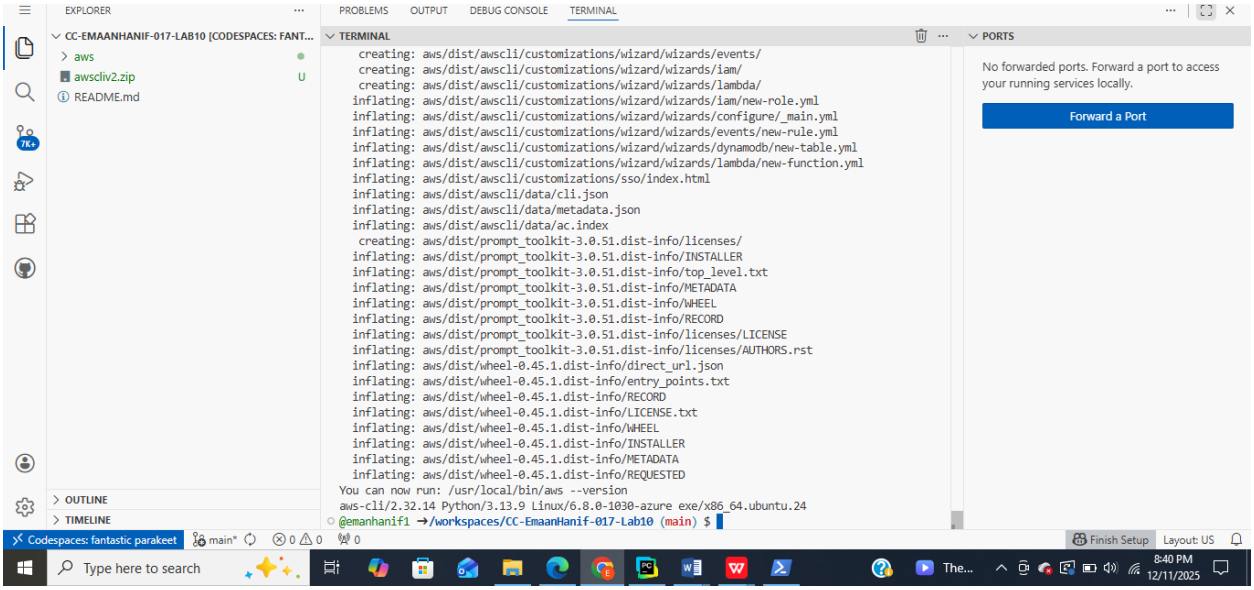
A. Install AWS CLI

Step: Open your Codespace shell and run the following commands:

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
aws –version
```

Explanation

- `curl` downloads the AWS CLI installer from the official AWS website.
- `unzip` extracts the downloaded ZIP file.
- `sudo ./aws/install` installs AWS CLI with administrative privileges.
- `aws --version` confirms that AWS CLI is installed and displays the version.

- 

Screenshot Required:
task2_aws_install_and_version.png — Show the terminal output with the AWS CLI version.

B. Configure AWS CLI

Run the configuration command:

aws configure

Explanation

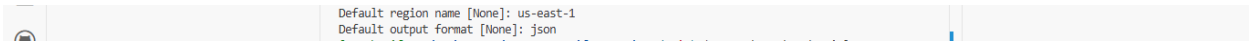
Prompts for:

1. **AWS Access Key ID** → Unique identifier for your AWS account credentials.
2. **AWS Secret Access Key** → Secret key associated with your account (keep it private!).
3. **Default region** → AWS region for commands (example: us-east-1, us-west-2).
4. **Default output format** → How command output is displayed (json, text, table).

Configuration saves:

- Credentials in ~/.aws/credentials
- Default settings in ~/.aws/config

Ensure sensitive values are **redacted** in screenshots.



2. Check if AWS CLI is Configured

Run these commands to see if configuration exists:

cat ~/.aws/credentials

- This file stores **AWS credentials** for the user.
- **Purpose:**
  - Confirms that AWS Access Key ID and Secret Access Key were saved correctly during aws configure.
  - Allows AWS CLI to authenticate your commands without entering keys every time.
- **Important:** Never share this file or your keys publicly.

cat ~/.aws/config

- This file stores **AWS configuration settings**
- **Purpose:**
  - Ensures your CLI commands run in the correct **AWS region**.
  - Sets **default output format** (json, table, or text) for easier readability

```
@emanhanif1 →/workspaces/CC-EmaanHanif-017-Lab10 (main) $ cat ~/.aws/credentials
cat ~/.aws/config
[default]
```

```
aws_access_key_id = AKIA3
aws_secret_access_key = 1
[default]
```

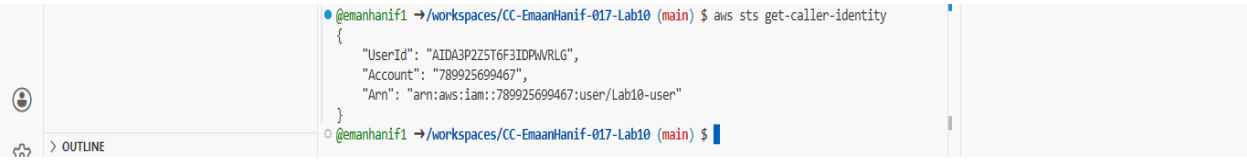
```
region = us-east-1
output = json
```

3. Check Connectivity to AWS

If CLI is installed and credentials exist, verify you can access AWS:

aws sts get-caller-identity

- If it returns UserId, Account, and Arn, credentials are working.
- If it returns an error, the keys may be missing or incorrect.



- UserId → Unique identifier for your AWS user.
- Account → Your AWS account number.
- Arn → Amazon Resource Name for your user; identifies the AWS user or role.

B.Install Terraform CLI

Goal

To install **Terraform CLI** in the Codespace and verify that it is working correctly. Terraform is used to provision and manage cloud infrastructure in a declarative way.

A. Add HashiCorp GPG Key

Command:

wget -O - https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg

```
@emanhanif1 ➜/workspaces/CC-EmaanHanif-017-Lab10 (main) $ wget -O - https://apt.releases.hashicorp.com/gpg | sudo gpg --de
armor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg
--2025-12-11 18:49:42--  https://apt.releases.hashicorp.com/gpg
Resolving apt.releases.hashicorp.com (apt.releases.hashicorp.com)... 18.161.229.117, 18.161.229.45, 18.161.229.23, ...
Connecting to apt.releases.hashicorp.com (apt.releases.hashicorp.com)|18.161.229.117|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3980 (3.9K) [binary/octet-stream]
Saving to: 'STDOUT'

-                     100%[=============================>]   3.89K  --.-KB/s    in 0s

2025-12-11 18:49:43 (140 MB/s) - written to stdout [3980/3980]
```

Explanation

- Downloads the official HashiCorp GPG key.
- Converts it to a format usable by apt to **verify the authenticity of Terraform packages**.
- curl -fsSL <URL> → Downloads the GPG key file from HashiCorp's official website.
- sudo gpg --dearmor → Converts the key to a format that can be used by the system's package manager.
- -o /usr/share/keyrings/hashicorp-archive-keyring.gpg → Saves the key in the system keyrings directory.

B. Add Terraform Repository

```
@emanhanif1 ➜/workspaces/CC-EmaanHanif-017-Lab10 (main) $ echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/shar
e/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(grep -oP '(?<=UBUNTU_CODENAME=).*' /etc/os-r
elease || lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list
deb [arch=amd64 signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com noble main
@emanhanif1 ➜/workspaces/CC-EmaanHanif-017-Lab10 (main) $
```

Explanation

- Adds the **official HashiCorp APT repository** to your system's sources list.
- Ensures that you can install Terraform using apt and get updates automatically.

C. Update Package List

Command:

sudo apt update

