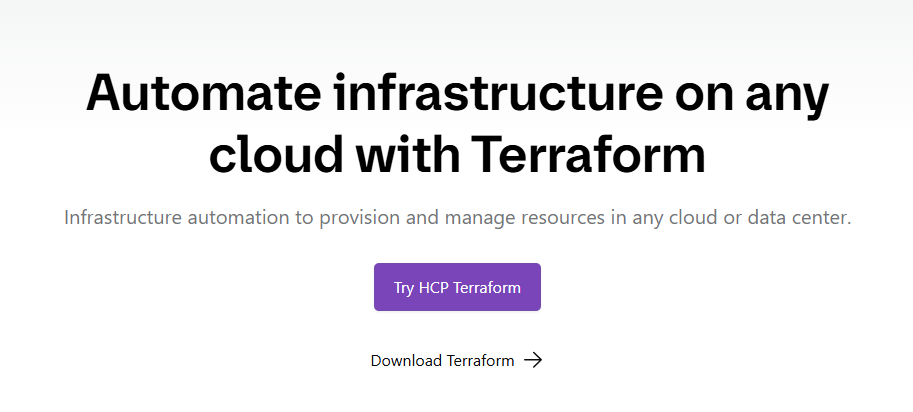
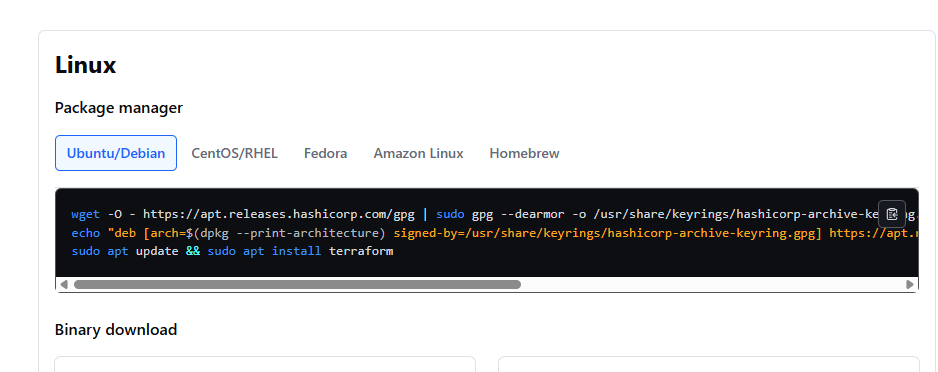
Terraform

<https://www.terraform.io/>



On Linux run this command, Terraform will be installed



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

echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb\_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list

sudo apt update && sudo apt install terraform

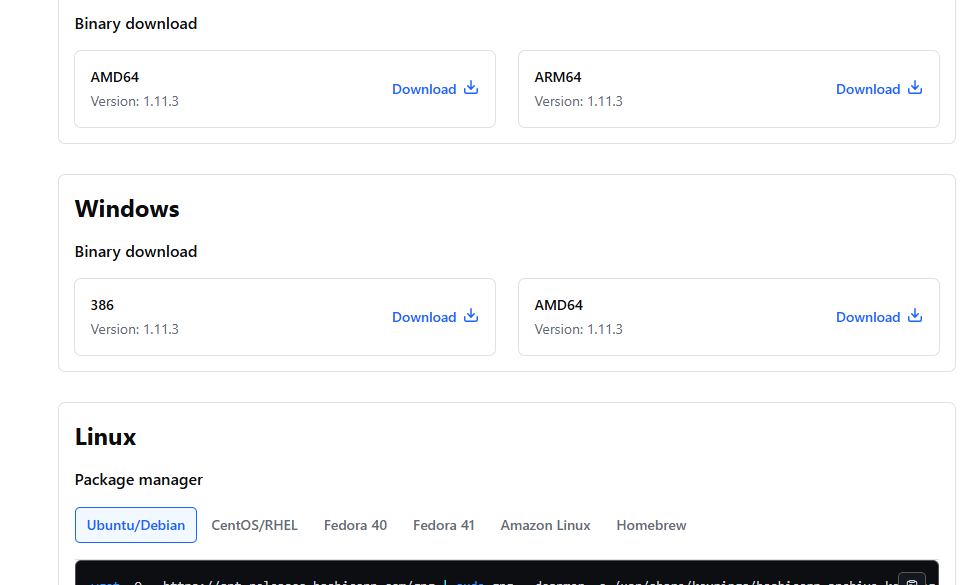
It is open-source and free to use

Terraform is an open-source and free software developed by HashiCorp

It is developed to create/provision infrastructure in cloud platform ---> it supports almost all the cloud platforms. (Infrastructure as a Code)

Terraform will use HCL ---> Hashicorp configuration language

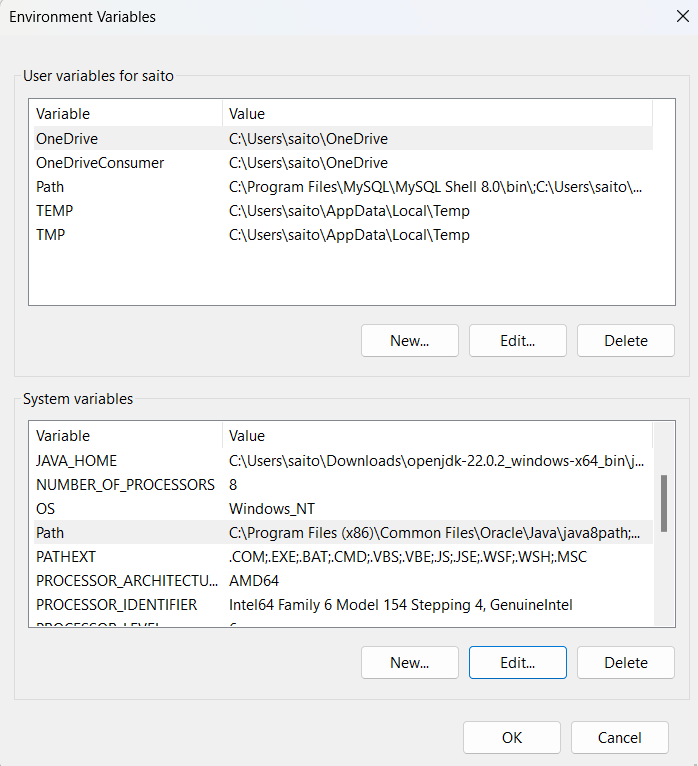
Terraform installation:

