

INFRASTRUCTURE AUTOMATION WITH TERRAFORM

DevOps infrastructure or Infrastructure Automation under DevOps refers to a concept that revolves around the idea of managing infrastructure with the help of code. This is done with the help of certain tools or programs which can help to carry out the tasks automatically!



What is Terraform?

Terraform is an infrastructure as code tool that lets you build, change, and version cloud and on-prem resources safely and efficiently.

Terraform is one of the most popular **Infrastructure-as-code (IaC) tool**, used by DevOps teams to automate infrastructure tasks. It is used to automate the provisioning of your cloud resources. Terraform is an open-source, cloud-agnostic provisioning tool developed by HashiCorp and written in GO language.

Why Terraform for DevOps?

You can have the independence to create, delete or update the resources, terraform will ensure that there is no change in the state of the infrastructure. Another great advantage of using Terraform with devops is that it is written in a very easy-to-understand language. Hashicorp Configuration Language (HCL).

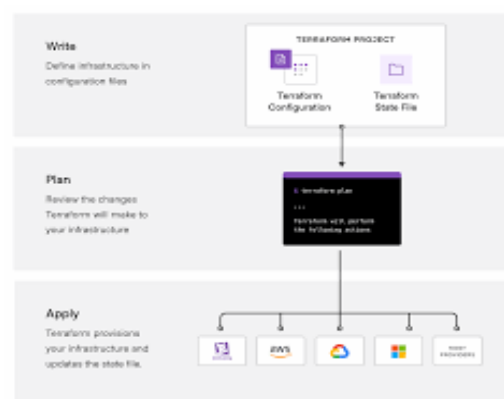
INSTALLATION OF TERRAFORM:

1. Go to the official Terraform website at <https://www.terraform.io/downloads.html>
2. Under the "Terraform" section, find the version of Terraform that you want to download for Windows.
3. Click on the download link for Windows. This will download the Terraform binary in a ZIP file format.
4. Extract the ZIP file to a directory on your Windows machine. You can use any directory of your choice.
5. Add the Terraform binary to your system's PATH environment variable. This will enable you to run Terraform from any directory in your command prompt or PowerShell.
6. To add Terraform to your system's PATH variable, open the Start menu and search for "Environment Variables". Click on the "Edit the system environment variables" option.
7. In the System Properties window, click on the "Environment Variables" button.
8. In the Environment Variables window, under the "System Variables" section, find the "Path" variable and click on the "Edit" button.
9. In the Edit Environment Variable window, click on the "New" button and add the directory path where you extracted the Terraform binary file.

10. Click "OK" to close all the windows.
11. Open a new command prompt or PowerShell window, and type "terraform" to verify that the installation was successful. If you see a list of Terraform commands, then you have successfully installed Terraform on your Windows machine.

Steps for the Terraform Creation:

1. What is Terraform Cloud - Intro and Sign Up.
2. Log in to Terraform Cloud from the CLI.
3. Create a Credentials Variable Set.
4. Create a Workspace.
5. Create Infrastructure.
6. Change Infrastructure.
7. Use VCS-Driven Workflow.
8. Destroy Resources and Workspaces.



Terraform Lifecycle:

Terraform lifecycle consists of – **init**, **plan**, **apply**, and **destroy**.



1. **Terraform init** initializes the (local) Terraform environment. Usually executed only once per session.
2. **Terraform plan** compares the Terraform state with the as-is state in the cloud, build and display an execution plan. This does not change the deployment (read-only).
3. **Terraform apply** executes the plan. This potentially changes the deployment.

4. **Terraform destroy** deletes all resources that are governed by this specific terraform environment.

Terraform Core Concept

1. Variables: Terraform has input and output variables, it is a key-value pair. Input variables are used as parameters to input values at run time to customize our deployments. Output variables are return values of a terraform module that can be used by other configurations.

2. Provider: Terraform users provision their infrastructure on the major cloud providers such as AWS, Azure, OCI, and others. A *provider* is a plugin that interacts with the various APIs required to create, update, and delete various resources.

3. Module: Any set of Terraform configuration files in a folder is a module. Every Terraform configuration has at least one module, known as its **root module**.

4. State: Terraform records information about what infrastructure is created in a Terraform *state* file. With the state file, Terraform is able to find the resources it created previously, supposed to manage and update them accordingly.

5. Resources: Cloud Providers provides various services in their offerings, they are referenced as Resources in Terraform. Terraform resources can be anything from compute instances, virtual networks to higher-level components such as DNS records. Each resource has its own attributes to define that resource.

6. Data Source: Data source performs a read-only operation. It allows data to be fetched or computed from resources/entities that are not defined or managed by Terraform or the current Terraform configuration.

7. Plan: It is one of the stages in the Terraform lifecycle where it determines what needs to be created, updated, or destroyed to move from the real/current state of the infrastructure to the desired state.

8. Apply: It is one of the stages in the Terraform lifecycle where it applies the changes real/current state of the infrastructure in order to achieve the desired state.

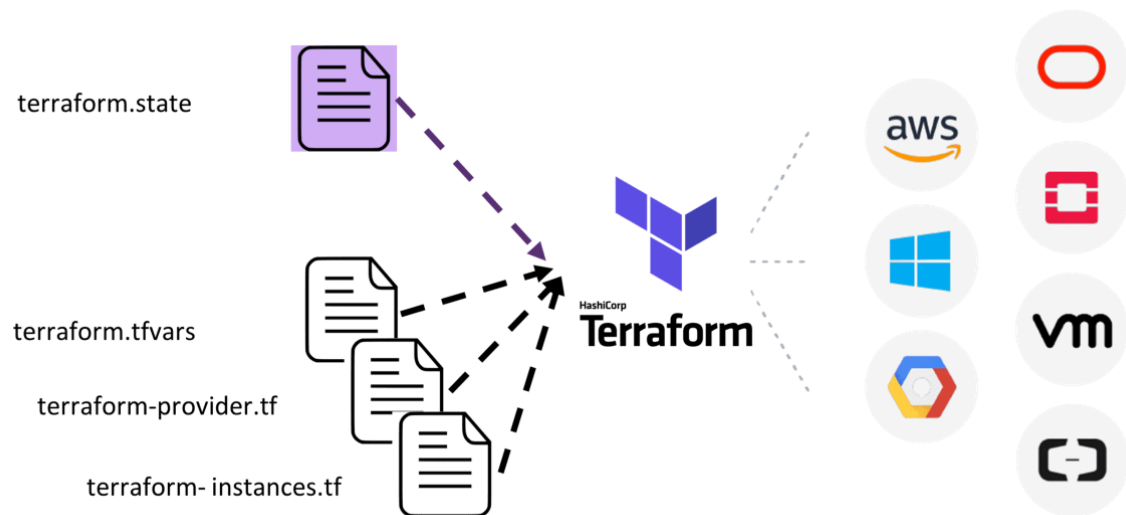
Terraform Installation:

Before you start working, make sure you have Terraform installed on your machine, it can be installed on any OS, say Windows, macOS, Linux, or others. Terraform installation is an easy process and can be done in a few minutes.



Terraform Configuration Files

Configuration files are a set of files used to describe infrastructure in Terraform and have the file extensions **.tf** and **.tf.json**. Terraform uses a declarative model for defining infrastructure. Configuration files let you write a configuration that declares your desired state. Configuration files are made up of resources with settings and values representing the desired state of your infrastructure.



A Terraform configuration is made up of one or more files in a directory, provider binaries, plan files, and state files once Terraform has run the configuration.

- 1. Configuration file (*.tf files):** Here we declare the provider and resources to be deployed along with the type of resource and all resources specific settings
- 2. Variable declaration file (variables.tf or variables.tf.json):** Here we declare the input variables required to provision resources
- 3. Variable definition files (terraform.tfvars):** Here we assign values to the input variables
- 4. State file (terraform.tfstate):** a state file is created once after Terraform is run. It stores state about our managed infrastructure.