```
@emanhanif1 ➜/workspaces/CC-EmaanHanif-017-Lab10 (main) $ sudo apt update
Get:1 https://repo.anaconda.com/pkgs/misc/debrepo/conda stable InRelease [3961 B]
Get:2 https://apt.releases.hashicorp.com noble InRelease [12.9 kB]
Get:3 https://dl.yarnpkg.com/debian stable InRelease
Get:4 https://repo.anaconda.com/pkgs/misc/debrepo/conda stable/main amd64 Packages [4557 B]
Get:5 https://apt.releases.hashicorp.com noble/main amd64 Packages [263 kB]
Get:6 https://dl.yarnpkg.com/debian stable/main all Packages [11.8 kB]
Get:7 https://dl.yarnpkg.com/debian stable/main amd64 Packages [11.8 kB]
Get:8 https://packages.microsoft.com/repos/microsoft-ubuntu-noble-prod noble InRelease [3600 B]
Get:9 http://archive.ubuntu.com/ubuntu noble InRelease [256 kB]
Get:10 https://packages.microsoft.com/repos/microsoft-ubuntu-noble-prod noble/main all Packages [643 B]
Get:11 https://packages.microsoft.com/repos/microsoft-ubuntu-noble-prod noble/main amd64 Packages [75.0 kB]
Get:12 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:13 http://archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:14 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Packages [33.1 kB]
Get:15 http://archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:16 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Packages [1183 kB]
Get:17 http://archive.ubuntu.com/ubuntu noble/main amd64 Packages [1808 kB]
Get:18 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Packages [2874 kB]
Get:19 http://archive.ubuntu.com/ubuntu noble/restricted amd64 Packages [117 kB]
Get:20 http://archive.ubuntu.com/ubuntu noble/universe amd64 Packages [19.3 MB]
Get:21 http://security.ubuntu.com/ubuntu noble-security/main amd64 Packages [1736 kB]
Get:22 http://archive.ubuntu.com/ubuntu noble/multiverse amd64 Packages [331 kB]
Get:23 http://archive.ubuntu.com/ubuntu noble-updates/universe amd64 Packages [1944 kB]
Get:24 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [2118 kB]
Get:25 http://archive.ubuntu.com/ubuntu noble-updates/restricted amd64 Packages [3048 kB]
Get:26 http://archive.ubuntu.com/ubuntu noble-updates/multiverse amd64 Packages [35.9 kB]
Get:27 http://archive.ubuntu.com/ubuntu noble-backports/universe amd64 Packages [34.3 kB]
Get:28 http://archive.ubuntu.com/ubuntu noble-backports/main amd64 Packages [49.4 kB]
97% [20 Packages store 0 B]
```

Explanation

- Updates your package list to include the newly added HashiCorp repository.

D. Install Terraform

Command:

sudo apt install terraform

```
@emanhanif1 ➜ /workspaces/CC-EmaanHanif-017-Lab10 (main) $ sudo apt install terraform
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  terraform
0 upgraded, 1 newly installed, 0 to remove and 30 not upgraded.
Need to get 30.6 MB of archives.
After this operation, 101 MB of additional disk space will be used.
Get:1 https://apt.releases.hashicorp.com noble/main amd64 terraform amd64 1.14.2-1 [30.6 MB]
Fetched 30.6 MB in 0s (66.5 MB/s)
Selecting previously unselected package terraform.
(Reading database ... 58629 files and directories currently installed.)
Preparing to unpack .../terraform_1.14.2-1_amd64.deb ...
Unpacking terraform (1.14.2-1) ...
Setting up terraform (1.14.2-1) ...
```

Explanation

- Installs the latest Terraform version from the official HashiCorp repository.

E. Verify Installation

Commands:

which terraform
terraform –version

```
Setting up terraform (1.14.2-1) ...
@emanhanif1 ➜ /workspaces/CC-EmaanHanif-017-Lab10 (main) $ which terraform
terraform --version
/usr/bin/terraform
Terraform v1.14.2
on linux_amd64
@emanhanif1 ➜ /workspaces/CC-EmaanHanif-017-Lab10 (main) $
```

Explanation

- which terraform shows the path where Terraform is installed.
- terraform --version displays the installed version and confirms that Terraform is working.

Screenshot Required:
task2_terraform_install_and_version.png — Show terminal output of installation path and version.

Summary

1. Added HashiCorp GPG key for security.
2. Added the Terraform APT repository.
3. Updated package list and installed Terraform CLI.
4. Verified installation by checking path and version.

Terraform Provider Configuration

Goal

To configure Terraform to use AWS as the provider, initialize Terraform, and verify setup. This enables Terraform to provision AWS resources using the credentials and configuration already set in the Codespace.

A. Create main.tf File

Command:

```
vim main.tf
```

Explanation

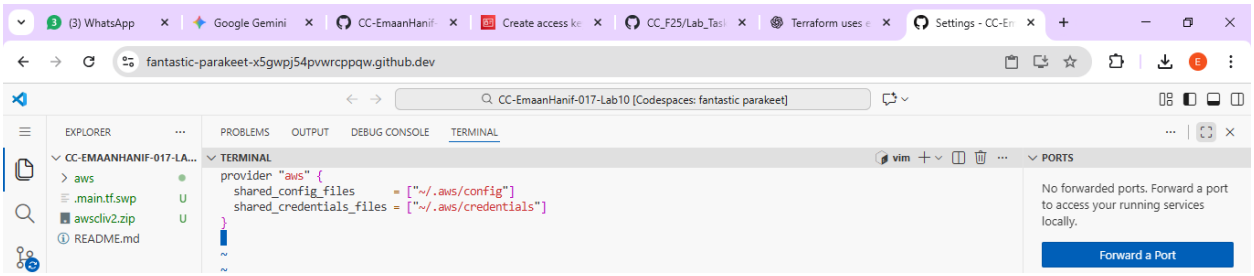
- main.tf is the main Terraform configuration file.
- Using vim (or any editor), you create this file to define the provider and resources for your project.

Screenshot Required:
task2_provider_file_creation.png — Terminal showing vim main.tf open.

B. Add AWS Provider Block

Inside main.tf, write:

```
provider "aws" {
  shared_config_files      = ["~/.aws/config"]
  shared_credentials_files = ["~/.aws/credentials"]
}
```



Explanation

- provider "aws" tells Terraform to use AWS as the cloud provider.
- shared_config_files points to the AWS config file.
- shared_credentials_files points to the AWS credentials file.
- This setup ensures Terraform uses your preconfigured AWS CLI credentials for all operations.

Screenshot Required:
task2_provider_block.png — Show the saved provider block in main.tf.

C. Save and Exit Vim

- Press ESC then type :wq and press Enter to save changes and quit Vim.

Screenshot Optional:
task2_vim_save_main_tf.png — Confirmation of file saved.

D. Initialize Terraform

Command:

```
terraform init
```

```
on linux_amd64
@emanhanif1 ➜/workspaces/CC-EmaanHanif-017-Lab10 (main) $ vim main.tf
@emanhanif1 ➜/workspaces/CC-EmaanHanif-017-Lab10 (main) $ terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v6.26.0...
- Installed hashicorp/aws v6.26.0 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

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.
@emanhanif1 ➜/workspaces/CC-EmaanHanif-017-Lab10 (main) $
```

Explanation

- terraform init initializes the Terraform working directory.
- Downloads provider plugins (in this case, AWS) and sets up .terraform/ directory for managing state and modules.

Screenshot Required:
task2_terraform_init_output.png — Show output of terraform init.

E. Check Terraform Lock File

Command:

cat .terraform.lock.hcl

```
@emanhanif1 →/workspaces/CC-EmaanHanif-017-Lab10 (main) $ cat .terraform.lock.hcl
# This file is maintained automatically by "terraform init".
# Manual edits may be lost in future updates.

provider "registry.terraform.io/hashicorp/aws" {
  version = "6.26.0"
  hashes = [
    "h1:B7X8EU6aZ9KzIvP0VBDhhgadgjXIrUgMXt/pJ6EUXvo=",
    "zh:038fd943de79acd9f9f73106fa0eba588c6a0d4e0993e146f51f3aa043728c5f",
    "zh:06fa0177d33d3d3f9cb7e205fbeb1c4c3095ba637e2b4d292429401ec5612e81",
    "zh:212714fc8b6ee57e26d11d0fdf2ecfe23b37a6eac1008b399c1d790528c3f072",
    "zh:3197725d772f360e9e466b68a5ba67363e9f6786809c9adefc50f7f7e525bf42",
    "zh:33385539f3e3fafb96c6036421fd72b05c76505eeefaaff8a089c3eeba25db65",
    "zh:4ce065e0d3c384d11c1b59fe92582d331aae27ff6e019ace07b8cedef5653aae",
    "zh:67863d6ff5517db2c0b8097443708dca548f1922d2e08ad76a98d493ff480cb1",
    "zh:771ccf61fc107013b437b0a05cdb342823a99200653bfe9b892702b9fd8997fe",
    "zh:80adcf83bef9d683606c48bbe53cbb2b5a04878641674e957939b5e8f124ada0",
    "zh:9675c7f209db8e64ba2d5197acc8ba0073bd73b48c3dd61a1961a44844bc8a81",
    "zh:9b12af85486a96aedd8d7984b0ff811a4b42e3d88dad1a3fb4c0b580d04fa425",
    "zh:b47d0f5eff91c94c5d5677815b9361e64dfbe2ee2d59ba2867e2d0f5fa7181e4",
    "zh:b4933663b8d6cc1cfb51aa47bd8f26c06012ee2e278e57663faffdc722dd5baa",
    "zh:d53a94ecdb6b68a8dc19ec6e16ba2d4c2acde575af254d1b8b80143e57c76abf",
    "zh:e7cb8c1770c6f87c5ce1d3e28b838380bb8e5296dd03034b796168de8be1c7ec",
  ]
}
@emanhanif1 →/workspaces/CC-EmaanHanif-017-Lab10 (main) $
```

Explanation

- .terraform.lock.hcl locks provider versions used in this configuration.
- Ensures consistent Terraform behavior across environments and team members.

Screenshot Required:
task2_terraform_lock_hcl.png — Show contents of .terraform.lock.hcl.

F. List Terraform Directory

Command:

ls .terraform