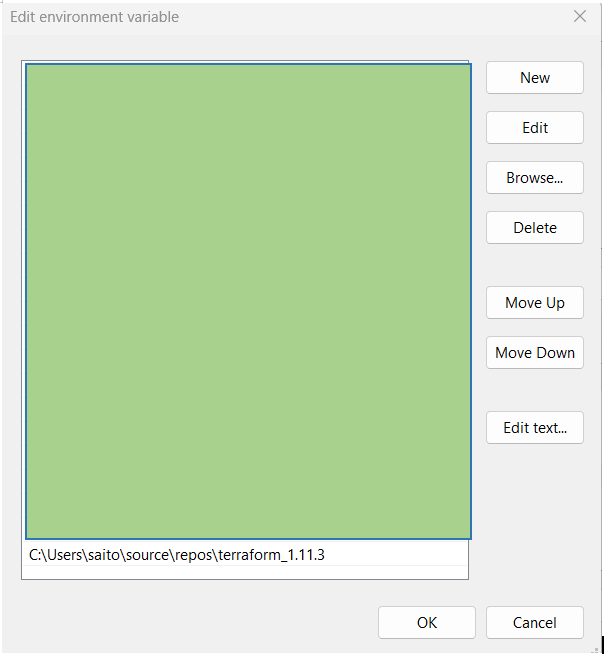


Windows Binary download

Extract Terraform files and copy terraform.exe to a folder

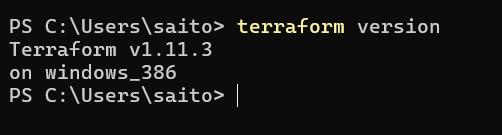
Open System variables and set path

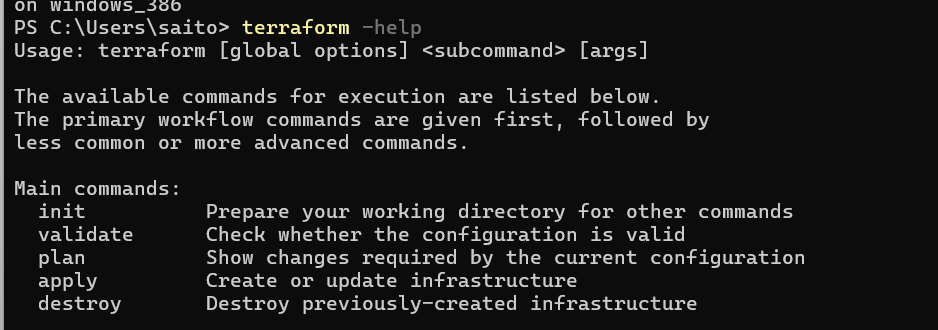




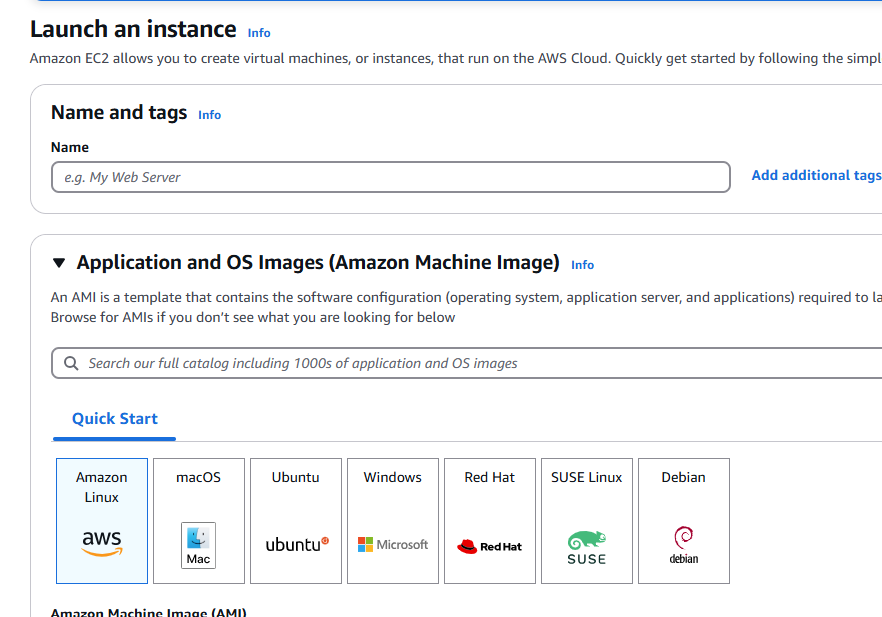
Add terraform path to the list of variables --> Ok

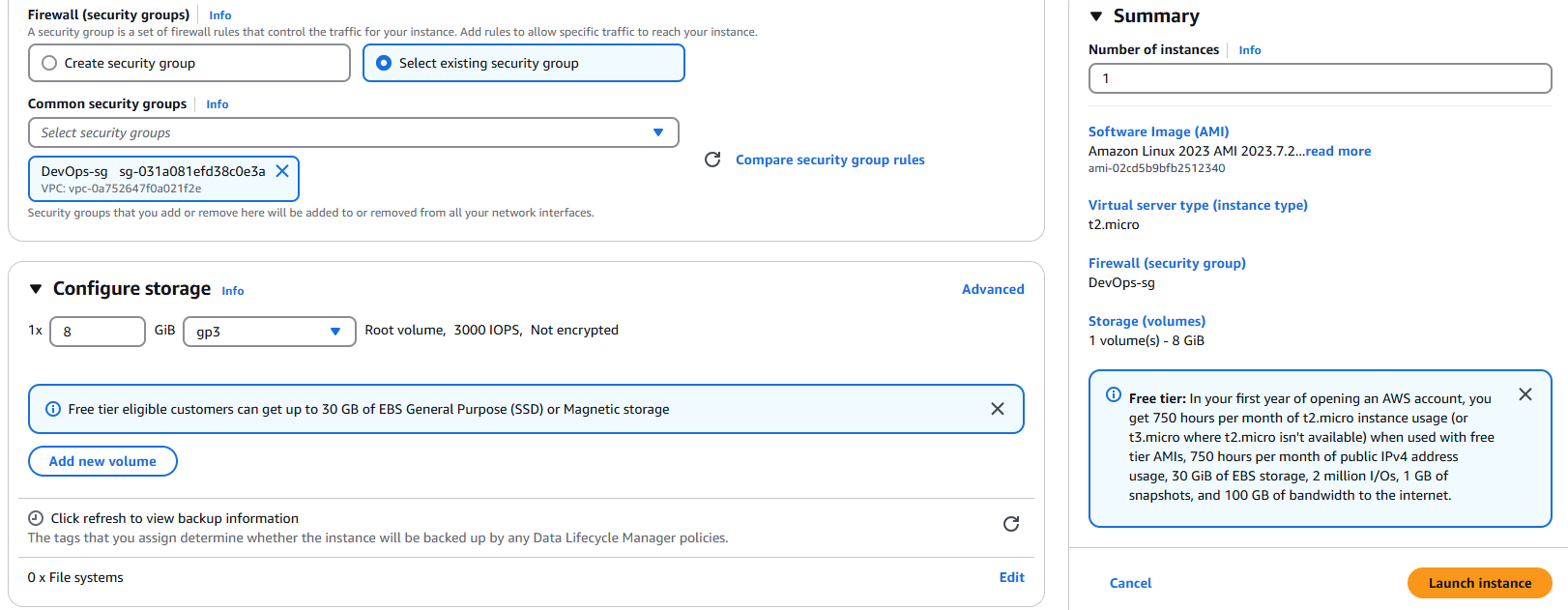
Open Windows Powershell or Command prompt





Create a new EC2



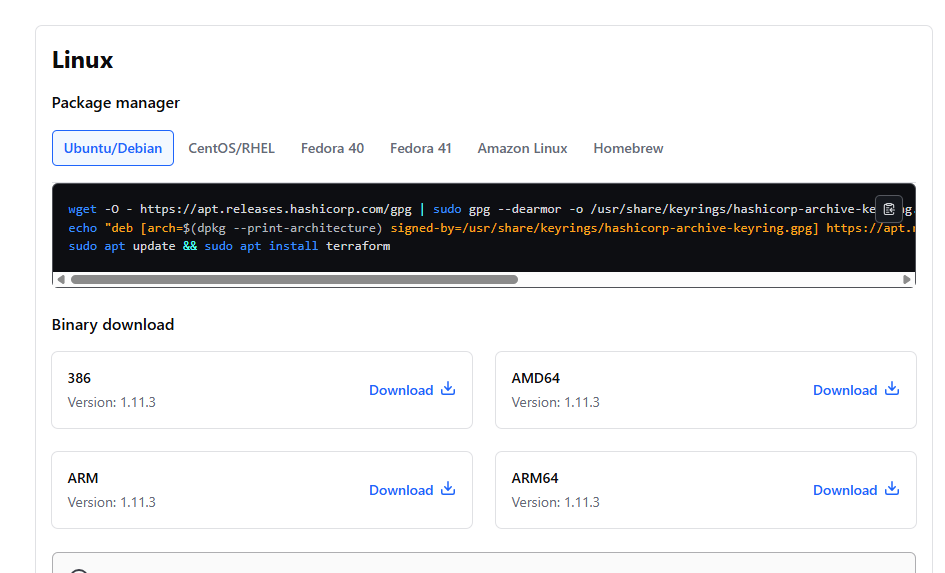


Copy public IP and open in MobaXTerm

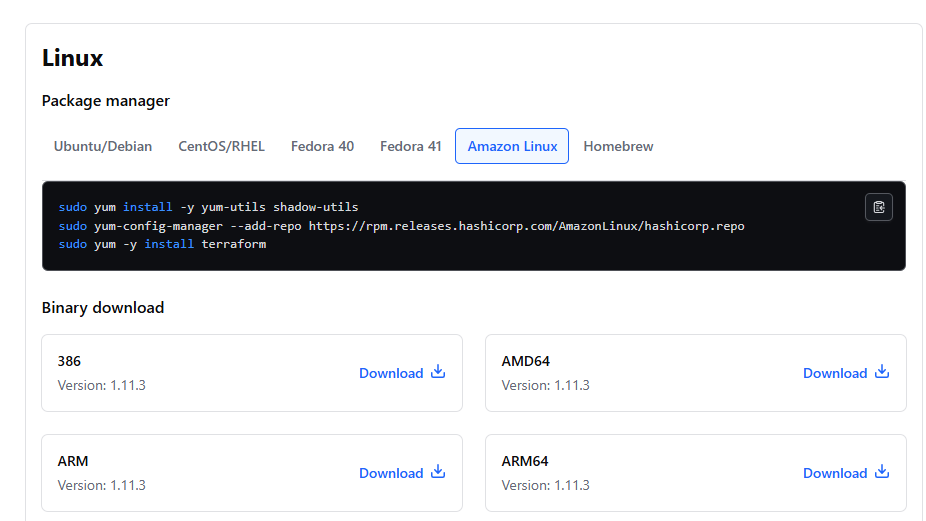


Go to Terraform website then download

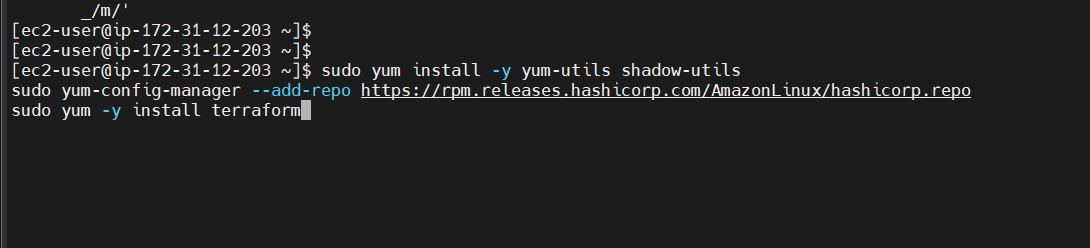
https://developer.hashicorp.com/terraform/install?product\_intent=terraform