Getting started using Terraform

To get started building infrastructure resources using Terraform, there are few things that you should take care of. The general steps to deploy a resource(s) in the cloud are:

1. Set up a Cloud Account on any cloud provider ([AWS](#), [Azure](#), [OCI](#))
2. Install Terraform
3. Add a provider – AWS, Azure, OCI, GCP, or others
4. Write configuration files
5. Initialize Terraform Providers
6. PLAN (DRY RUN) using terraform plan

7. APPLY (Create a Resource) using terraform apply
8. DESTROY (Delete a Resource) using terraform destroy

EXAMPLE: creates an EC2 instance

```
provider "aws" {  
    region = "us-west-1"  
}  
  
resource "aws_instance" "myec2" {  
    ami = "ami-12345qwerty"  
    instance_type = "t2.micro"  
}
```

HOW TO CREATE NEW FILE BY USING TERRAFORM BY USING COMMAND PROMPT:

1. Create a new folder on Desktop. Open the folder and go to the command prompt path.
2. Enter the command "code ." to go to the vs code.
3. To create new folders using Terraform through the command prompt, we can use `local_file` resource.
4. Create a new Terraform file(e.g. `main.tf`) and add the following code.

```
resource "local_file" "new_folder" {  
    filename = "${path.module}/new_folder"  
    content = ""  
}
```
5. In the same directory where we have created the Terraform file, open the command prompt and navigate to the directory where we want to create the new folder.
6. Run the following commands:

```
terraform init  
terraform apply
```
7. Terraform will prompt to confirm the creation of the new folder. Type `yes` and hit Enter.
8. Then Terraform will create a new folder with the name "new_folder" in the directory.

HOW TO CREATE EC2 INSTANCE BY USING TERRAFORM:(AWS)

Go to command prompt and type the following commands

1. AWS configure
 - Access id: Enter user access id
 - Access Key: Enter access Key
 - Region name: Enter default region name

- Output Format: Enter default output Format
2. Create a notepad with Terraform file(eg: filename.tf) with two filenames

File1 code:

```
provider"aws"{
  access_key="user access id"
  secret_key="user secret key"
  region="us-west-1"
}
```

File2 code:

```
resource "aws_instance" "myFirstInstance" {
  ami="enter ami-id"
  count=1
  key_name = "Key-pair name"
  instance_type = "t2.micro"
  security_groups= [ "security_jenkins_port"]
  tags={
    Name = "jenkins_instance"
  }
}

resource "aws_security_group" "security_jenkins_port" {
  name= "security_jenkins_port"
  description = "security group for jenkins"
  ingress{
    from_port= 8080
    to_port = 8080
    protocol= "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port = 22
    to_port = 22
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress{
    from_port = 0
    to_port= 65535
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  tags={
    Name = "security_jenkins_port"
  }
}
```

3. Based on user interest increase the count in file2 code.
4. Now initialize the Terraform command is
Code: terraform init
5. Now apply the Terraform command is
Code: terraform apply
6. Then the instances will be created

```
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

aws_instance.myFirstInstance[1]: Creating...
aws_instance.myFirstInstance[2]: Creating...
aws_instance.myFirstInstance[0]: Creating...
aws_security_group.security_jenkins_port: Creation complete after 8s [id=sg-090f626f22626a364]
aws_instance.myFirstInstance[2]: Still creating... [10s elapsed]
aws_instance.myFirstInstance[0]: Still creating... [10s elapsed]
aws_instance.myFirstInstance[1]: Still creating... [10s elapsed]
aws_instance.myFirstInstance[1]: Still creating... [20s elapsed]
aws_instance.myFirstInstance[2]: Still creating... [20s elapsed]
aws_instance.myFirstInstance[0]: Still creating... [20s elapsed]
aws_instance.myFirstInstance[0]: Still creating... [30s elapsed]
aws_instance.myFirstInstance[2]: Still creating... [30s elapsed]
aws_instance.myFirstInstance[1]: Still creating... [30s elapsed]
aws_instance.myFirstInstance[0]: Still creating... [40s elapsed]
aws_instance.myFirstInstance[2]: Still creating... [40s elapsed]
aws_instance.myFirstInstance[1]: Still creating... [40s elapsed]
aws_instance.myFirstInstance[2]: Creation complete after 42s [id=i-0ad2462277d8ee9c3]
aws_instance.myFirstInstance[0]: Creation complete after 42s [id=i-028a4e2141490e216]
aws_instance.myFirstInstance[1]: Creation complete after 43s [id=i-07493e802d4321e0c]

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

7. Now we can verify through AWS console user Account.

HOW TO ADD S3 BUCKET TO AWS THROUGH TERRAFORM:

Go to command prompt and type the following commands

1. AWS configure
 - Access id :Enter user access id
 - Access Key: Enter access Key
 - Region name: Enter default region name
 - Output Format: Enter default output Format
2. Create a Terraform file with notepad(eg: filename.tf) with two filenames

File1 code:

```
provider"aws"{
  access_key="user access id"
  secret_key="user secret key"
  region="us-east-1"
}
```

File2 code:

```
resource "aws_s3_bucket" "my_bucket" {
  bucket = "bucket-name"
  acl    = "private"

  tags = {
    Environment = "Dev"
  }
}
```

3. After Executing command successfully we check through AWS console of user account of S3 bucket.

TO ADD OBJECT TO BUCKET:

4. It is continuous of the above S3. Create new file

File3 code:

```
resource "aws_s3_bucket_object" "object" {
  bucket = "bucket name"
  key    = "filename"
  source = "source\\to\\filename"
  etag   = filemd5("source\\to\\filename ")
}
```

5. After execution of 4th step successfully now go to user AWS console and we Can see object is added to the bucket.

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

Warning: Argument is deprecated
  with aws_s3_bucket.my_bucket,
  on b.tf line 3, in resource "aws_s3_bucket" "my_bucket":
   3:   acl    = "private"

Use the aws_s3_bucket_acl resource instead
(and 5 more similar warnings elsewhere)

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_object.object: Creating...
aws_s3_bucket_object.object: Creation complete after 3s [id=ghost]

Warning: Argument is deprecated
  with aws_s3_bucket_object.object,
  on c.tf line 2, in resource "aws_s3_bucket_object" "object":
   2:   bucket = "my-terra---form-bucket"

Use the aws_s3_object resource instead
(and one more similar warning elsewhere)

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


HOW TO CREATE VPC USING TERRAFORM:

1. Go to command prompt and type the following commands
2. Create a Terraform file with notepad which code consists of all the vpc formation resources.

File code:

```
# Declare provider and region
provider "aws" {
  region = "us-west-2"
}

# Declare VPC
resource "aws_vpc" "example_vpc" {
  cidr_block = "10.0.0.0/16"
  enable_dns_support = true
  enable_dns_hostnames = true
  tags = {
    Name = "example_vpc"
  }
}

# Declare Internet Gateway
resource "aws_internet_gateway" "example_igw" {
  vpc_id = aws_vpc.example_vpc.id
  tags = {
    Name = "example_igw"
  }
}

# Declare Route Table
resource "aws_route_table" "example_route_table" {
  vpc_id = aws_vpc.example_vpc.id
  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.example_igw.id
  }
  tags = {
    Name = "example_route_table"
  }
}

# Declare Subnet
resource "aws_subnet" "example_subnet" {
  vpc_id = aws_vpc.example_vpc.id
  cidr_block = "10.0.1.0/24"
  availability_zone = "us-west-1a"
  tags = {
    Name = "example_subnet"
  }
}
```

```

# Declare Subnet Association
resource "aws_route_table_association"
"example_route_table_association" {
    subnet_id = aws_subnet.example_subnet.id
    route_table_id = aws_route_table.example_route_table.id
}

```

3. Save the file and run the commands
 terraform init
 terraform apply
4. After Executing command successfully we check through AWS console of user account of VPC.

```

}
Plan: 3 to add, 0 to change, 0 to destroy.