```
@emanhanif1 →/workspaces/CC-EmaanHanif-017-Lab10 (main) $ ls .terraform/
providers
```

Explanation

- Shows the internal Terraform directory created after terraform init.
- Contains provider plugins and other initialization files.

Screenshot Required:
task2_terraform_dir_ls.png — Output of ls .terraform/.

Summary

1. Created main.tf for AWS provider configuration.
2. Added provider block pointing to AWS CLI credentials and config files.
3. Initialized Terraform using terraform init.
4. Verified provider lock file and internal .terraform/ directory.

Task 3 — VPC/Subnet Creation, Initialization, Verification

Goal

To use Terraform to create an AWS VPC and Subnet, apply the changes, and verify the resources using AWS CLI.

A. Edit main.tf to Add VPC and Subnet Resources

Step: Open your Terraform configuration file:

```
vim main.tf
```

Add the following code inside main.tf:

```
resource "aws_vpc" "development_vpc" {
  cidr_block = "10.0.0.0/16"
}

resource "aws_subnet" "dev_subnet_1" {
  vpc_id            = aws_vpc.development_vpc.id
  cidr_block        = "10.0.10.0/24"
  availability_zone = "me-central-1a"
}
```

[Preview] README.md | Settings | main.tf U

main.tf

```
provider "aws" {
  shared_config_files      = ["~/.aws/config"]
  shared_credentials_files = ["~/.aws/credentials"]
}
  resource "aws_vpc" "development_vpc" {
  cidr_block = "10.0.0.0/16"
}

resource "aws_subnet" "dev_subnet_1" {
  vpc_id            = aws_vpc.development_vpc.id
  cidr_block        = "10.0.10.0/24"
  availability_zone = "me-central-1a"
}


```

Explanation

- resource "aws_vpc" → Creates a new VPC named development_vpc with CIDR 10.0.0.0/16.
- resource "aws_subnet" → Creates a subnet dev_subnet_1 inside the above VPC.
- vpc_id = aws_vpc.development_vpc.id → Links the subnet to the VPC created by Terraform.
- availability_zone = "me-central-1a" → Ensures the subnet is created in a specific AZ.

Screenshot Required:
task3_main_tf_resource_add.png — Terminal showing the added VPC and Subnet resources in main.tf.

B. Apply Terraform Configuration

Step: Run:

terraform apply

- Terraform will show a **plan** describing the changes it will make.
- Type yes when prompted to confirm.

```
@emanhanif1 ➜/workspaces/CC-EmaanHanif-017-Lab10 (main) $ terraform apply
      + availability_zone_id                          = (known after apply)
      + cidr_block                                    = "10.0.10.0/24"
      + enable_dns64                                  = false
      + enable_resource_name_dns_a_record_on_launch   = false
      + enable_resource_name_dns_aaaa_record_on_launch = false
      + id                                            = (known after apply)
      + ipv6_cidr_block_association_id                = (known after apply)
      + ipv6_native                                   = false
      + map_public_ip_on_launch                       = false
      + owner_id                                      = (known after apply)
      + private_dns_hostname_type_on_launch           = (known after apply)
      + region                                        = "us-east-1"
      + tags_all                                      = (known after apply)
      + vpc_id                                        = "vpc-0a86d1bc502c936e5"
    }

Plan: 1 to add, 0 to change, 0 to destroy.

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_subnet.dev_subnet_1: Creating...
aws_subnet.dev_subnet_1: Creation complete after 2s [id=subnet-0345fe49ae0d7d081]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

Explanation

- Terraform provisions the VPC and subnet in your AWS account.
- After successful execution, Terraform outputs the resource IDs and attributes.

 Terraform is going to **create 1 resource**: the subnet.

 Nothing else will be modified or destroyed.

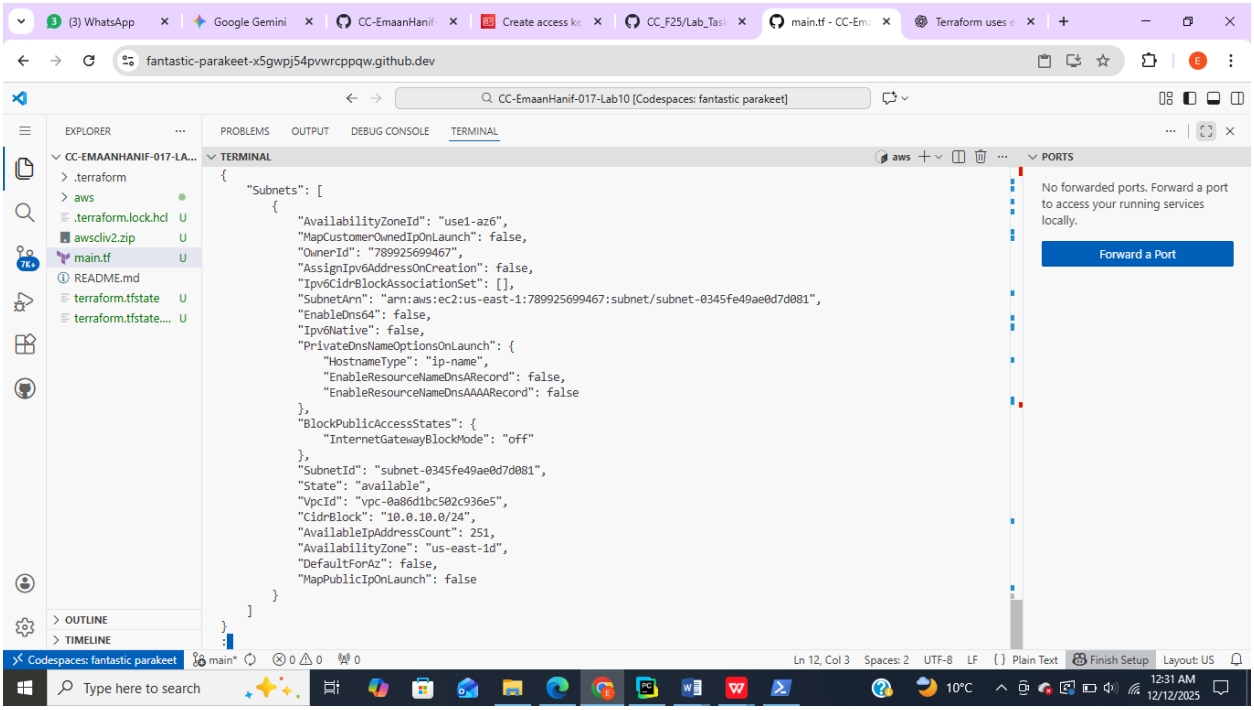
-

Screenshot Required:
task3_terraform_apply_vpc_subnet.png — Show successful terraform apply output with created resource IDs.

C. Verify Resources Using AWS CLI

1. Verify Subnet

aws ec2 describe-subnets --filter "Name=subnet-id,Values=<subnet-id>"



- Replace <subnet-id> with the ID output by Terraform for dev_subnet_1.

Screenshot Required:
task3_aws_cli_verify_subnet.png — Show AWS CLI output verifying the subnet exists.

---

2. Verify VPC

aws ec2 describe-vpcs --filter "Name=vpc-id,Values=<vpc-id>"

- Replace <vpc-id> with the ID output by Terraform for development_vpc.

```
0345fe49ae0d7d081"
}
@emanhanif1 ➜/workspaces/CC-EmaanHanif-017-Lab10 (main) $ aws ec2 describe-vpcs --filter "Name=vpc-id,Values=vpc-0a86d1bc5
02c936e5"
{
    "Vpcs": [
        {
            "OwnerId": "789925699467",
            "InstanceTenancy": "default",
            "CidrBlockAssociationSet": [
                {
                    "AssociationId": "vpc-cidr-assoc-0802d2cfa2c56b4df",
                    "CidrBlock": "10.0.0.0/16",
                    "CidrBlockState": {
                        "State": "associated"
                    }
                }
            ],
            "IsDefault": false,
            "BlockPublicAccessStates": {
                "InternetGatewayBlockMode": "off"
            },
            "VpcId": "vpc-0a86d1bc502c936e5",
            "State": "available",
            "CidrBlock": "10.0.0.0/16",
            "DhcpOptionsId": "dopt-071d81944d9a20fdc"
        }
    ]
}
@emanhanif1 ➜/workspaces/CC-EmaanHanif-017-Lab10 (main) $
```

Screenshot Required:
task3_aws_cli_verify_vpc.png — Show AWS CLI output verifying the VPC exists.