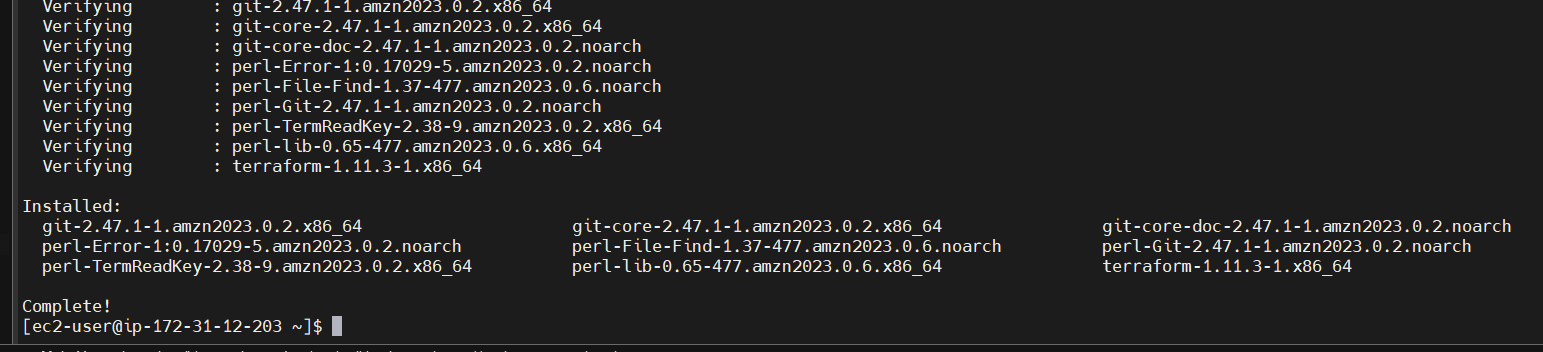


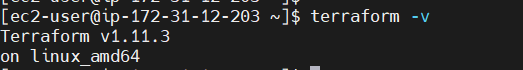
Because our EC2 has Amazon Linux, click that



Copy and paste into EC2







1. Created Linux VM in AWS cloud (AMI: Amazon Linux)
2. Conenct with Linux VM using MobaXTerm or Gitbash
3. Execute the below commands to setup Terraform in Linux VM

sudo yum install -y yum-utils shadow-utils

sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo

sudo yum -y install terraform

1. Verify Terraform installation
   1. terraform -v
   2. terraform version

Terraform Architecture

---> Terraform uses HCL to write scripts

---> .tf file

terraform validate

terraform init

terraform destroy

terraform apply

terraform plan

.tf file

terraform fmt

terraform destroy --> used to destroy only the resources created using the script

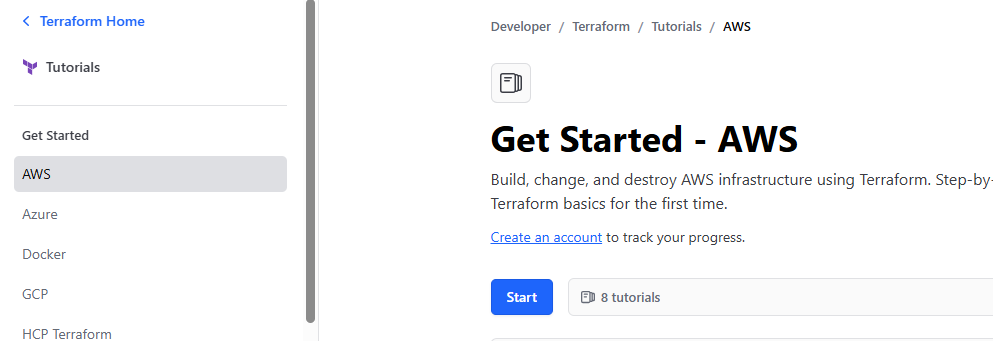
terraform validate --> validate the terraform commands you are written

terraform fmt --> it will format terraform commands with proper indent spacing

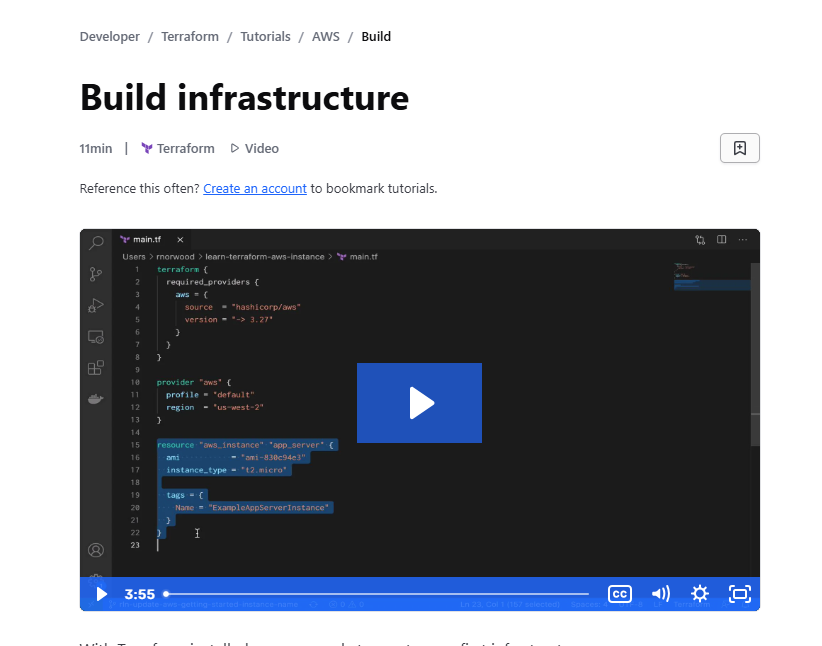
terraform plan --> create execution plan for the script

terraform apply --> it will actually create the resources

Go to the tutorials page: <https://developer.hashicorp.com/terraform/tutorials>



Build infrastructure



Example



<https://developer.hashicorp.com/terraform/tutorials/aws-get-started/aws-build>

---> Terraform script we use .tf extension

.tf --> init --> fmt --> validate --> plan --> apply --> destroy

terraform init --> initialize terraform script (.tf file)

terraform fmt --> format terraform script indent spacings (optional)

terraform validate --> verify terraform script syntax valid or not

terraform plan --> create execution plan for terraform script

terraform apply --> create actual resource in cloud based on given plan

terraform destroy --> it is used to delete resources created wkth our Terraform script

Terraform script to create EC2 instance

provider “aws” {

region=”ca-central-1”

access\_key=xxxxxxxxxxxxx

secret\_key=xxxxxxxxxxxxxxxxxx

}

resource “aws\_instance” “linux-vm”{

ami=”ami-02cd5b9bfb2512340”

instance\_type=”t2.micro”

key\_name=”terraform”

security\_groups=[“default”]

tags={

Name=”Terraform-test-VM”

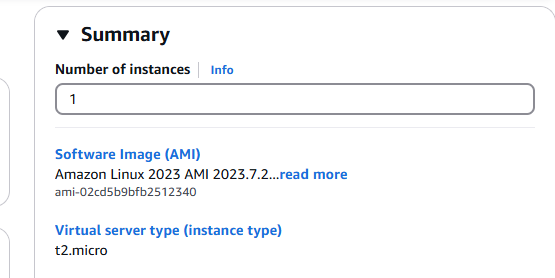
}

}

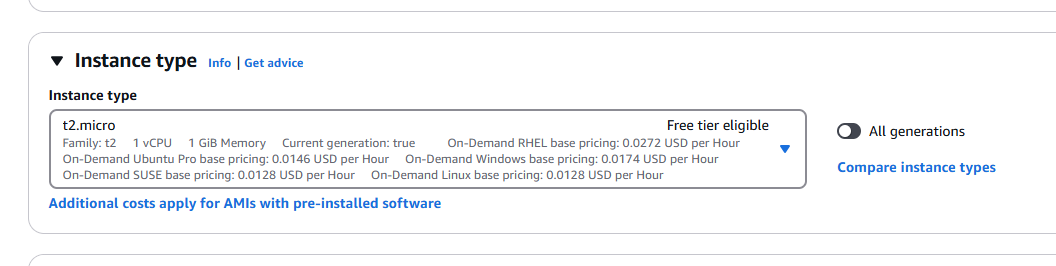
Create access keys

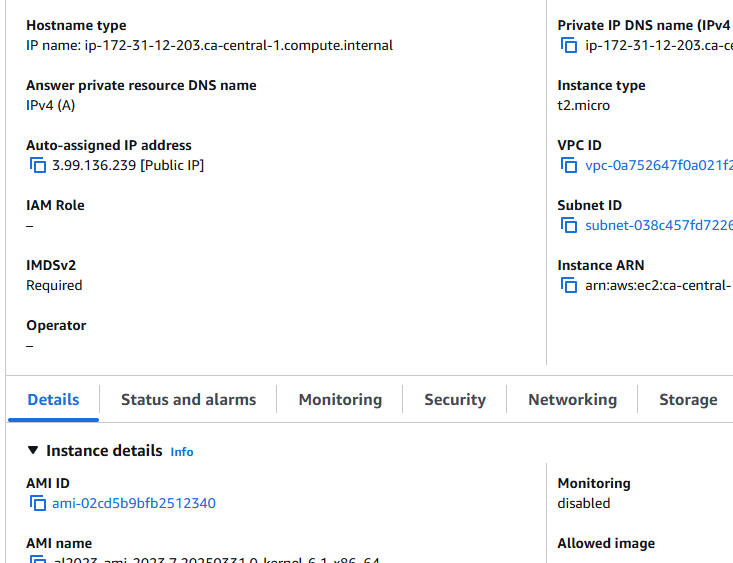
Go to EC2 launch instance and find AMI ID

AMI ID



t2.micro





Which cloud provider you want to use? Which resource within provider you want to use?