Warning: Argument is deprecated

    with aws_s3_bucket.my_bucket,
    on b.tf line 3, in resource "aws_s3_bucket" "my_bucket":
     3:   acl   = "private"

Use the aws_s3_bucket_acl resource instead

(and 5 more similar warnings elsewhere)

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_object.object: Creating...
aws_subnet.example_subnet: Creating...
aws_subnet.example_subnet: Creation complete after 4s [id=subnet-0c830cc31dc69c68b]
aws_s3_bucket_object.object: Creation complete after 4s [id=ghost]
aws_route_table_association.example_route_table_association: Creating...
aws_route_table_association.example_route_table_association: Creation complete after 2s [id=rtbassoc-05ba560510d120c73]

```

5. To delete VPC use command : terraform destroy

HOW TO CREATE EC2 INSTANCE BY USING TERRAFORM:(AZURE)

- 1.Go to command prompt and create a file and write the following code

FILE CODE:

```

provider "azurerm" {
    features {}
}

resource "azurerm_resource_group" "example" {
    name     = "my-resource-group"
    location = "eastus"
}

resource "azurerm_virtual_network" "example" {
    name                = "my-vnet"
    address_space       = ["10.0.0.0/16"]
    location             = azurerm_resource_group.example.location
    resource_group_name = azurerm_resource_group.example.name
}

```

```
resource "azurerm_subnet" "example" {
  name = "my-subnet"
  resource_group_name = azurerm_resource_group.example.name
  virtual_network_name = azurerm_virtual_network.example.name
  address_prefixes    = ["10.0.1.0/24"]
}
```

```
resource "azurerm_storage_account" "example" {
  name = "mystorageaccount4910548"
  resource_group_name = azurerm_resource_group.example.name
  location = azurerm_resource_group.example.location
  account_tier = "Standard"
  account_replication_type = "GRS"
}
```

```
resource "azurerm_network_interface" "example" {
  name = "my-nic"
  location = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name
  ip_configuration {
    name = "my-ip-config"
    subnet_id = azurerm_subnet.example.id
    private_ip_address_allocation = "Dynamic"
  }
}
```

```
resource "azurerm_virtual_machine" "example" {
  name = "my-vm"
  location = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name
  network_interface_ids = [azurerm_network_interface.example.id]
  vm_size = "Standard_B2s"
```

```
  storage_os_disk {
    name = "my-os-disk"
    caching = "ReadWrite"
    create_option = "FromImage"
    managed_disk_type = "Standard_LRS"
  }
```

```
  storage_image_reference {
    publisher = "Canonical"
    offer     = "UbuntuServer"
    sku       = "16.04-LTS"
    version   = "latest"
  }
```

```
  os_profile {
    computer_name = "vm name"
    admin_username = "username"
```

```

    admin_password = "password"
  }
  os_profile_linux_config {
    disable_password_authentication = false
  }
}

```

```

C:\Windows\System32\cmd.exe
+ secondary_connection_string = (sensitive value)
+ secondary_dfs_endpoint      = (known after apply)
+ secondary_dfs_host          = (known after apply)
+ secondary_file_endpoint     = (known after apply)
+ secondary_file_host         = (known after apply)
+ secondary_location          = (known after apply)
+ secondary_queue_endpoint    = (known after apply)
+ secondary_queue_host        = (known after apply)
+ secondary_table_endpoint    = (known after apply)
+ secondary_table_host        = (known after apply)
+ secondary_web_endpoint      = (known after apply)
+ secondary_web_host          = (known after apply)
+ sftp_enabled                = false
+ shared_access_key_enabled    = true
+ table_encryption_key_type    = "Service"
}

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

azurerm_storage_account.example: Creating...
azurerm_storage_account.example: Still creating... [10s elapsed]
azurerm_storage_account.example: Still creating... [20s elapsed]
azurerm_storage_account.example: Still creating... [30s elapsed]
azurerm_storage_account.example: Still creating... [40s elapsed]
azurerm_storage_account.example: Still creating... [50s elapsed]
azurerm_storage_account.example: Creation complete after 50s [id=/subscriptions/3eb93084-cdfe-4fd9-ba10-7ade05d985b1/resourceGroups/my-resource-group/providers/Microsoft.Storage/storageAccounts/mystorageaccount4910548]

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

```

HOW TO CREATE MULTIPLE EC2 INSTANCES WITH DIFFERENT CONFIGURATIONS:

1. First open the command prompt and create the file1 and file2.

File1 Code:

```

provider"aws"{
  access_key="AKIAQWJ27HIMWVF4Y6WI"
  secret_key="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"
  region="us-west-2"
}

```

File2 Code: (with certain number)

```

variable "instances" {
  type = map(object({
    ami = string
    instance_type = string
    key_name = string
  }))
  default = {
    web1 = {
      ami = "ami-0c55b159cbfafa1f0"
      instance_type = "t2.micro"
      key_name = "my_key"
    },
    web2 = {
      ami = "ami-0c55b159cbfafa1f0"

```

```

        instance_type = "t2.small"
        key_name      = "my_key"
    },
    db1 = {
        ami          = "ami-0c55b159cbfafa1f0"
        instance_type = "t2.medium"
        key_name      = "my_key"
    }
}
}
}

```

```

resource "aws_instance" "instance" {
  for_each = var.instances

  ami          = each.value.ami
  instance_type = each.value.instance_type
  key_name      = each.value.key_name

  tags = {
    Name = each.key
  }
}

```

2. If user required more number of count with simple code the use file3 code
Instead file2 code.

File3 Code: (with n number of count)

```

resource "aws_instance" "instance" {
  count = 40
  ami = count.index < 20 ? "ami-0c55b159cbfafa1f0" : "ami-0123456789abcdef"
  instance_type = count.index < 20 ? "t2.micro" : "t2.small"
  key_name      = count.index < 20 ? "my_key1" : "my_key2"

  tags = {
    Name = "instance-${count.index}"
  }
}

```

3. Hence user get desired number of instances with using file3 code can verify through the user AWS console page.

WEB HOSTING USING TERRAFORM THROUGH JENKINS WITH GITHUB:

1. First create two files with the HCL code with help VS or Notepad.

File1 code:

```
provider"aws"{
  access_key="access key"
  secret_key="secret key "
  region="us-west-2"
}
```

File2 code:

```
resource "aws_instance" "myFirstInstance" {
  ami="ami-0efa651876de2a5ce"
  count=1
  key_name = "ubuntu-key"
  instance_type = "t2.micro"
  security_groups= [ "security_jenkins_port"]
  tags={
    Name = "jenkins_instance"
  }
}

User_data=<<-EOF(End Of File)
#!/bin/bash
yum update -y
yum install -y httpd
yum install unzip
wget https://www.tooplate.com/zip-templates/2113_earth.zip
unzip 2113_earth.zip
cp -r 2113_earth/* /var/www/html/
systemctl start httpd
systemctl enable httpd
EOF