Summary

1. Edited main.tf to add AWS VPC and subnet resources.
2. Applied Terraform configuration to create resources.
3. Verified the existence of VPC and subnet using AWS CLI comman

main.tf

```
provider "aws" {
  shared_config_files      = ["~/.aws/config"]
  shared_credentials_files = ["~/.aws/credentials"]
}
  data "aws_vpc" "existing_vpc" {
  default = true
}

resource "aws_subnet" "dev_subnet_1_existing" {
  vpc_id            = data.aws_vpc.existing_vpc.id
  cidr_block        = "172.31.48.0/24"
  availability_zone = "us-east-1a"
}
```

Task 4 — Data Source, Targeted Destroy, Tags

A. Add Data Source + New Subnet Resource

1. Open file

vim main.tf

So write this:

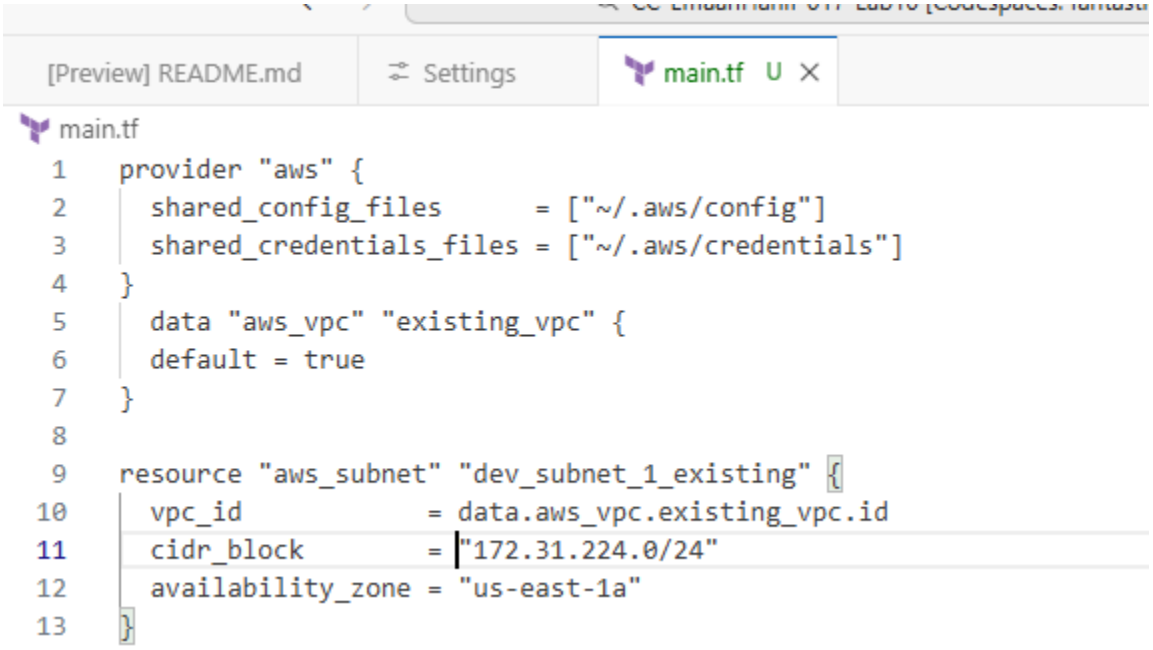
```
data "aws_vpc" "existing_vpc" {
  default = true
}

resource "aws_subnet" "dev_subnet_1_existing" {
  vpc_id            = data.aws_vpc.existing_vpc.id
  cidr_block        = "172.31.48.0/24"
  availability_zone = "us-east-1a"
}
```

3. Save file

Press:

ESC
:wq

Take screenshot → task4_main_tf_datasource_resource_add.png



Use a CIDR block that does NOT conflict

B. Apply Only These Changes

Run:

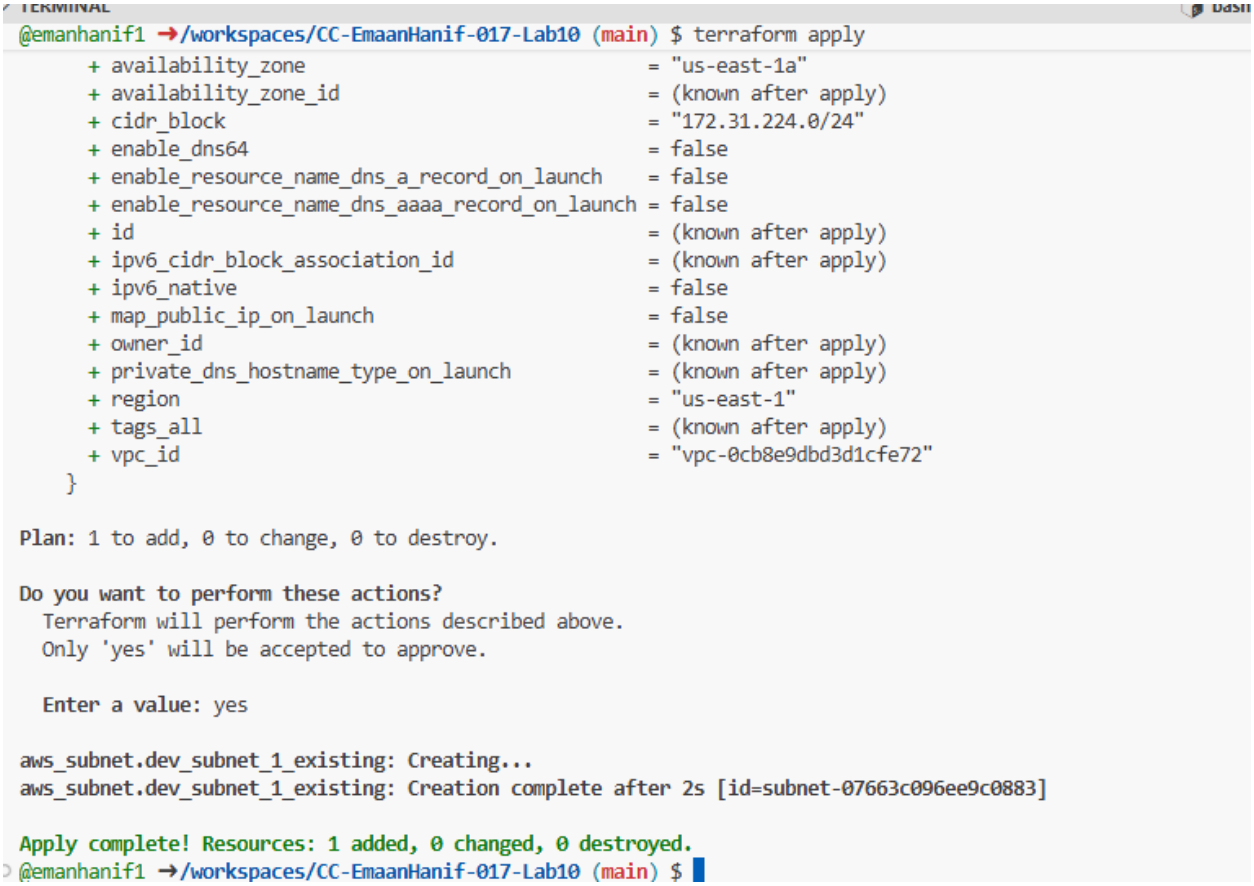
terraform apply

Type:

yes

This should show:

- 1 resource to add
- Only aws_subnet.dev_subnet_1_existing created
  (because VPC is a data source, not created)



Take screenshot → task4_terraform_apply_datasource_resource.png

Targeted Destroy (Destroy ONLY ONE Resource)

You are destroying ONLY the subnet created with data source:

terraform destroy -target=aws_subnet.dev_subnet_1_existing

🎯 This destroys only this resource, not your whole Terraform environment.

📸 Screenshot name: task4_terraform_destroy_targeted.png

```
@emanhanif1 →/workspaces/CC-EmaanHanif-017-Lab10 (main) $ terraform destroy -target=aws_subnet.dev_subnet_1_existing
data.aws_vpc.existing_vpc: Reading...
data.aws_vpc.existing_vpc: Read complete after 2s [id=vpc-0cb8e9dbd3d1cfe72]
aws_subnet.dev_subnet_1_existing: Refreshing state... [id=subnet-07663c096ee9c0883]

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

Terraform will perform the following actions:

  # aws_subnet.dev_subnet_1_existing will be destroyed
  - resource "aws_subnet" "dev_subnet_1_existing" {
      - arn                                            = "arn:aws:ec2:us-east-1:789925699467:subnet/subnet-07663c096ee
3" -> null
      - assign_ipv6_address_on_creation                = false -> null
      - availability_zone                              = "us-east-1a" -> null
      - availability_zone_id                           = "use1-az1" -> null
      - cidr_block                                     = "172.31.224.0/24" -> null
      - enable_dns64                                   = false -> null
      - enable_lni_at_device_index                     = 0 -> null
      - enable_resource_name_dns_a_record_on_launch    = false -> null
      - enable_resource_name_dns_aaaa_record_on_launch = false -> null
      - id                                             = "subnet-07663c096ee9c0883" -> null
      - ipv6_native                                    = false -> null
```

Refresh State

After destroying one resource, Terraform must sync state with AWS:

terraform refresh

This updates the .tfstate file to match AWS.