[ec2-user@ip-172-31-12-203 ~]$ mkdir 01-tf-script

[ec2-user@ip-172-31-12-203 ~]$

[ec2-user@ip-172-31-12-203 ~]$ ls -l

total 0

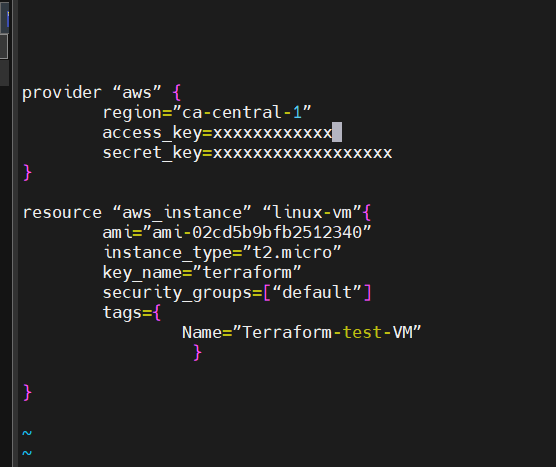
drwxr-xr-x. 2 ec2-user ec2-user 6 Apr 5 18:08 01-tf-script

[ec2-user@ip-172-31-12-203 ~]$

[ec2-user@ip-172-31-12-203 01-tf-script]$ vi main.tf

[ec2-user@ip-172-31-12-203 01-tf-script]$

Paste the entire script in Vim



provider “aws” {

region=”ca-central-1”

access\_key=xxxxxxxxxxxx

secret\_key=xxxxxxxxxxxxxxxxxx

}

resource “aws\_instance” “linux-vm”{

ami=”ami-02cd5b9bfb2512340”

instance\_type=”t2.micro”

key\_name=”terraform”

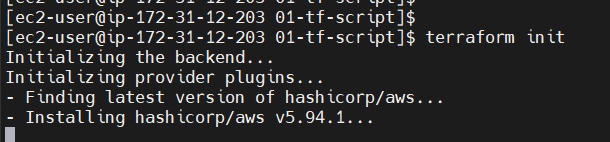
security\_groups=[“default”]

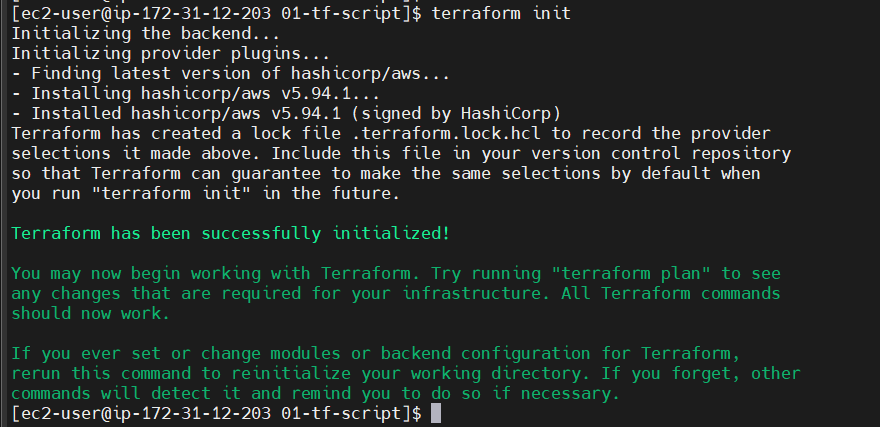
tags={

Name=”Terraform-test-VM”

}

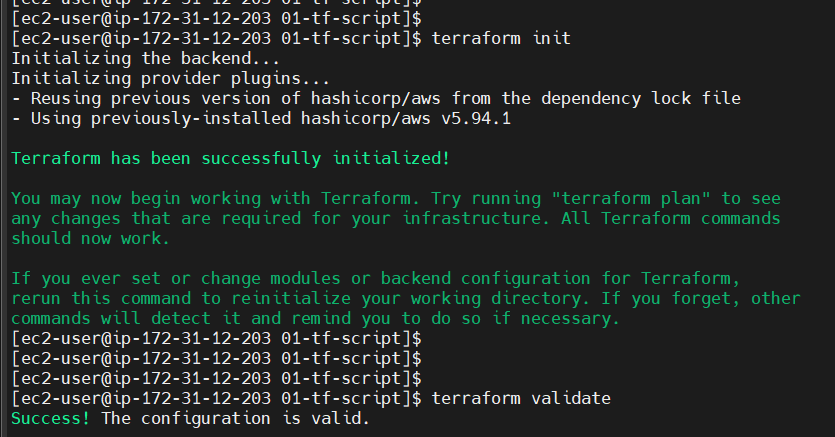
}



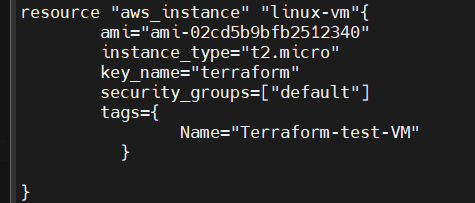


[ec2-user@ip-172-31-12-203 01-tf-script]$ terraform validate

Success! The configuration is valid.



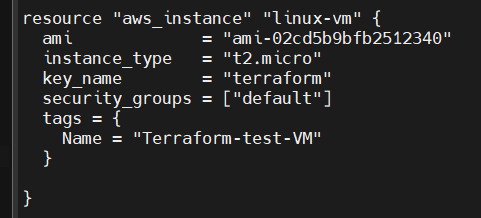
Shape before formatting



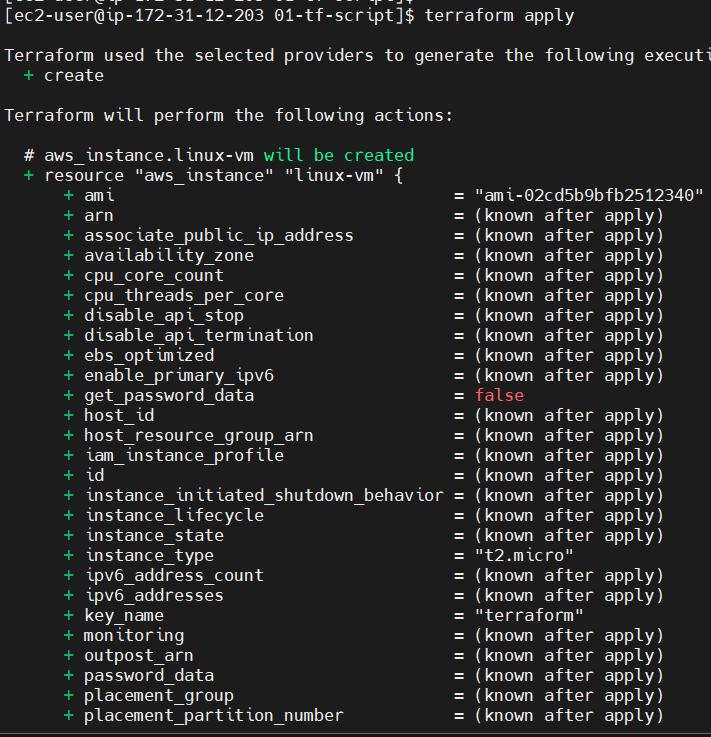
[ec2-user@ip-172-31-12-203 01-tf-script]$ terraform fmt