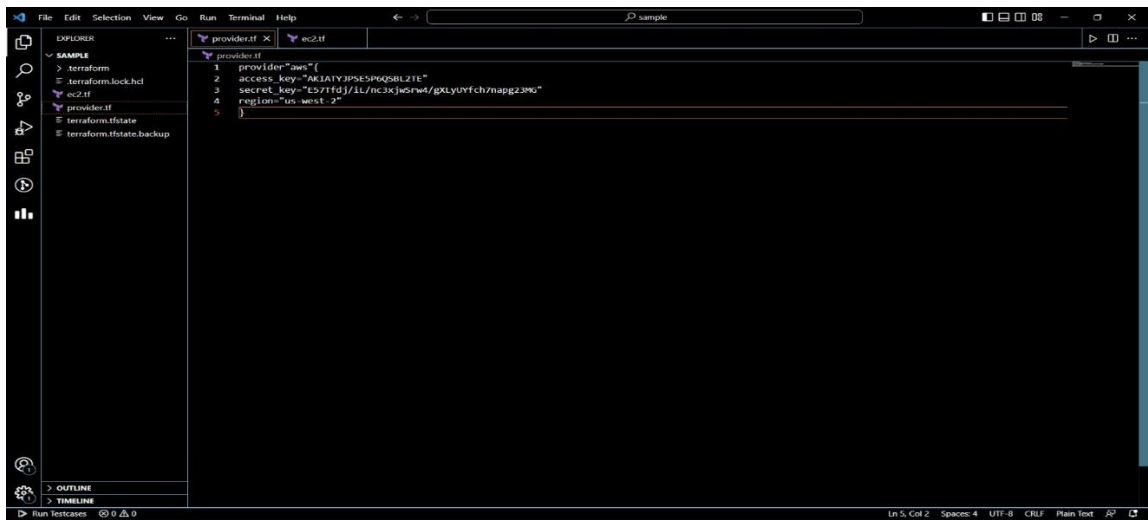
resource "aws_security_group" "security_jenkins_port" {
  name= "security_jenkins_port"
  description = "security group for jenkins"
  ingress{
    from_port= 8080
    to_port = 8080
    protocol= "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port = 22
    to_port = 22
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

```

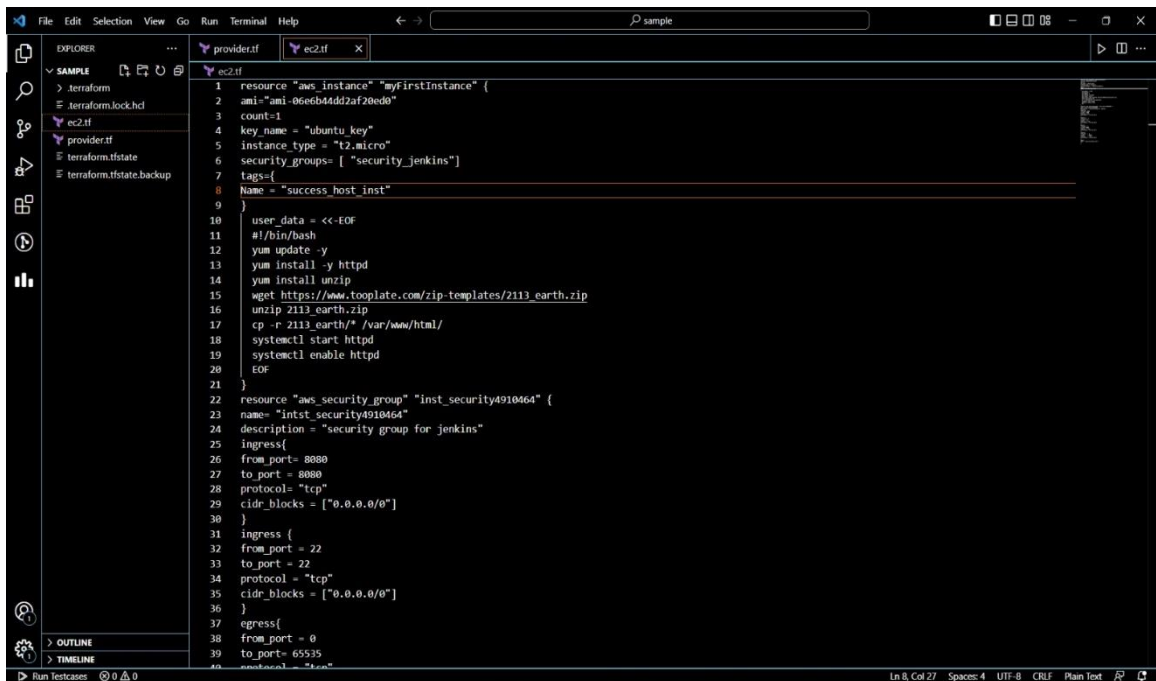
egress{
  from_port = 0
  to_port= 65535
  protocol = "tcp"
  cidr_blocks = ["0.0.0.0/0"]
}
tags={
  Name = "security_jenkins_port"
}
}

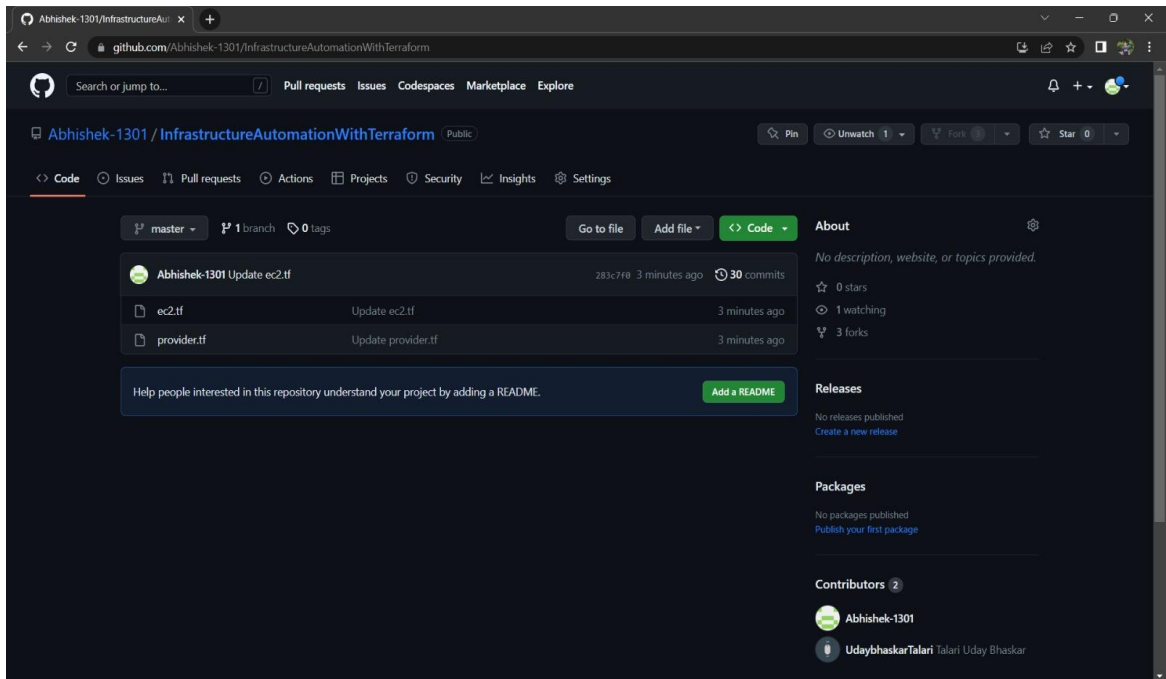
```