📸 Screenshot name: **task4_terraform_refresh_state.png**

Re-apply Configuration

Now recreate whatever is in main.tf:

terraform apply

Type:

yes

```
TERMINAL                                                                    terraform
@emanhanif1 ➜/workspaces/CC-EmaanHanif-017-Lab10 (main) $ terraform apply
  # aws_subnet.dev_subnet_1_existing will be created
  + resource "aws_subnet" "dev_subnet_1_existing" {
      + arn                                            = (known after apply)
      + assign_ipv6_address_on_creation                = false
      + availability_zone                              = "us-east-1a"
      + availability_zone_id                           = (known after apply)
      + cidr_block                                     = "172.31.224.0/24"
      + enable_dns64                                   = false
      + enable_resource_name_dns_a_record_on_launch    = false
      + enable_resource_name_dns_aaaa_record_on_launch = false
      + id                                             = (known after apply)
      + ipv6_cidr_block_association_id                 = (known after apply)
      + ipv6_native                                    = false
      + map_public_ip_on_launch                        = false
      + owner_id                                       = (known after apply)
      + private_dns_hostname_type_on_launch            = (known after apply)
      + region                                         = "us-east-1"
      + tags_all                                       = (known after apply)
      + vpc_id                                         = "vpc-0cb8e9dbd3d1cfe72"
    }

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
```

📸 Screenshot name: **task4_terraform_apply_after_refresh.png**

Destroy ALL Resources

Now completely remove everything Terraform created:

terraform destroy

Type:

Yes

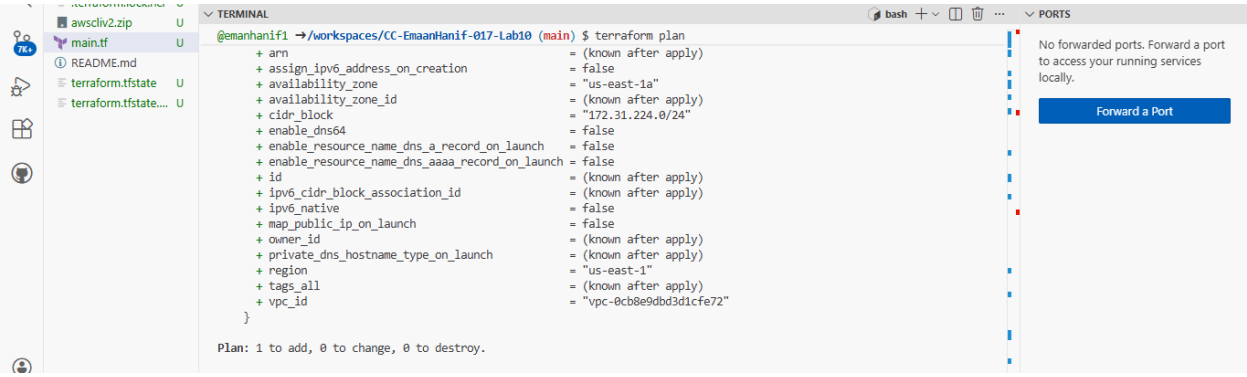
```
@emanhanif1 ➜/workspaces/CC-EmaanHanif-017-Lab10 (main) $ terraform destroy
      - ipv6_native                                  = false -> null
      - map_customer_owned_ip_on_launch              = false -> null
      - map_public_ip_on_launch                      = false -> null
      - owner_id                                     = "789925699467" -> null
      - private_dns_hostname_type_on_launch          = "ip-name" -> null
      - region                                       = "us-east-1" -> null
      - tags                                         = {} -> null
      - tags_all                                     = {} -> null
      - vpc_id                                       = "vpc-0cb8e9dbd3d1cfe72" -> null
        # (4 unchanged attributes hidden)
    }

Plan: 0 to add, 0 to change, 1 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

aws_subnet.dev_subnet_1_existing: Destroying... [id=subnet-03aa934c2bc4d293e]
aws_subnet.dev_subnet_1_existing: Destruction complete after 1s

Destroy complete! Resources: 1 destroyed.
```

📸 Screenshot name: **task4_terraform_destroy_all.png**

Run Terraform Plan

See what Terraform *wants* to create now:

terraform plan



📸 Screenshot name: **task4_terraform_plan_output.png**

Apply Again (Re-create Everything)

terraform apply

Type:

Yes

```
TERMINAL
@emanhanif1 →/workspaces/CC-EmaanHanif-017-Lab10 (main) $ terraform apply
      + enable_resource_name_dns_aaaa_record_on_launch = false
      + id                                             = (known after apply)
      + ipv6_cidr_block_association_id                 = (known after apply)
      + ipv6_native                                    = false
      + map_public_ip_on_launch                        = false
      + owner_id                                       = (known after apply)
      + private_dns_hostname_type_on_launch            = (known after apply)
      + region                                         = "us-east-1"
      + tags_all                                       = (known after apply)
      + vpc_id                                         = "vpc-0cb8e9dbd3d1cfe72"
    }

Plan: 1 to add, 0 to change, 0 to destroy.

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_subnet.dev_subnet_1_existing: Creating...
aws_subnet.dev_subnet_1_existing: Creation complete after 2s [id=subnet-0f34a760a5b528053]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
@emanhanif1 →/workspaces/CC-EmaanHanif-017-Lab10 (main) $
```

📸 Screenshot name: task4_terraform_apply_after_destroy.png

---

update main.tf — Add Tags

Open main.tf:

vim main.tf

```
@emanhanif1 →/workspaces/CC-EmaanHanif-017-Lab10 (main) $ vim main.tf
```

Add the tag blocks exactly like this:

✓ If keeping your previous resources, your main.tf should look like:

[Preview] README.md | Settings | main.tf U

main.tf

```
provider "aws" {
  shared_config_files      = ["~/.aws/config"]
  shared_credentials_files = ["~/.aws/credentials"]
}

data "aws_vpc" "existing_vpc" {
  default = true
}

resource "aws_vpc" "development_vpc" {
  cidr_block = "10.0.0.0/16"
  tags = {
    Name   = "development"
    vpc_env = "dev"
  }
}

resource "aws_subnet" "dev_subnet_1" {
  vpc_id            = aws_vpc.development_vpc.id
  cidr_block        = "10.0.10.0/24"
  availability_zone = "us-east-1a"
  tags = {
    Name = "subnet-1-dev"
  }
}

resource "aws_subnet" "dev_subnet_1_existing" {
  vpc_id            = data.aws_vpc.existing_vpc.id
```

📸 Save screenshot:
task4_main_tf_tagging.png

Run Refresh

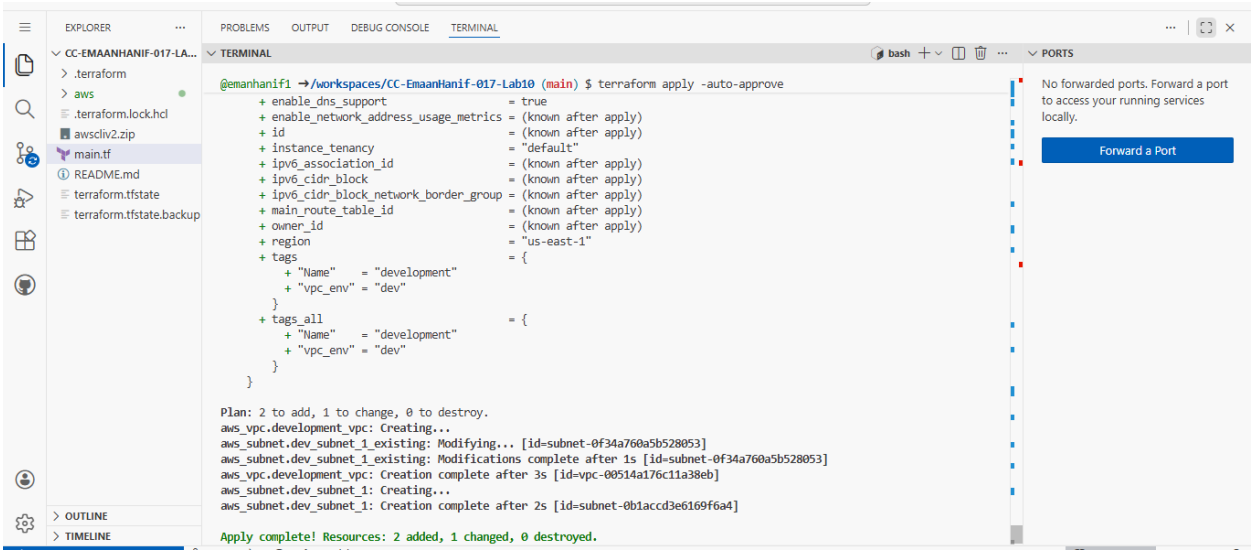
terraform refresh

📸 Save screenshot:
task4_terraform_refresh_tagging.png (optional)

Apply the updated configuration

terraform apply -auto-approve

This will apply the tag changes.