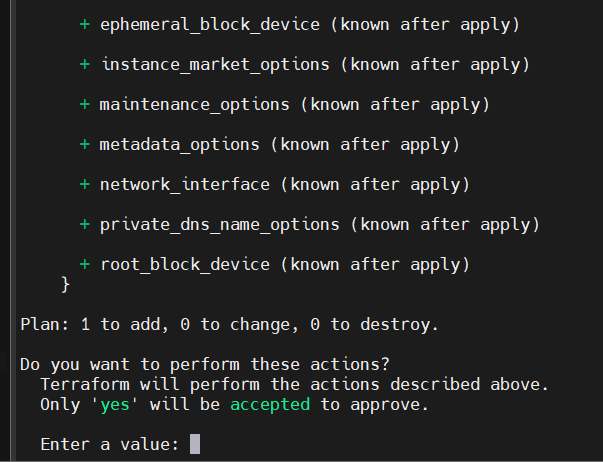
main.tf

After formatting

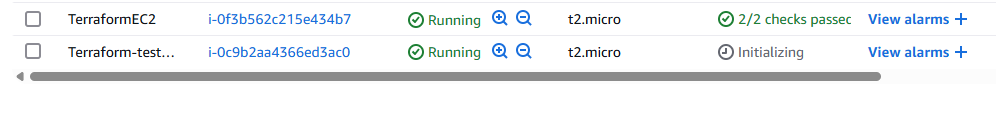


terraform apply

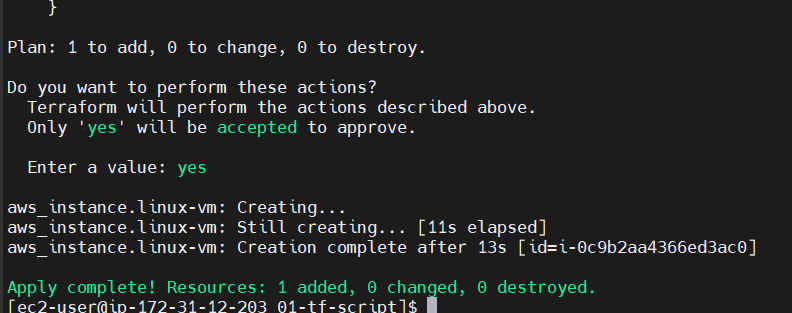




We verify what resources we want to create then we say yes



See now the new EC2 is being created ---> Terraform-test-VM



[ec2-user@ip-172-31-12-203 01-tf-script]$ ls -l

total 16

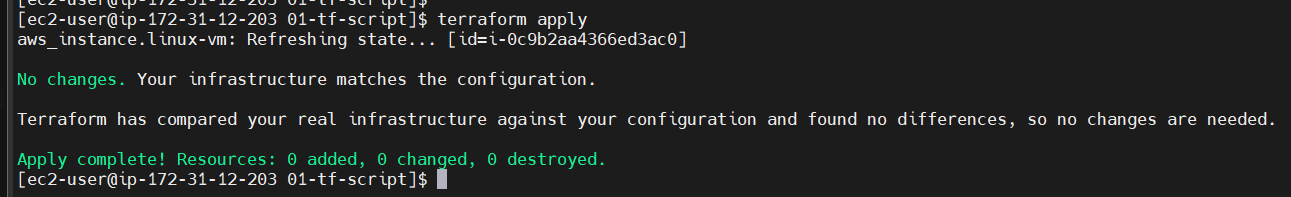
-rw-r--r--. 1 ec2-user ec2-user 374 Apr 5 18:30 main.tf

-rw-r--r--. 1 ec2-user ec2-user 4935 Apr 5 18:31 terraform.tfstate

-rw-r--r--. 1 ec2-user ec2-user 181 Apr 5 18:31 terraform.tfstate.backup

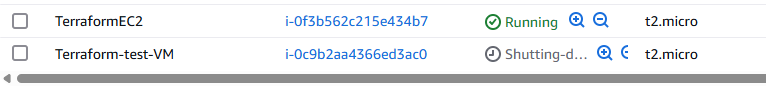
terraform.tfstate --> state file has all the information about terraform created resources

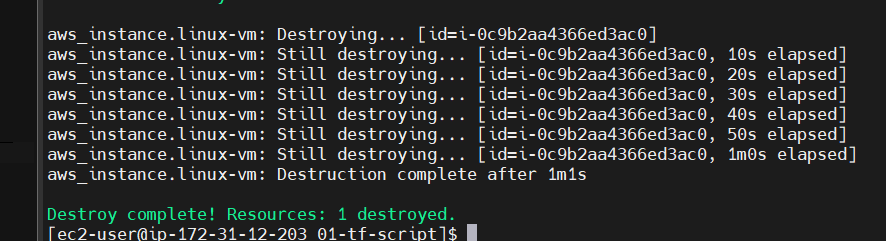
If I re-run terraform apply, no changes

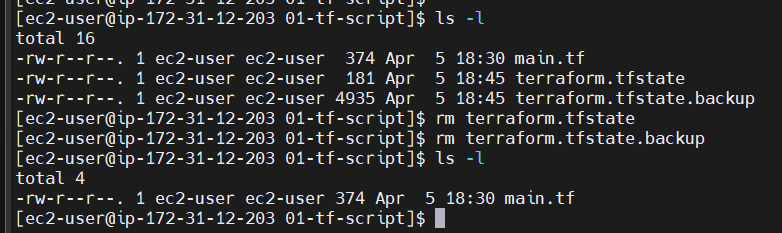


For destroying also, terraform will look for state file only

When I say destroy we can see it is shutting down



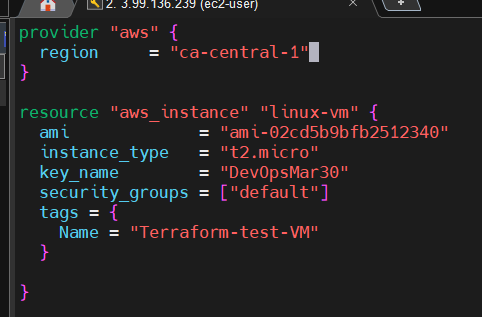




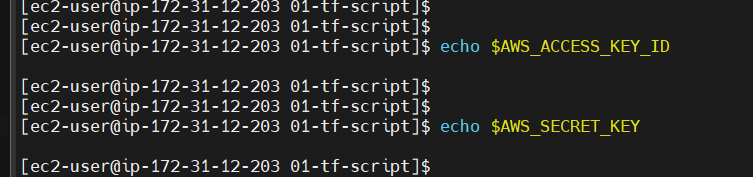
Dealing with Access key and Secret key

--> Instead of configuring access key and secret key in Terraform script file, we could configure them in environment variables. But even if you disconnect and re-connect to VM with the tools, then also environment variables will be gone. Again you have to add these values into environment variables

I have removed access key and secret key from main.tf



Currently we don’t have the keys in environment variables



Add environment variables using export command

[ec2-user@ip-172-31-12-203 01-tf-script]$ export AWS\_SECRET\_KEY=""

You can print and see as well

[ec2-user@ip-172-31-12-203 01-tf-script]$ echo $AWS\_ACCESS\_KEY\_ID

[ec2-user@ip-172-31-12-203 01-tf-script]$ echo $AWS\_SECRET\_KEY

[ec2-user@ip-172-31-12-203 01-tf-script]$ terraform init

Initializing the backend...

Initializing provider plugins...

- Reusing previous version of hashicorp/aws from the dependency lock file

- Using previously-installed hashicorp/aws v5.94.1

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.

[ec2-user@ip-172-31-12-203 01-tf-script]$ terraform fmt

main.tf

[ec2-user@ip-172-31-12-203 01-tf-script]$ ls -l

total 4

-rw-r--r--. 1 ec2-user ec2-user 274 Apr 5 19:14 main.tf

[ec2-user@ip-172-31-12-203 01-tf-script]$ terraform validate

Success! The configuration is valid.

[ec2-user@ip-172-31-12-203 01-tf-script]$ cat main.tf

provider "aws" {

region = "ca-central-1"

}

resource "aws\_instance" "linux-vm" {

ami = "ami-02cd5b9bfb2512340"

instance\_type = "t2.micro"

key\_name = "DevOpsMar30"

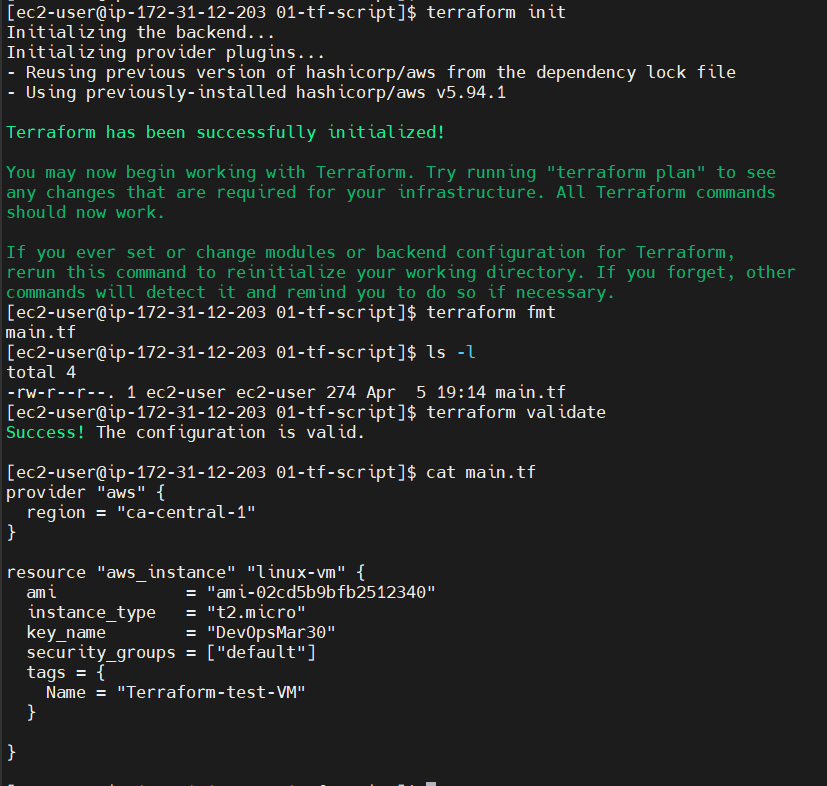
security\_groups = ["default"]

tags = {

Name = "Terraform-test-VM"