2. Now upload the files to the Git Hub Repository

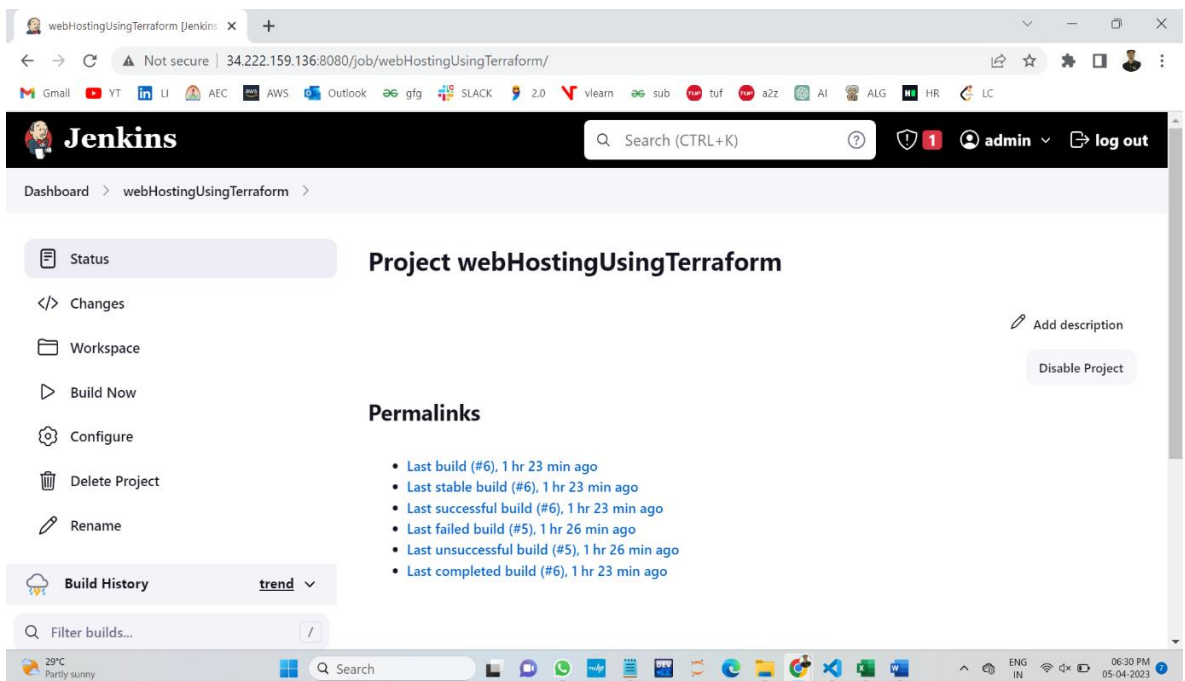


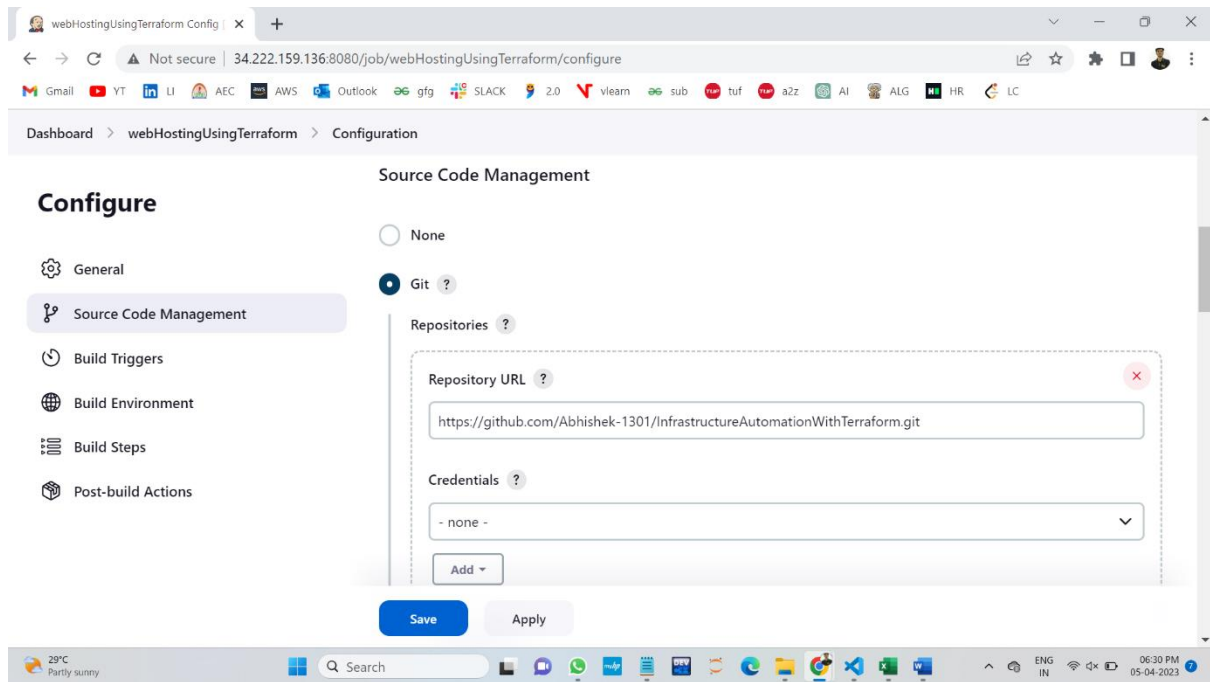
Uploaded files



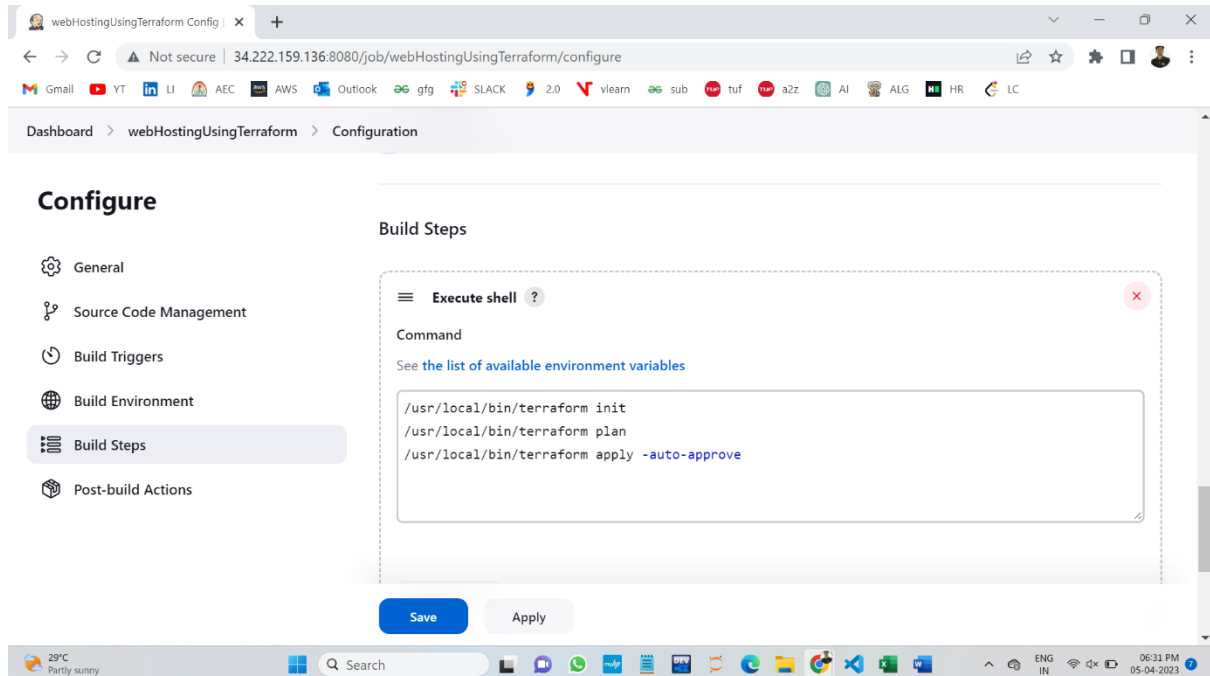


3. Go to Jenkins and build the job and give the repository URL to it and configure it.

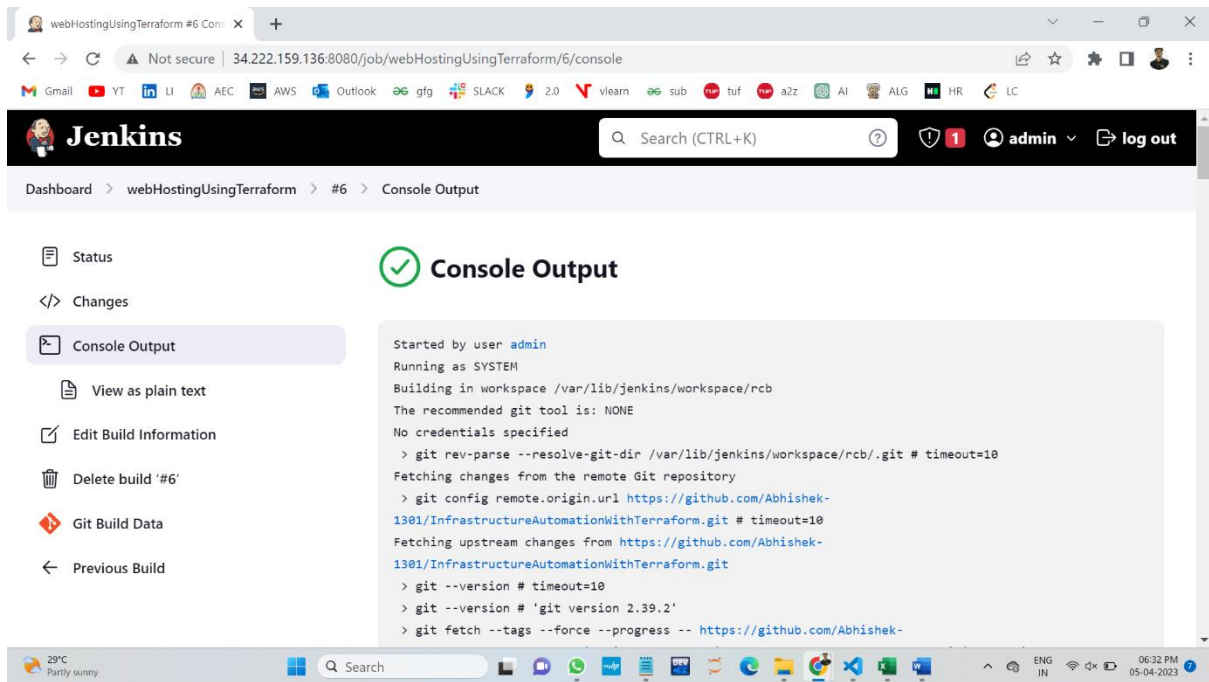




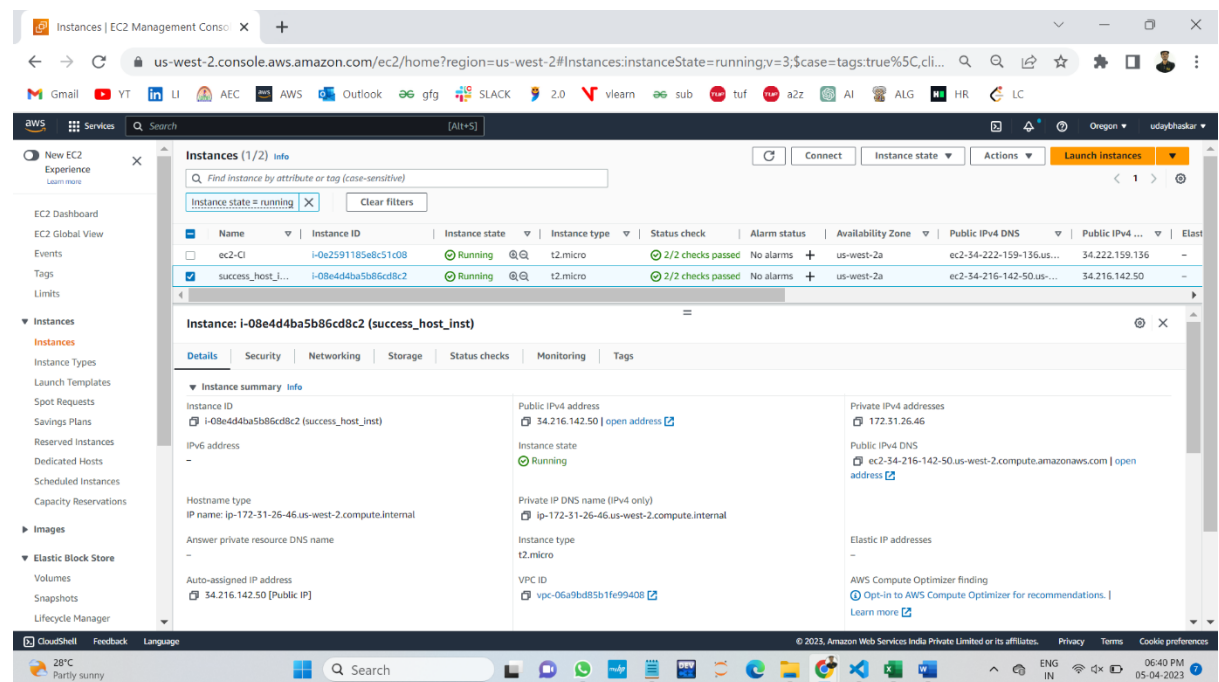