📸 Save screenshot:
task4_terraform_apply_tagging.png

Remove vpc_env tag

Now edit main.tf again and **REMOVE this line**:

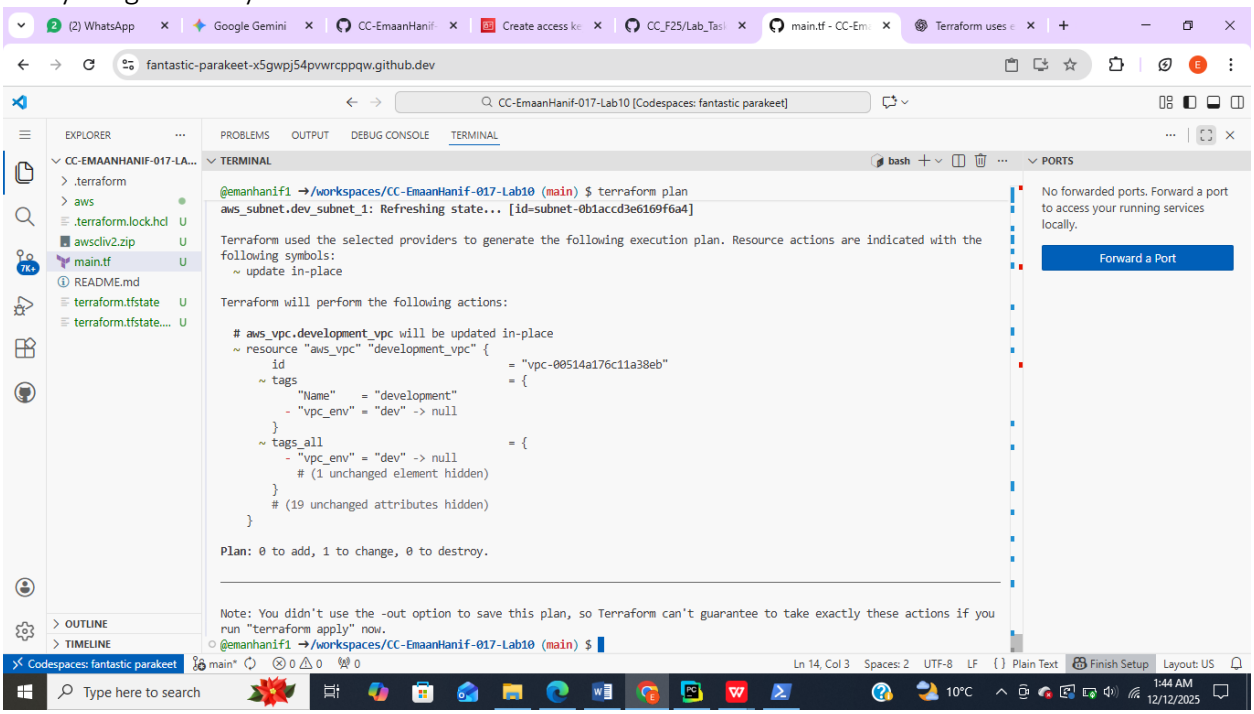
vpc_env = "dev"

So the VPC block becomes:

📸 Save screenshot after editing:
task4_main_tf_remove_tag.png (optional)

Re-plan to show tag removal

terraform plan

- Terraform will **update in-place**, meaning no new VPC is created and nothing is destroyed.
- Only the **tags are changed**.
- The tag vpc_env = "dev" will be **removed**.
- Everything else stays the same.
- 

📸 Save screenshot:
task4_terraform_plan_remove_tag.png

Apply the tag removal

terraform apply -auto-approve

```
@emanhanif1 →/workspaces/CC-EmaanHanif-017-Lab10 (main) $ terraform apply -auto-approve
aws_subnet.dev_subnet_1_existing: Refreshing state... [id=subnet-0f34a760a5b528053]
aws_subnet.dev_subnet_1: Refreshing state... [id=subnet-0b1accd3e6169f6a4]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the
following symbols:
  ~ update in-place

Terraform will perform the following actions:

  # aws_vpc.development_vpc will be updated in-place
  ~ resource "aws_vpc" "development_vpc" {
        id                                   = "vpc-00514a176c11a38eb"
      ~ tags                                 = {
            "Name"    = "development"
          - "vpc_env" = "dev" -> null
        }
      ~ tags_all                             = {
          - "vpc_env" = "dev" -> null
            # (1 unchanged element hidden)
        }
        # (19 unchanged attributes hidden)
    }

Plan: 0 to add, 1 to change, 0 to destroy.
aws_vpc.development_vpc: Modifying... [id=vpc-00514a176c11a38eb]
aws_vpc.development_vpc: Modifications complete after 3s [id=vpc-00514a176c11a38eb]

Apply complete! Resources: 0 added, 1 changed, 0 destroyed.
@emanhanif1 →/workspaces/CC-EmaanHanif-017-Lab10 (main) $
```

📸 Save screenshot:
task4_terraform_apply_remove_tag.png

Task 5 — State File Inspection & Terraform State Commands

destroy all resources

terraform destroy

- Type yes to confirm.
- This deletes **all resources** managed by Terraform.

📸 Screenshot → **task5_terraform_destroy.png**

Inspect state files after destroy

Check main state file

cat terraform.tfstate

```
@emanhanif1 ➜/workspaces/CC-EmaanHanif-017-Lab10 (main) $ cat terraform.tfstate
{
  "version": 4,
  "terraform_version": "1.14.2",
  "serial": 30,
  "lineage": "58757615-a9b6-593a-1461-3fc47c6790f0",
  "outputs": {},
  "resources": [],
  "check_results": null
}
```

- After destroy, this file should be **empty or only contain metadata**.

📸 Screenshot → **task5_terraform_state_file_empty.png**

Check backup state file

cat terraform.tfstate.backup

```
@emanhanif1 →/workspaces/CC-EmaanHanif-017-Lab10 (main) $ cat terraform.tfstate.backup
{
  "version": 4,
  "terraform_version": "1.14.2",
  "serial": 25,
  "lineage": "58757615-a9b6-593a-1461-3fc47c6790f0",
  "outputs": {},
  "resources": [
    {
      "mode": "data",
      "type": "aws_vpc",
      "name": "existing_vpc",
      "provider": "provider[\"registry.terraform.io/hashicorp/aws\"]",
      "instances": [
        {
          "schema_version": 0,
          "attributes": {
            "arn": "arn:aws:ec2:us-east-1:789925699467:vpc/vpc-0cb8e9dbd3d1cfe72",
            "cidr_block": "172.31.0.0/16",
            "cidr_block_associations": [
              {
                "association_id": "vpc-cidr-assoc-0554445ff1a68ac71",
                "cidr_block": "172.31.0.0/16",
                "state": "associated"
              }
            ],
            "default": true,
            "dhcp_options_id": "dopt-071d81944d9a20fdc",
```

- This file keeps the **previous state** before destroy.

📸 Screenshot → **task5_terraform_state_backup_prev.png**

Recreate resources

terraform apply

```
@emanhanif1 →/workspaces/CC-EmaanHanif-017-Lab10 (main) $ terraform apply
data.aws_vpc.existing_vpc: Reading...
data.aws_vpc.existing_vpc: Read complete after 3s [id=vpc-0cb8e9dbd3d1cfe72]

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

Terraform will perform the following actions:

  # aws_subnet.dev_subnet_1 will be created
  + resource "aws_subnet" "dev_subnet_1" {
      + arn                                            = (known after apply)
      + assign_ipv6_address_on_creation                = false
      + availability_zone                              = "us-east-1a"
      + availability_zone_id                           = (known after apply)
      + cidr_block                                     = "10.0.10.0/24"
      + enable_dns64                                   = false
      + enable_resource_name_dns_a_record_on_launch    = false
      + enable_resource_name_dns_aaaa_record_on_launch = false
      + id                                             = (known after apply)
      + ipv6_cidr_block_association_id                 = (known after apply)
      + ipv6_native                                    = false
      + map_public_ip_on_launch                        = false
      + owner_id                                       = (known after apply)
      + private_dns_hostname_type_on_launch            = (known after apply)
      + region                                         = "us-east-1"
      + tags                                           = {
          + "Name" = "subnet-1-dev"
```

- Type yes (or use -auto-approve).
- All resources defined in main.tf will be **recreated**.