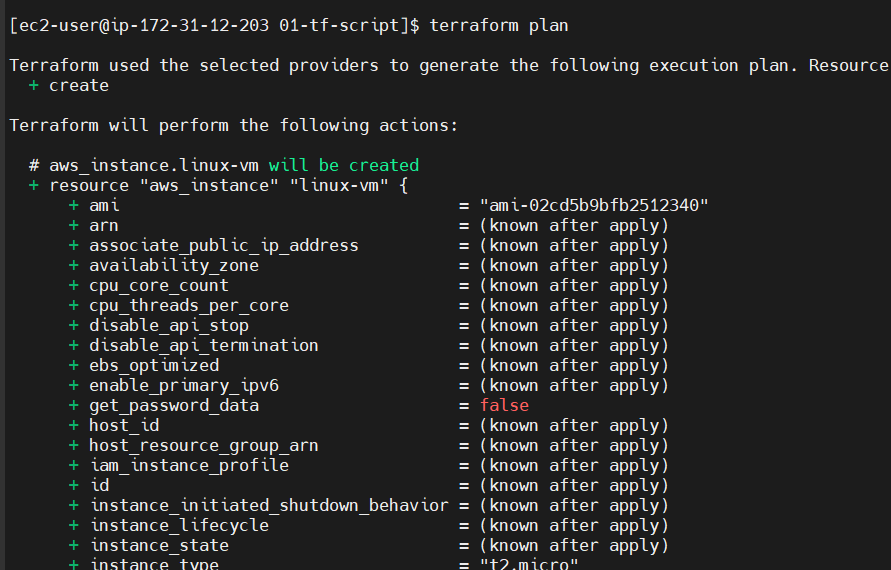
}

}

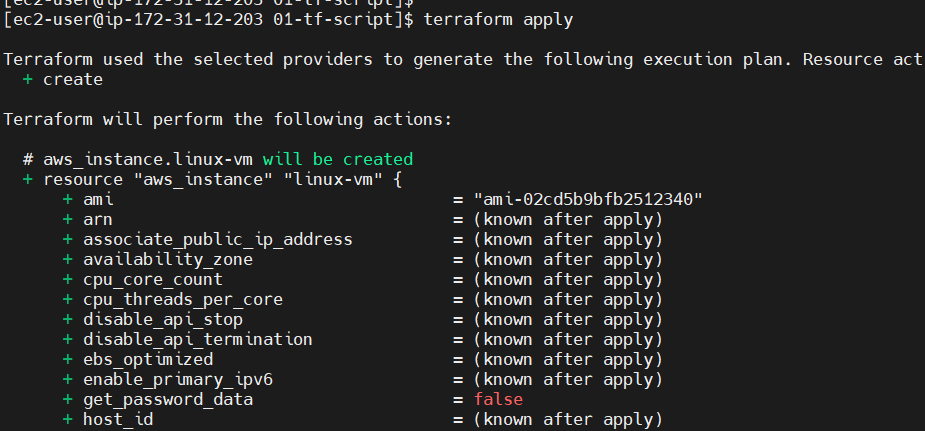


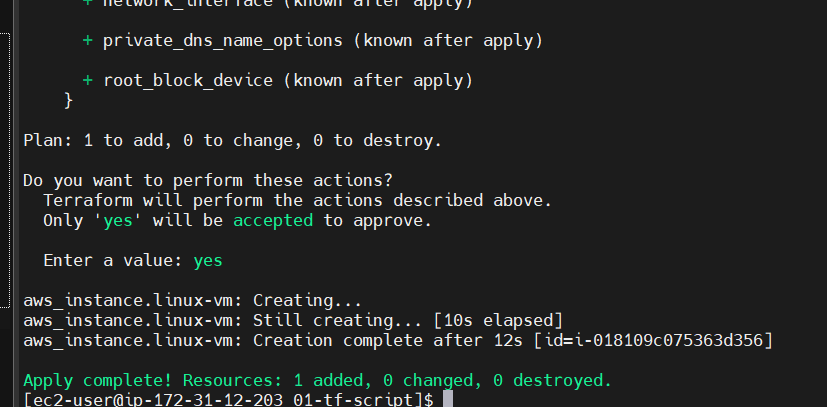
Because we have deleted tfstate files, now Terraform init and apply will work again

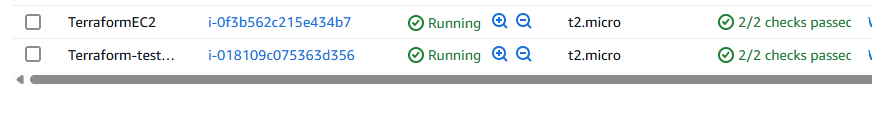
We have added secret key and access key in the environment variables



terraform apply



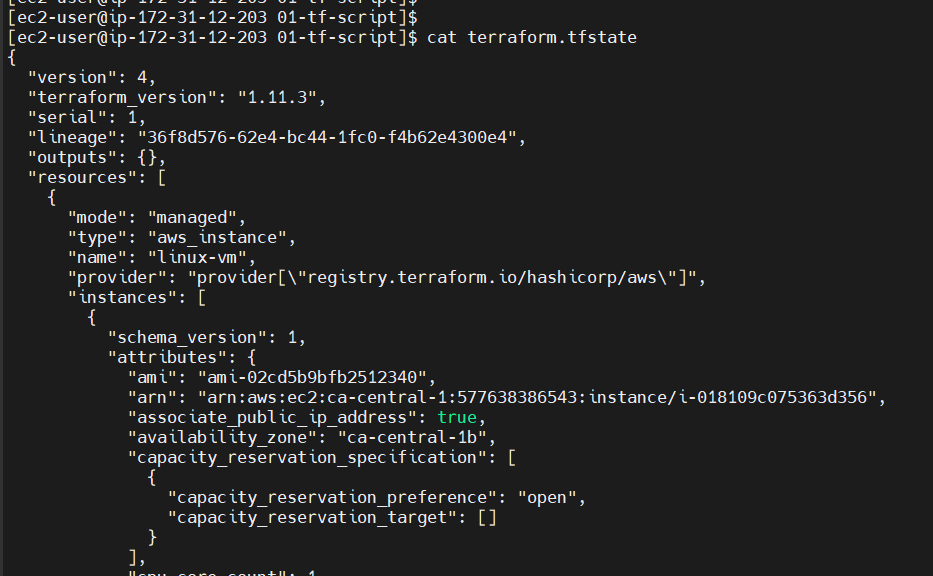


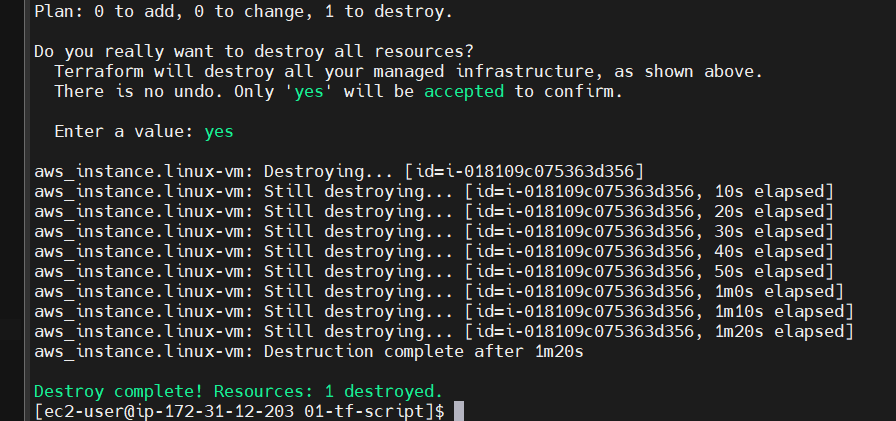


Again it’s up and running

terraform init, fmt, validate, plan, apply then yes

[ec2-user@ip-172-31-12-203 01-tf-script]$ cat terraform.tfstate





**Creating EC2 VM with User data:**

[ec2-user@ip-172-31-12-203 ~]$ cd 01-tf-script-userdata/

[ec2-user@ip-172-31-12-203 01-tf-script-userdata]$

[ec2-user@ip-172-31-12-203 01-tf-script-userdata]$

[ec2-user@ip-172-31-12-203 01-tf-script-userdata]$ ls

For that we got to write a script and reference in the Terraform file

Create Script sh file

[ec2-user@ip-172-31-12-203 01-tf-script-userdata]$ vi installHttpd.sh

#! /bin/bash

sudo su

yum install httpd -y

cd /var/www/html

echo "<html><h1> Welcome to Terraform Webserver </h1></html>" > index.html

service httpd start

[ec2-user@ip-172-31-12-203 01-tf-script-userdata]$ cat installHttpd.sh

#! /bin/bash

sudo su

yum install httpd -y

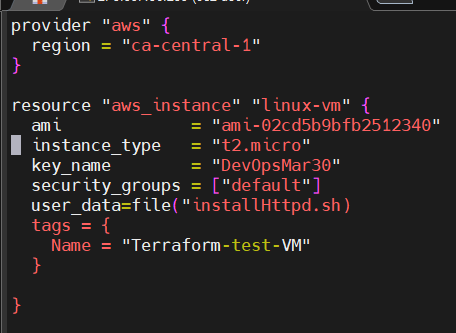
cd /var/www/html

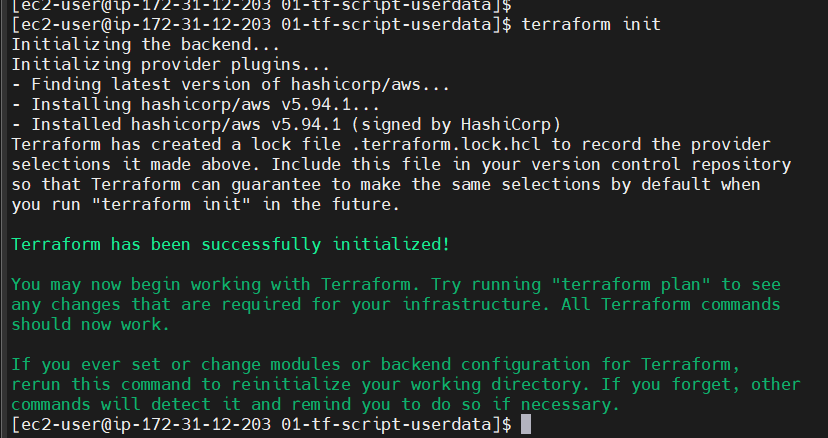
echo "<html><h1> Welcome to Terraform Webserver </h1></html>" > index.html