4. After go to Build Steps and enter the following command with Linux path
Source/Terraform path init
Source/Terraform path plan
Source/Terraform path apply -auto -approve



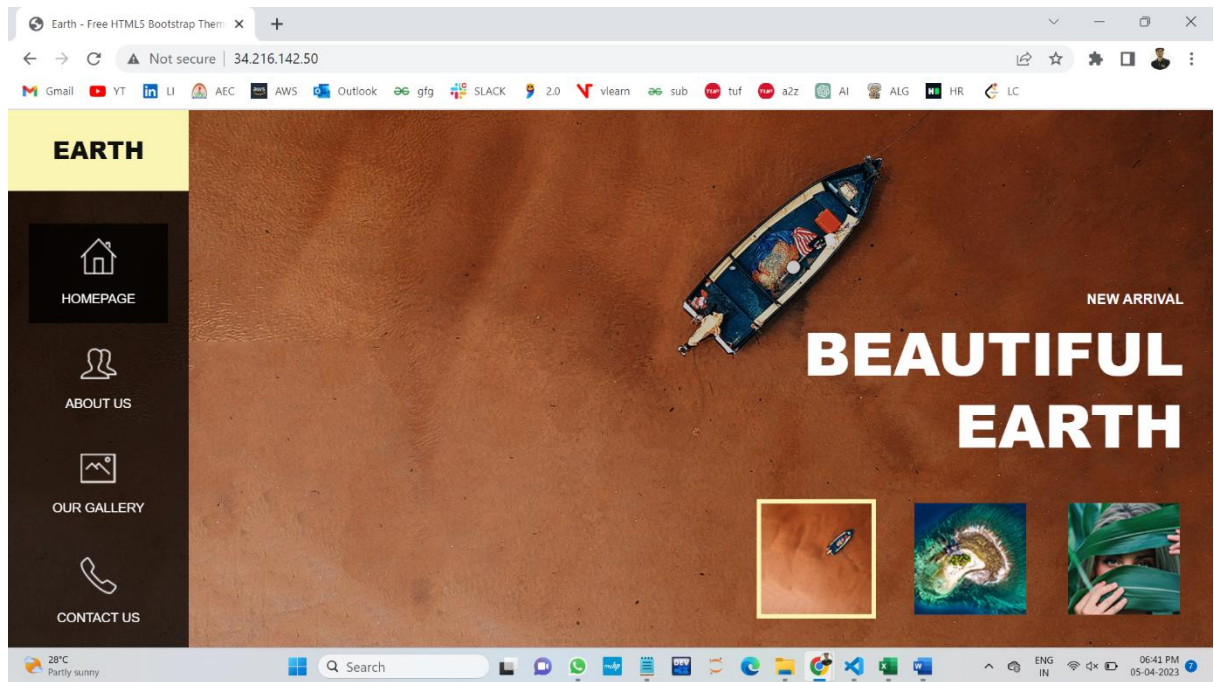
5. After configuring, Build the job in it and run it.



6. After Executed Successfully with no error then go to AWS console Management And we can see new EC2 Instances created in AWS Management Console