📸 Screenshot → **task5_terraform_apply_recreated.png**

View state files after recreate

Main state file

cat terraform.tfstate

```
@emanhanif1 →/workspaces/CC-EmaanHanif-017-Lab10 (main) $ terraform apply
@emanhanif1 →/workspaces/CC-EmaanHanif-017-Lab10 (main) $ cat terraform.tfstate
cat terraform.tfstate.backup
{
  "version": 4,
  "terraform_version": "1.14.2",
  "serial": 34,
  "lineage": "58757615-a9b6-593a-1461-3fc47c6790f0",
  "outputs": {},
  "resources": [
    {
      "mode": "data",
      "type": "aws_vpc",
      "name": "existing_vpc",
      "provider": "provider[\"registry.terraform.io/hashicorp/aws\"]",
      "instances": [
        {
          "schema_version": 0,
          "attributes": {
            "arn": "arn:aws:ec2:us-east-1:789925699467:vpc/vpc-0cb8e9dbd3d1cfe72",
            "cidr_block": "172.31.0.0/16",
            "cidr_block_associations": [
              {
                "association_id": "vpc-cidr-assoc-0554445ff1a68ac71",
                "cidr_block": "172.31.0.0/16",
                "state": "associated"
              }
            ],
            "default": true,
            "dhcp_options_id": "dopt-071d81944d9a20fdc"
```

- Should now be **populated with all current resources**.

📸 Screenshot → **task5_terraform_state_file_populated.png**

Backup file

cat terraform.tfstate.backup

```
cat terraform.tfstate.backup
            "account_id": "789925699467",
            "id": "subnet-06a0efde7f32823ac",
            "region": "us-east-1"
          },
          "private": "eyJlMmJmYjczMC1lY2FhLTExZTYtOGY4OC0zNDM2M2JjN2M0YzAiOnsiY3JlYXRlIjo2MDAwMDAwMDAwMDAsImRlbGV0ZSI6MTIw
DAwMDAwMDAwMH0sInNjaGVtYV92ZXJzaW9uIjoiMSJ9",
          "dependencies": [
            "aws_vpc.development_vpc"
          ]
        }
      ]
    },
    {
      "mode": "managed",
      "type": "aws_subnet",
      "name": "dev_subnet_1_existing",
      "provider": "provider[\"registry.terraform.io/hashicorp/aws\"]",
      "instances": [
        {
          "schema_version": 1,
          "attributes": {
            "arn": "arn:aws:ec2:us-east-1:789925699467:subnet/subnet-0dd952f8e63b3fe1e",
            "assign_ipv6_address_on_creation": false,
            "availability_zone": "us-east-1a",
            "availability_zone_id": "use1-az1",
            "cidr_block": "172.31.224.0/24",
            "customer_owned_ipv4_pool": "",
            "enable_dns64": false,
```

- After recreation, this backup may now be **empty or contain minimal metadata**.

📸 Screenshot → **task5_terraform_state_backup_empty.png**

List all resources

terraform state list

```
}
@emanhanif1 →/workspaces/CC-EmaanHanif-017-Lab10 (main) $ terraform state list
data.aws_vpc.existing_vpc
aws_subnet.dev_subnet_1
aws_subnet.dev_subnet_1_existing
aws_vpc.development_vpc
```

- Lists all resources managed by Terraform in the current state.

📸 Screenshot → **task5_terraform_state_list.png**

Show full attributes of a resource

terraform state show <resource-name>

- Replace <resource-name> with a resource, e.g.:

terraform state show aws_vpc.development_vpc

```
@emanhanif1 ➜/workspaces/CC-EmaanHanif-017-Lab10 (main) $ terraform state show aws_subnet.dev_subnet_1_existing
# aws_subnet.dev_subnet_1_existing:
resource "aws_subnet" "dev_subnet_1_existing" {
    arn                                            = "arn:aws:ec2:us-east-1:789925699467:subnet/subnet-0dd952f8e63b3fe1e"
    assign_ipv6_address_on_creation                = false
    availability_zone                              = "us-east-1a"
    availability_zone_id                           = "use1-az1"
    cidr_block                                     = "172.31.224.0/24"
    customer_owned_ipv4_pool                       = null
    enable_dns64                                   = false
    enable_lni_at_device_index                     = 0
    enable_resource_name_dns_a_record_on_launch    = false
    enable_resource_name_dns_aaaa_record_on_launch = false
    id                                             = "subnet-0dd952f8e63b3fe1e"
    ipv6_cidr_block                                = null
    ipv6_cidr_block_association_id                 = null
    ipv6_native                                    = false
    map_customer_owned_ip_on_launch                = false
    map_public_ip_on_launch                        = false
    outpost_arn                                    = null
    owner_id                                       = "789925699467"
    private_dns_hostname_type_on_launch            = "ip-name"
    region                                         = "us-east-1"
```

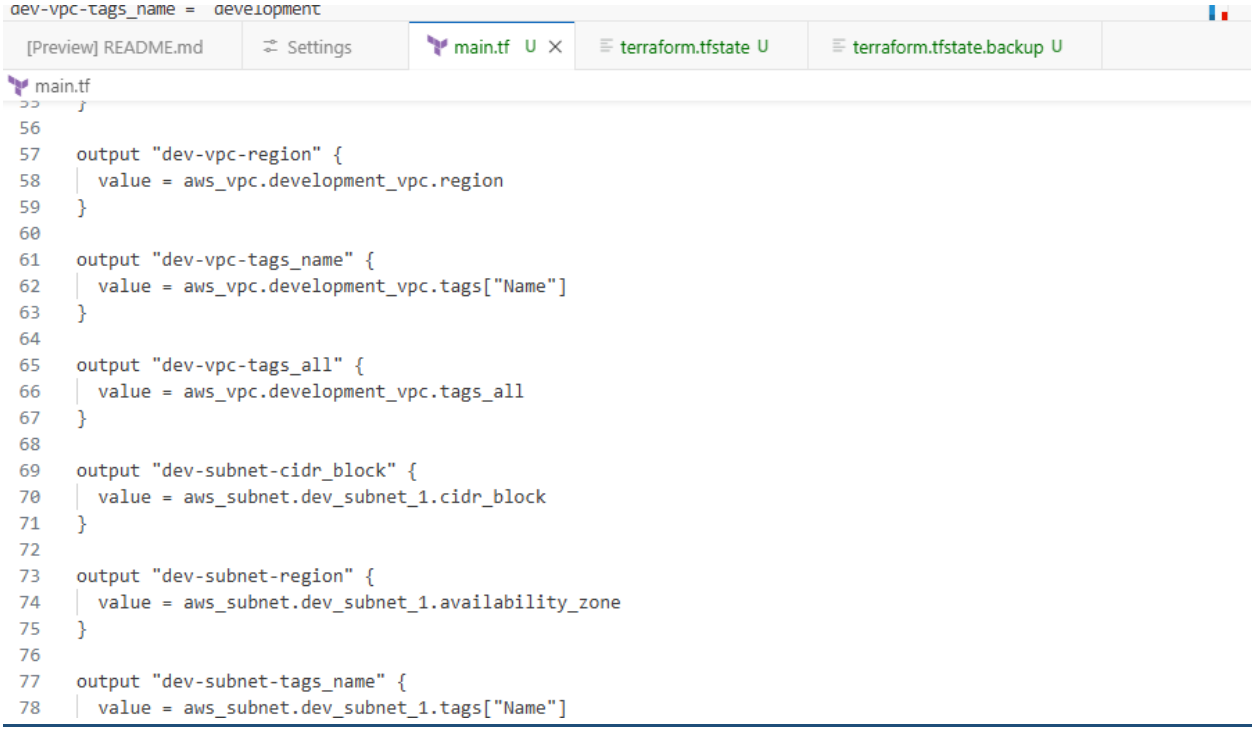
- Shows **all attributes** of that resource from the state file.