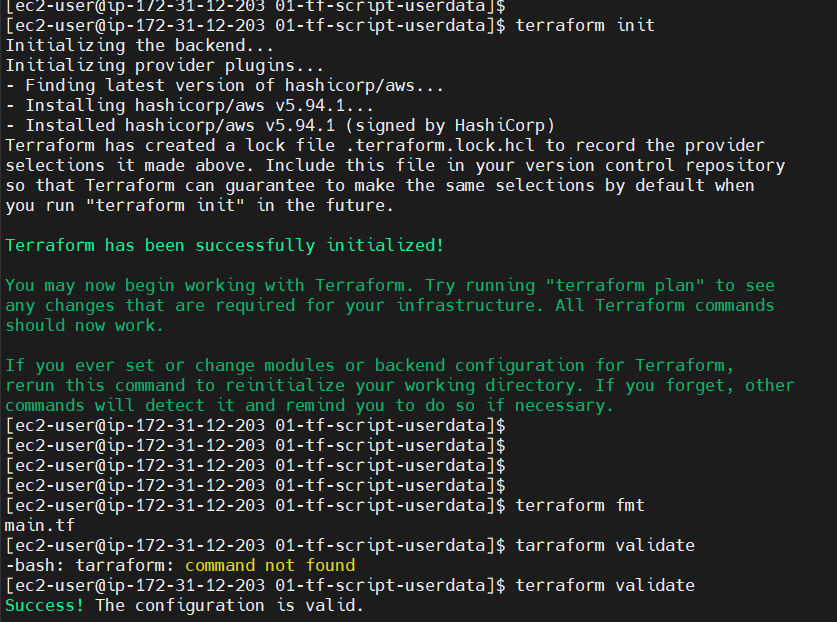
service httpd start

Give permission

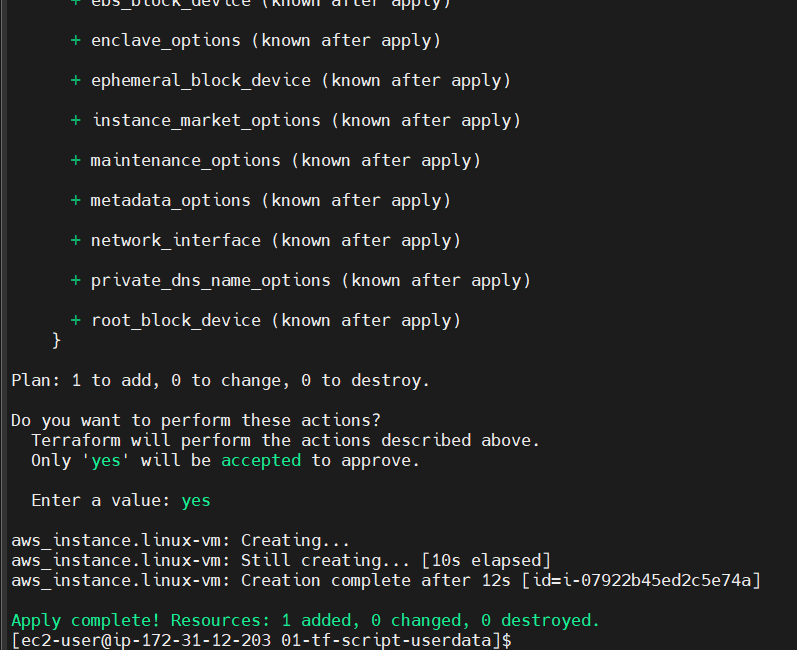
ec2-user@ip-172-31-12-203 01-tf-script-userdata]$ chmod u+x installHttpd.sh





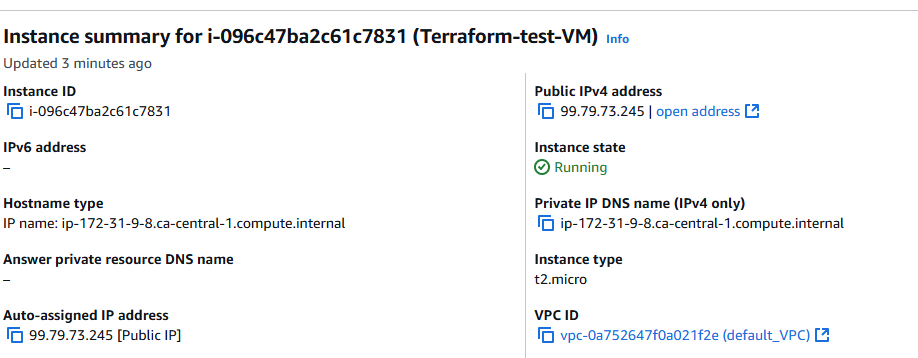


terraform apply --auto-approve also works

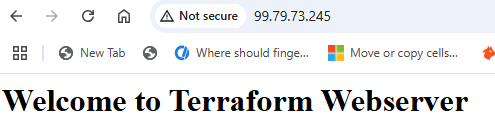


It created a VM



Open copy Public IP  


Open Public IP in browser



Add Access Key and Secret Key in environment variables

Create a new directory: mkdir 01-tf-script-userdata

Create a script file: vi installHttpd.sh

#! /bin/bash

sudo su

yum install httpd -y

cd /var/www/html

echo "<html><h1> Welcome to Terraform Webserver </h1></html>" > index.html

service httpd start

Provide the execute permission for Script file

chmod u+x installHttpd.sh

Create a script file: vi main.tf

provider "aws" {

region = "ca-central-1"

}

resource "aws\_instance" "linux-vm" {

ami = "ami-02cd5b9bfb2512340"

instance\_type = "t2.micro"

key\_name = "DevOpsMar30"

security\_groups = ["default"]

user\_data = file("installHttpd.sh")

tags = {

Name = "Terraform-test-VM"

}

}

terraform init

terraform fmt

terraform validate

terraform apply --auto-approve

terraform destroy

<https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/s3_bucket>

resource "aws\_s3\_bucket" "example" {

bucket = "my-tf-test-bucket"

tags = {

Name = "My bucket"

Environment = "Dev"

}

}

Variables in Terraform:

Variables are used to store data in key-value format

name=”DevOps”

Types of variables in Terraform ecosystem

1. Input variables ---> Supply values to Terraform script

We can remove hardcoded values from our resources script using Input variables concept

1. Output variables ---> Get the values from Terraform script after execution

Examples: After EC2 VM is created, print EC2-VM Public IP

After IAM user got created, print IAM user information

After S3 bucket got created, print bucket info

===> Variables

These variables we can maintain in separate Terraform tf file

[ec2-user@ip-172-31-12-203 03-tf-script-vars]$ cat vars.tf

variable "ami" {

description = "Amazon vm image value"

default = "ami-02cd5b9bfb2512340"

}

variable "instance\_type" {

description = "Represents the type of instance"

default = "t2.micro"

}

Created a new directory mkdir 03-tf-script-vars

cd 03-tf-script-vars

Created separate tf file for vars

vi vars.tf

variable "ami" {

description = "Amazon vm image value"

default = "ami-02cd5b9bfb2512340"

}

variable "instance\_type" {

description = "Represents the type of instance"

default = "t2.micro"

}

$ vi main.tf

[ec2-user@ip-172-31-12-203 03-tf-script-vars]$ cat main.tf

resource "aws\_instance" "linux-vm" {

ami = "${var.ami}"

instance\_type = "${var.instance\_type}"

key\_name = "DevOpsMar30"

security\_groups = ["default"]

tags = {

name = "Variable-Linux-VM"

}

}

[ec2-user@ip-172-31-12-203 03-tf-script-vars]$ ls -l

total 12

-rw-r--r--. 1 ec2-user ec2-user 230 Apr 5 23:04 main.tf

-rw-r--r--. 1 ec2-user ec2-user 44 Apr 5 23:05 provider.tf

-rw-r--r--. 1 ec2-user ec2-user 197 Apr 5 22:55 vars.tf

[ec2-user@ip-172-31-12-203 03-tf-script-vars]$ cat provider.tf

provider "aws" {

region = "ca-central-1"