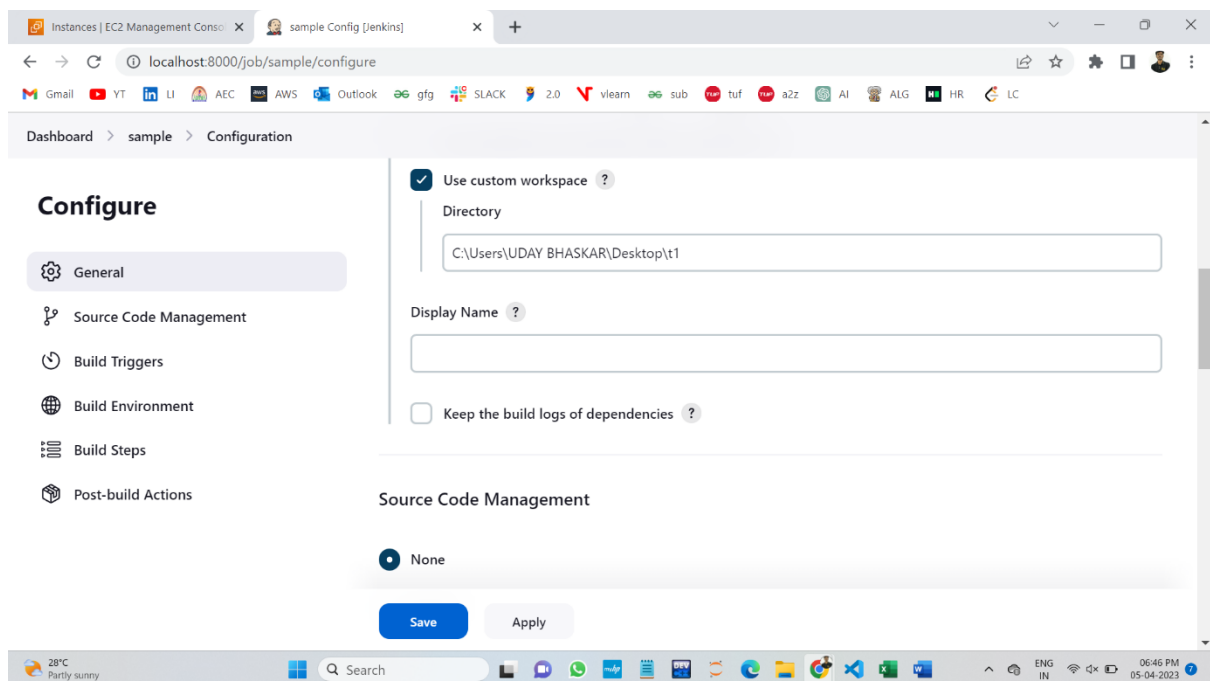


7. Now copy the public ip of that instances and paste it in the browser and we can See the html web page.

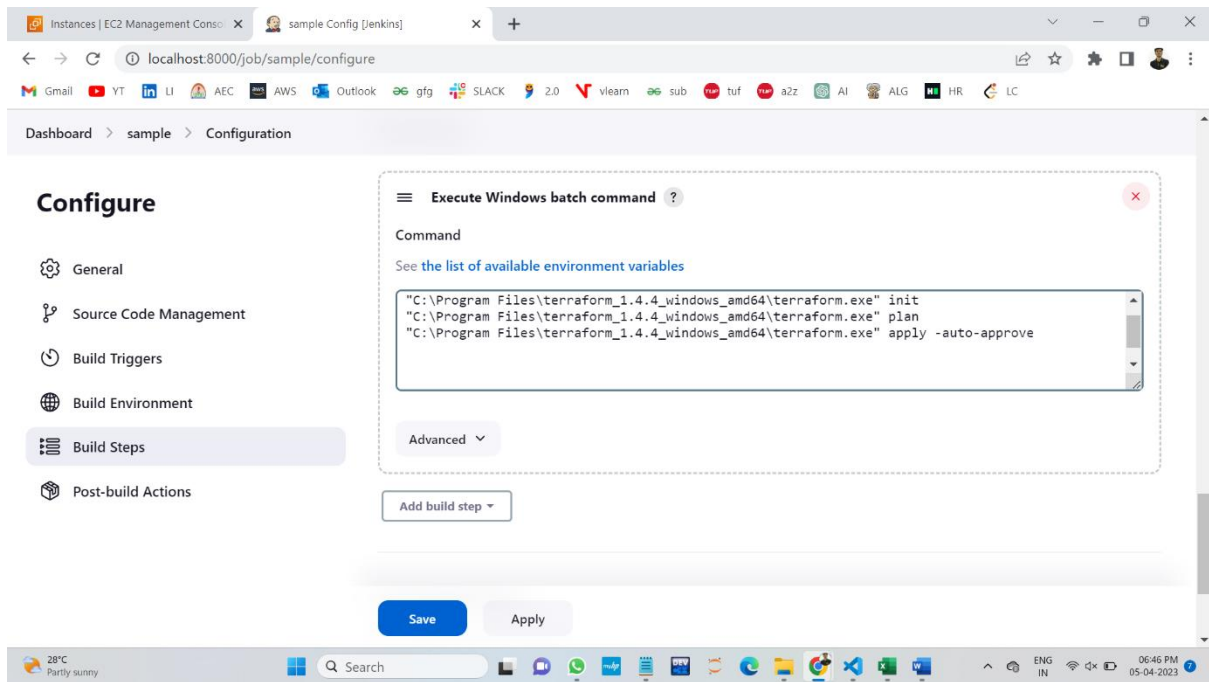


WITHOUT USING ANY GIT HUB REPOSITORY:

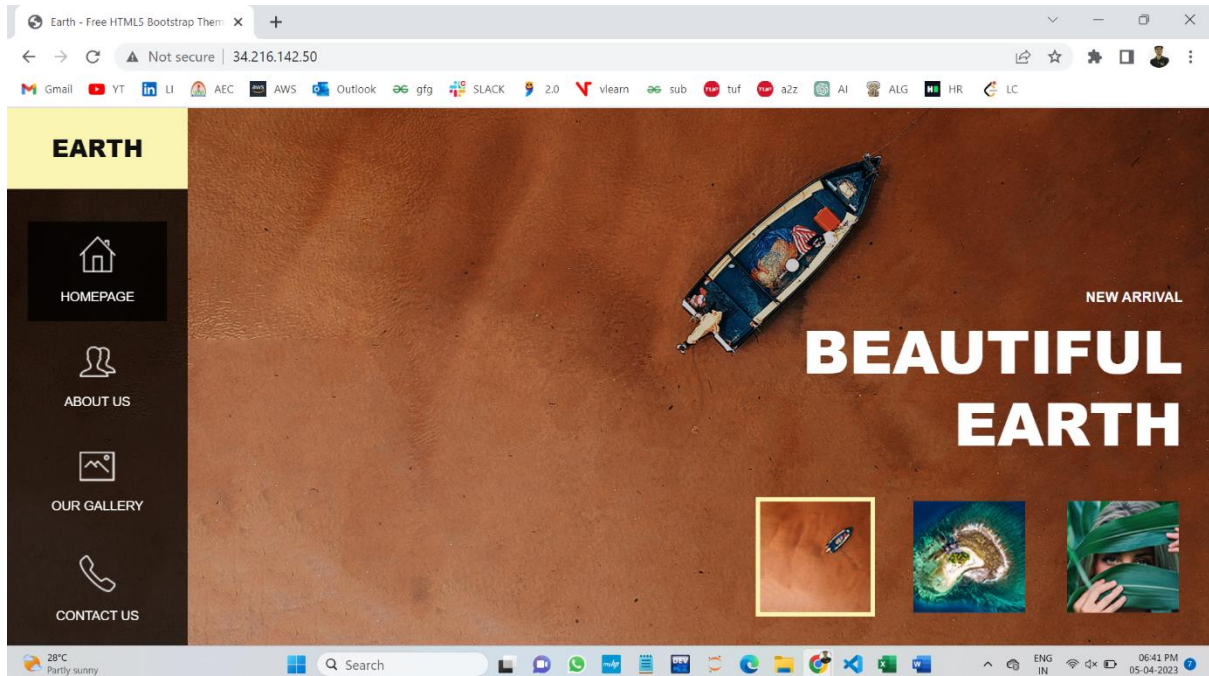
1. Here we don't need to upload in the git hub repository.
2. From 4th step we can use windows path and Global configure management And set the window paths of terraform, git and java.
3. Upload the directory where files are stored.



4. Upload the windows path as following in path with terraform commands Init, plan, apply.

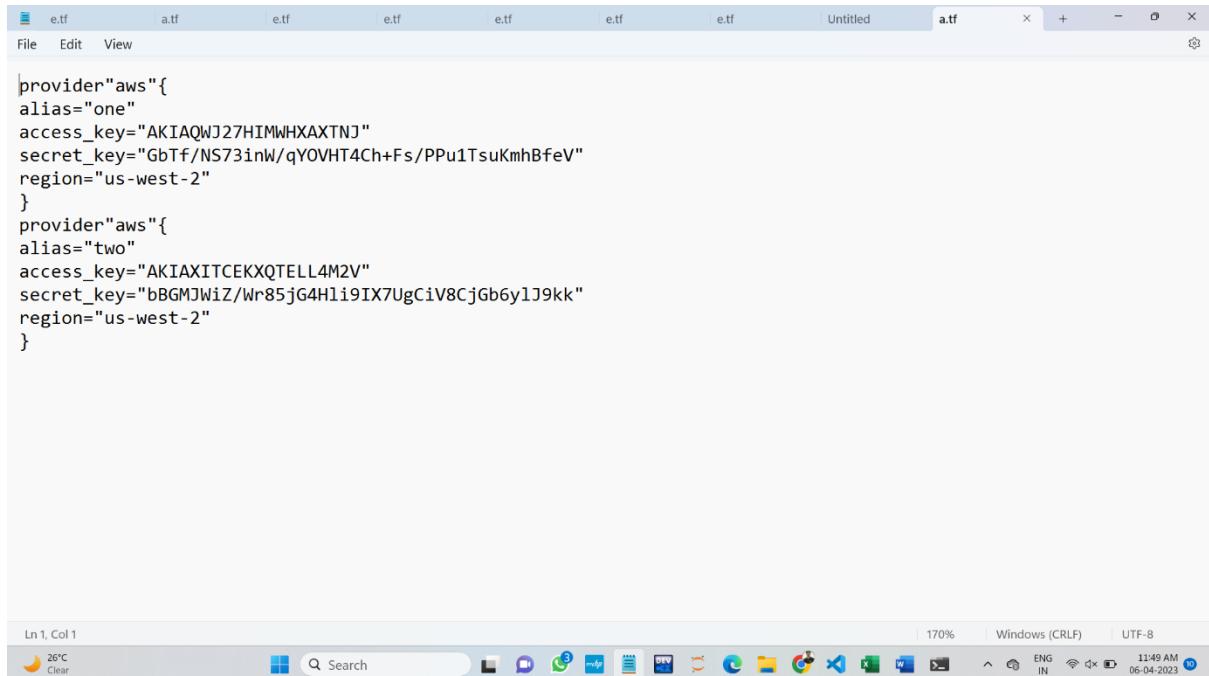


5. Build the Job in the Jenkins and run it.
6. Hence we can see instance created in the AWS Account.
7. By browse the public Ip Address.
8. Hence we can see Html web page displayed in the browser.