📸 Screenshot → **task5_terraform_state_show_resource.png**

Task 6 — Terraform Outputs & Attributes Reporting

What Terraform Outputs Are

In your main.tf, you defined outputs like this:



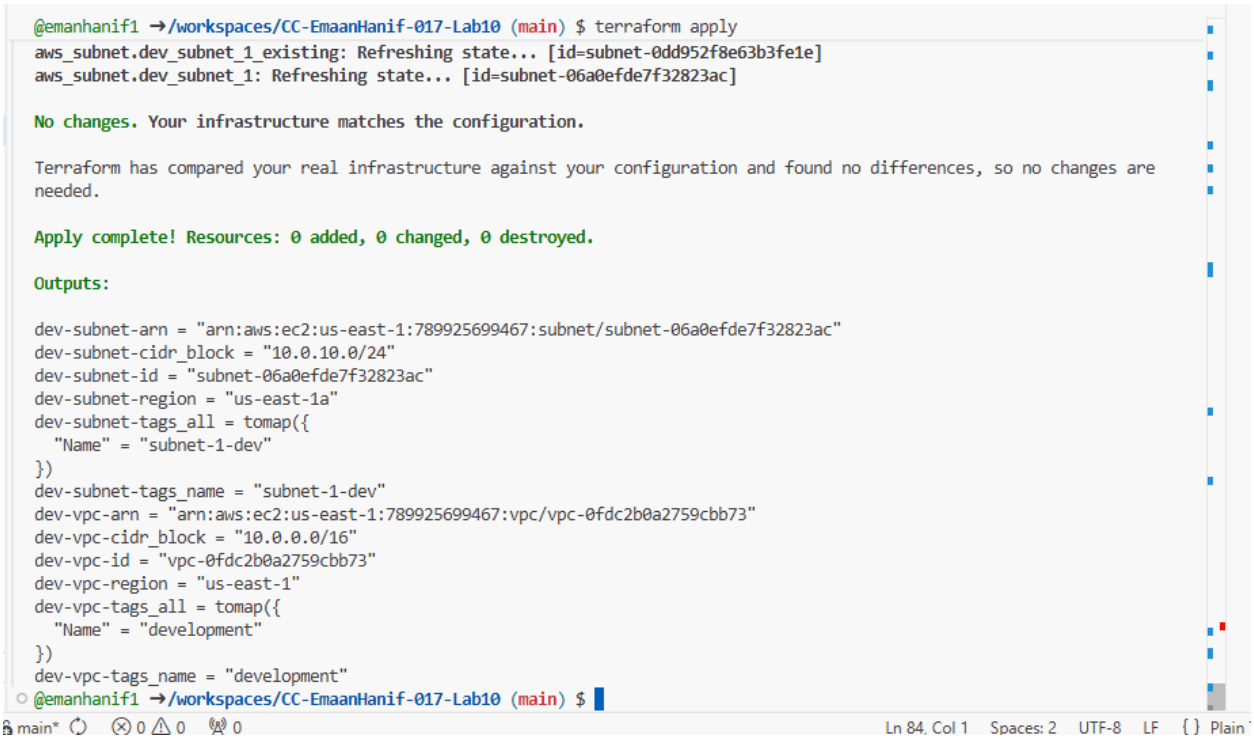
- aws_vpc.development_vpc.id → Terraform shows the **VPC ID**, e.g., vpc-0fdc2b0a2759cbb73.
- aws_vpc.development_vpc.cidr_block → The VPC IP range, e.g., 10.0.0.0/16.
- aws_subnet.dev_subnet_1.tags["Name"] → The Name tag of the subnet, e.g., "subnet-1-dev".
- tags_all → Shows **all tags**, even if you added more later.

Apply the configuration

Run:

terraform apply

✅ Terraform will create/update resources (if needed) and print all the outputs at the end.



- Each line corresponds to **one output you defined**.
- For tags_all, Terraform prints it in **HCL map format** { "Name" = "subnet-1-dev" }.
- region for subnet shows **availability zone**, not just the AWS region (us-east-1a vs us-east-1).

Apply the configuration

Run:

terraform apply -auto-approve

Terraform will create/update resources (if needed) and print all the outputs at the end.

```
@emanhanif1 ➜/workspaces/CC-EmaanHanif-017-Lab10 (main) $ terraform apply -auto-approve
  + dev-vpc-id            = "vpc-0fdc2b0a2759cbb73"
  + dev-vpc-region        = "us-east-1"
  + dev-vpc-tags_all      = {
      + Name = "development"
    }
  + dev-vpc-tags_name     = "development"

You can apply this plan to save these new output values to the Terraform state, without changing any real infrastructure.

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

Outputs:

dev-subnet-arn = "arn:aws:ec2:us-east-1:789925699467:subnet/subnet-06a0efde7f32823ac"
dev-subnet-cidr_block = "10.0.10.0/24"
dev-subnet-id = "subnet-06a0efde7f32823ac"
dev-subnet-region = "us-east-1a"
dev-subnet-tags_all = tomap({
  "Name" = "subnet-1-dev"
})
dev-subnet-tags_name = "subnet-1-dev"
dev-vpc-arn = "arn:aws:ec2:us-east-1:789925699467:vpc/vpc-0fdc2b0a2759cbb73"
dev-vpc-cidr_block = "10.0.0.0/16"
dev-vpc-id = "vpc-0fdc2b0a2759cbb73"
dev-vpc-region = "us-east-1"
dev-vpc-tags_all = tomap({
  "Name" = "development"
})
dev-vpc-tags_name = "development"
```

Cleanup — Delete Resources & State Verification

Destroy All Resources

Run:

terraform destroy

- Terraform will list all resources to destroy (VPCs, subnets, etc.).
- Confirm by typing yes when prompted.
- **Screenshot:** Save the terminal output as cleanup_destroy_resources.png.

```
@emanhanif1 →/workspaces/CC-EmaanHanif-017-Lab10 (main) $ terraform destroy
  - dev-subnet-region     = "us-east-1a" -> null
  - dev-subnet-tags_all   = {
      - Name = "subnet-1-dev"
    } -> null
  - dev-subnet-tags_name  = "subnet-1-dev" -> null
  - dev-vpc-arn           = "arn:aws:ec2:us-east-1:789925699467:vpc/vpc-0fdc2b0a2759cbb73" -> null
  - dev-vpc-cidr_block    = "10.0.0.0/16" -> null
  - dev-vpc-id            = "vpc-0fdc2b0a2759cbb73" -> null
  - dev-vpc-region        = "us-east-1" -> null
  - dev-vpc-tags_all      = {
      - Name = "development"
    } -> null
  - dev-vpc-tags_name     = "development" -> null

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

aws_subnet.dev_subnet_1: Destroying... [id=subnet-06a0efde7f32823ac]
aws_subnet.dev_subnet_1_existing: Destroying... [id=subnet-0dd952f8e63b3fe1e]
aws_subnet.dev_subnet_1_existing: Destruction complete after 1s
aws_subnet.dev_subnet_1: Destruction complete after 1s
aws_vpc.development_vpc: Destroying... [id=vpc-0fdc2b0a2759cbb73]
aws_vpc.development_vpc: Destruction complete after 1s

Destroy complete! Resources: 3 destroyed.
@emanhanif1 →/workspaces/CC-EmaanHanif-017-Lab10 (main) $
```

After completion, Terraform should report:

Destroy complete! Resources: X destroyed.

Inspect Terraform State Files

Check the content of your state files after destruction:

cat terraform.tfstate
cat terraform.tfstate.backup

- terraform.tfstate → should now be **empty** (no resources).
- terraform.tfstate.backup → will still have a **backup of previous resources** (from before destroy).
- **Screenshot:** Save output as cleanup_state_files.png.

This shows Terraform keeps a backup even after destruction.

```
@emanhanif1 ➜/workspaces/CC-EmaanHanif-017-Lab10 (main) $ cat terraform.tfstate
{
  "version": 4,
  "terraform_version": "1.14.2",
  "serial": 40,
  "lineage": "58757615-a9b6-593a-1461-3fc47c6790f0",
  "outputs": {},
  "resources": [],
  "check_results": null
}
```

- This file is **empty of resources**, meaning Terraform currently sees **no managed infrastructure**.
- After terraform destroy, all your VPCs and subnets are gone from the active state.
- outputs is also empty because Terraform removed the resources, so there's nothing to output.

```
@emanhanif1 ➜/workspaces/CC-EmaanHanif-017-Lab10 (main) $ cat terraform.tfstate.backup
            "ipv6_cidr_block": "",
            "ipv6_cidr_block_network_border_group": "",
            "ipv6_ipam_pool_id": "",
            "ipv6_netmask_length": 0,
            "main_route_table_id": "rtb-09832ea05d48e3b3f",
            "owner_id": "789925699467",
            "region": "us-east-1",
            "tags": {
              "Name": "development"
            },
            "tags_all": {
              "Name": "development"
            }
          },
          "sensitive_attributes": [],
          "identity_schema_version": 0,
          "identity": {
            "account_id": "789925699467",
            "id": "vpc-0fdc2b0a2759cbb73",
            "region": "us-east-1"
          },
          "private": "eyJzY2hlbWFfdmVyc2lvbiI6IjEifQ=="
        }
      ]
    }
  ],
  "check_results": null
}
@emanhanif1 ➜/workspaces/CC-EmaanHanif-017-Lab10 (main) $
```

- The **backup file still contains all previous resources** and their output values.

  It includes:

- dev-vpc-id, dev-subnet-id, dev-vpc-arn, dev-subnet-arn, etc.
- Full resource attributes (CIDR block, region, tags) from before you destroyed them.
- This allows Terraform (or you) to **recover or reference** the previous state if needed.

THE END