}

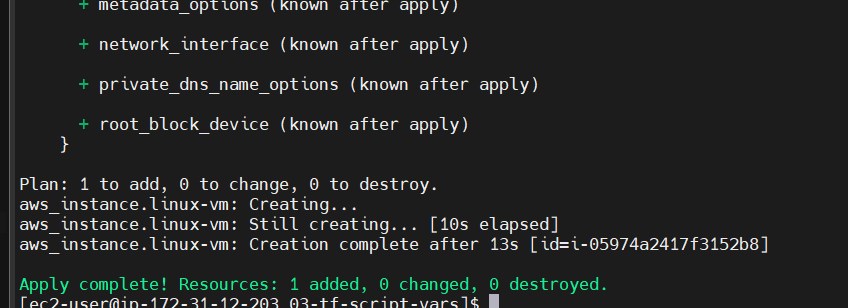
terraform init

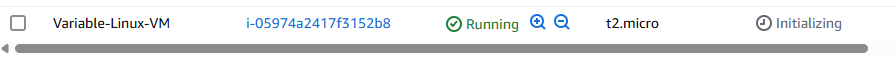
terraform fmt

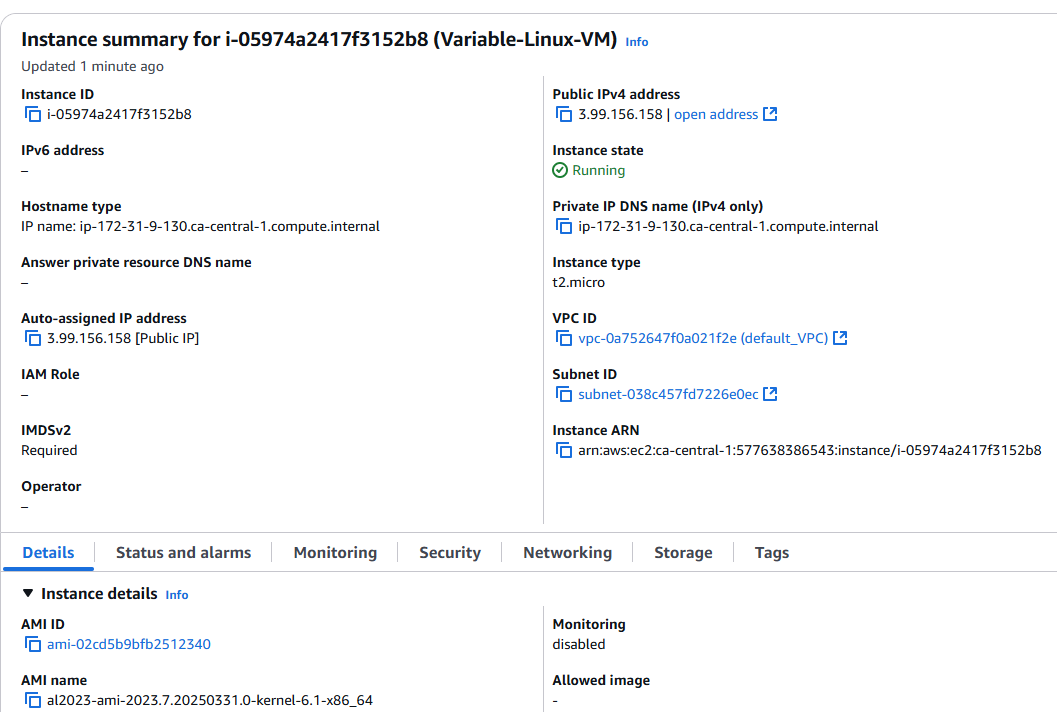
terraform validate

terraform plan

terraform apply







Instance\_type: t2.micro and AMI ID: ami-02cd5b9bfb2512340 match the vars.tf file

[ec2-user@ip-172-31-12-203 04-tf-script-var]$ ls -l

total 16

-rw-r--r--. 1 ec2-user ec2-user 207 Apr 5 23:43 input-vars.tf

-rw-r--r--. 1 ec2-user ec2-user 220 Apr 5 23:45 main.tf

-rw-r--r--. 1 ec2-user ec2-user 71 Apr 5 23:52 output.tf

-rw-r--r--. 1 ec2-user ec2-user 48 Apr 5 23:42 provider.tf

[ec2-user@ip-172-31-12-203 04-tf-script-var]$ cat input-vars.tf

variable "ami" {

description = "Amazon vm image value"

default = "ami-02cd5b9bfb2512340"

}

variable "instance\_type" {

description = "Represents the type of instance"

default = "t2.micro"

}

[ec2-user@ip-172-31-12-203 04-tf-script-var]$ cat main.tf

resource "aws\_instance" "linux\_vm" {

ami = vars.ami

instance\_type = vars.instance\_type

key\_name = "DevOpsMar30"

security\_groups = ["default"]

tags = {

Name = "Variable-Linux-VM"

}

}

[ec2-user@ip-172-31-12-203 04-tf-script-var]$ cat output.tf

output "ec2\_vm\_public\_ip" {

value=aws\_instance.linux\_vm.public\_ip

}

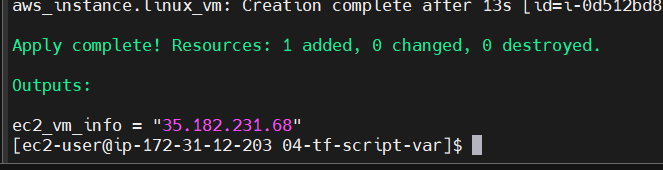
[ec2-user@ip-172-31-12-203 04-tf-script-var]$ cat provider.tf

provider "aws" {

region = "ca-central-1"

}

Change output.tf



[ec2-user@ip-172-31-12-203 04-tf-script-var]$ cat output.tf

output "ec2\_vm\_info" {

value = aws\_instance.linux\_vm

}

Now in the output we see the entire virtual machine

Terraform Script with Input & Output variables

Create new directory

Cd into new dir

Create provider.tf

vi provider.tf

Create input-vars.tf

vi input-vars.tf

Create output.tf

vi output.tf

resource "aws\_s3\_bucket" "example" {

bucket = "my-tf-test-bucket"

tags = {

Name = "My bucket"

Environment = "Dev"

}

}

Creating S3

[ec2-user@ip-172-31-12-203 05-tf-script-var]$ ls -l

total 20

-rw-r--r--. 1 ec2-user ec2-user 200 Apr 6 01:50 main.tf

-rw-r--r--. 1 ec2-user ec2-user 231 Apr 6 01:45 output.tf

-rw-r--r--. 1 ec2-user ec2-user 254 Apr 6 01:51 provider.tf

-rw-r--r--. 1 ec2-user ec2-user 3181 Apr 6 01:52 terraform.tfstate

-rw-r--r--. 1 ec2-user ec2-user 181 Apr 6 01:52 terraform.tfstate.backup

[ec2-user@ip-172-31-12-203 05-tf-script-var]$ cat main.tf

resource "aws\_s3\_bucket" "my\_bucket" {

bucket = "my-unique-bucket-name-12345-sai-dev-20250330" # make sure it's unique

tags = {

Name = "My S3 Bucket"

Environment = "Dev"

}

}

[ec2-user@ip-172-31-12-203 05-tf-script-var]$ cat provider.tf

provider "aws" {

region = "ca-central-1" # or your preferred AWS region like us-east-1, etc.

# Optional if you have AWS credentials configured via CLI or environment variables

# access\_key = "YOUR\_ACCESS\_KEY"

# secret\_key = "YOUR\_SECRET\_KEY"

}

[ec2-user@ip-172-31-12-203 05-tf-script-var]$ cat output.tf

output "s3\_bucket\_name" {

description = "The name of the S3 bucket"

value = aws\_s3\_bucket.my\_bucket.id

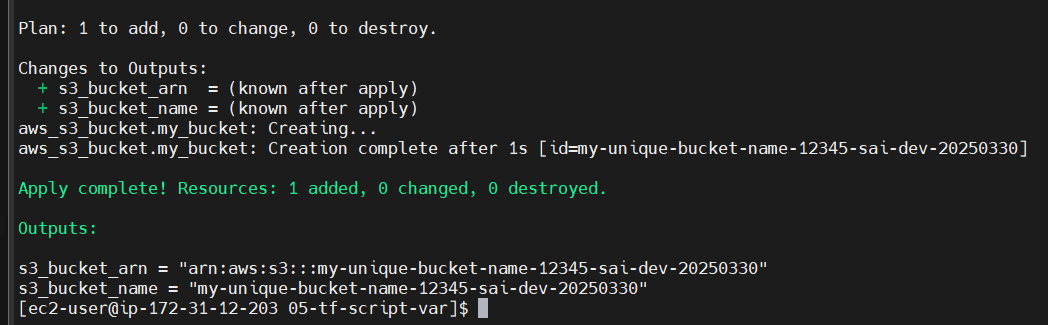
}

output "s3\_bucket\_arn" {

description = "The ARN of the S3 bucket"

value = aws\_s3\_bucket.my\_bucket.arn

}



Terraform has created the S3 bucket