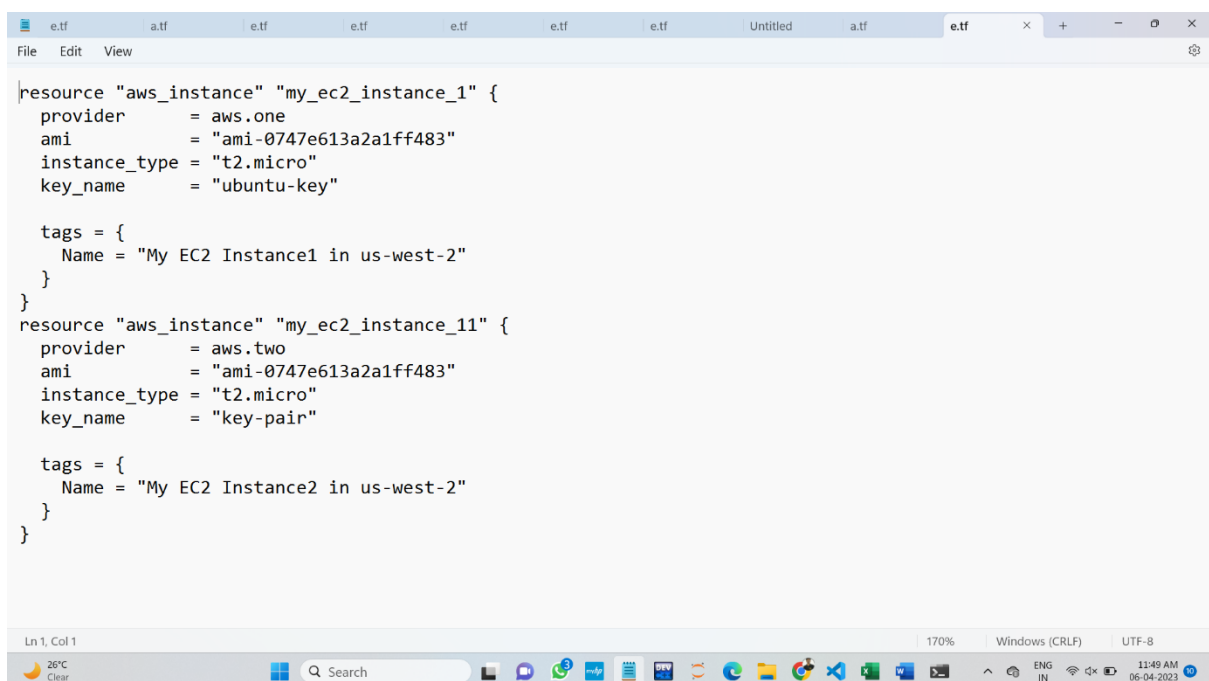
CREATING MULTI CONFIGURATION AND MULTI AUTHENTICATION USING TERRAFORM THROUGH JENKINS :

1.Create two files in notepad or vs and based on the required configuration of multi accounts



```
provider "aws" {
  alias = "one"
  access_key = "AKIAQWJ27HIMWHXAXTNJ"
  secret_key = "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"
  region = "us-west-2"
}
provider "aws" {
  alias = "two"
  access_key = "AKIAXITCEKXQTELL4M2V"
  secret_key = "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"
  region = "us-west-2"
}
```

The screenshot shows a Notepad++ window with a single tab named 'a.tf'. The text area contains Terraform provider configurations for two AWS accounts. The first provider is aliased 'one' and the second is aliased 'two'. Both are configured for the 'us-west-2' region. The status bar at the bottom indicates 'Ln 1, Col 1', '170%', 'Windows (CRLF)', and 'UTF-8'.



```
resource "aws_instance" "my_ec2_instance_1" {
  provider      = aws.one
  ami           = "ami-0747e613a2a1ff483"
  instance_type = "t2.micro"
  key_name      = "ubuntu-key"

  tags = {
    Name = "My EC2 Instance1 in us-west-2"
  }
}
resource "aws_instance" "my_ec2_instance_11" {
  provider      = aws.two
  ami           = "ami-0747e613a2a1ff483"
  instance_type = "t2.micro"
  key_name      = "key-pair"

  tags = {
    Name = "My EC2 Instance2 in us-west-2"
  }
}
```

The screenshot shows a Notepad++ window with a single tab named 'e.tf'. The text area contains Terraform resource configurations for two EC2 instances. The first resource, 'my_ec2_instance_1', uses the 'aws.one' provider and has a key name of 'ubuntu-key'. The second resource, 'my_ec2_instance_11', uses the 'aws.two' provider and has a key name of 'key-pair'. Both instances are of type 't2.micro' and use the same AMI. The status bar at the bottom indicates 'Ln 1, Col 1', '170%', 'Windows (CRLF)', and 'UTF-8'.


```
File Edit View
provider "aws" {
  alias = "group2"
  access_key = "ACCESS_KEY_4"
  secret_key = "SECRET_KEY_4"
  region = "us-east-1"
}

provider "aws" {
  alias = "group2"
  access_key = "ACCESS_KEY_5"
  secret_key = "SECRET_KEY_5"
  region = "us-east-2"
}

provider "aws" {
  alias = "group2"
  access_key = "ACCESS_KEY_6"
  secret_key = "SECRET_KEY_6"
  region = "us-west-1"
}
resource "aws_instance" "example" {
  provider = aws.group1
  #requirements
}

resource "aws_s3_bucket" "example" {
  provider = aws.group2
  #requirements
}
```

2. Now give the directory path in the configuration.

Instances | EC2 Management Console | sample Config [Jenkins]

localhost:8000/job/sample/configure

Dashboard > sample > Configuration

Configure

- General
- Source Code Management
- Build Triggers
- Build Environment
- Build Steps
- Post-build Actions

☒ Use custom workspace ?

Directory

C:\Users\UDAY BHASKAR\Desktop\t1

Display Name ?

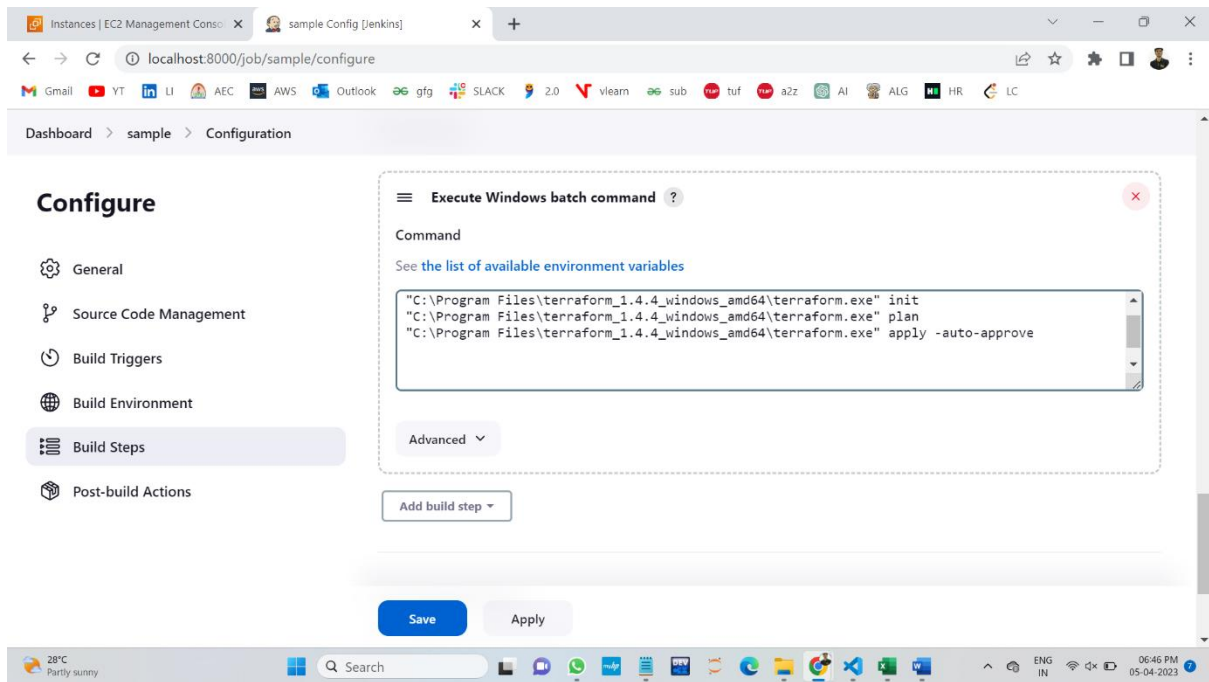
☐ Keep the build logs of dependencies ?

Source Code Management

☒ None

Save Apply

3. Give the commands which are to be executed in the Build Steps configuration and save it.



4. We can see that different services are created in different accounts by using Access key and access id.

