



# **VIRTUAL PRIVATE CLOUD USING TERRAFORM**

SUBMITTED BY

**TERRIFIC TERRAFORMS (77)**

**B.V.S.S.SUSHMA(TEAM LEADER)**

**T.KAVYA SRI**

**V.N.S.SRINITHA**

**Y.PUSHPA MOULIKA**

UNDER THE ESTEEMED GUIDANCE OF

**BOBBY SIR**

**SURYA ASHOK SIR**

## INTRODUCTION:

Terraform and VPC allow you to create and manage your VPC resources in a programmatic and automated way. This allows you to quickly and easily provision and manage VPC resources, while also reducing the risk of human error and enabling greater consistency across your infrastructure.

**A Virtual Private Cloud (VPC)** is a virtual network environment that provides a secure and isolated space for resources within the Amazon Web Services (AWS) cloud. VPCs can be customized to meet specific needs, and can be connected to on-premises data centers or other VPCs using VPN or Direct Connect.

**Terraform** is an open-source infrastructure as code (IaC) tool that allows you to define and manage your infrastructure as code. Terraform enables you to create, modify, and delete infrastructure resources in a safe and repeatable way, ensuring consistency and reducing the potential for human error.

**Subnet:** A subnet, or subnetwork, is a segmented piece of a larger network. More specifically, subnets are a logical partition of an IP network into multiple, smaller network segments. The Internet Protocol (IP) is the method for sending data from one computer to another over the internet. Each computer, or host, on the internet has at least one IP address as a unique identifier.

**Route table:** A route table contains a set of rules, called routes, that determine where network traffic from your subnet or gateway is directed.

**Internet Gateway:** An Internet gateway is a network "node" that connects two different networks that use different protocols (rules) for communicating. In the most basic terms, an Internet gateway is where data stops on its way to or from other networks.

**NAT Gateway:** A NAT gateway is a Network Address Translation (NAT) service. You can use a NAT gateway so that instances in a private subnet can connect to services outside your VPC but external services cannot initiate a connection with those instances.

In essence, using Terraform to create and manage a VPC enables you to define your infrastructure as code, enabling you to rapidly and consistently create and manage your cloud infrastructure.

## SCOPE OF THE PROJECT

The scope of a Virtual Private Cloud (VPC) using Terraform project is to create a secure and isolated network environment in the cloud. The project includes configuring the VPC, subnets, route tables, internet gateways, security groups, NAT gateways, and VPN connections using infrastructure as code. The goal is to provide a reliable, scalable, and customizable infrastructure that can be easily managed and updated. Overall, the project scope is to create a flexible and secure cloud-based networking solution that can meet the requirements of various applications and workloads.

## PURPOSE OF THE PROJECT

The purpose of a Virtual Private Cloud (VPC) using Terraform project is to create a secure, flexible, and cost-effective networking infrastructure in the cloud for running applications and workloads. The project achieves this by defining and managing the VPC resources as code, which enables easy and efficient deployment, management, and updating of the infrastructure.

## TOOLS USED

There are several tools used in a Virtual Private Cloud (VPC) using Terraform project to create and manage the networking infrastructure in the cloud. These tools include

**1.Terraform:** Terraform is an open-source infrastructure as code tool used to create, manage, and update the VPC and its resources.

**2.Amazon Virtual Private Cloud (VPC):** Amazon VPC is a service that allows you to create a virtual network environment in the cloud.

**3.AWS CLI:** It allows users to interact with AWS services and resources using commands in the terminal. This can be useful for tasks such as configuring AWS credentials, managing EC2 instances, and performing other AWS resource management tasks.

**4.Visual Studio Code (VS Code):** Visual studio code can be used for creating and managing a Virtual Private Cloud (VPC) using Terraform.

## TECHNOLOGIES USED

The technology used in a Virtual Private Cloud (VPC) using Terraform includes AWS, Amazon VPC and IAM. Terraform is used as an infrastructure as code (IaC) tool to define and manage the VPC and its resources. Other tools and technologies may also be used to automate the deployment and management of the VPC using Terraform.

## PROOF OF CONCEPT

The proof of concept (POC) for a Virtual Private Cloud (VPC) using Terraform typically involves creating a simple VPC with a few resources to demonstrate the basic functionality and feasibility of using Terraform for infrastructure as code (IaC).

**STEP1:** first we need to install terraform and aws cli in our local system. Now open aws account and create IAM user with administrator access and go to security credentials and create access key and secret key and click on download csv file.

Identity and Access Management (IAM)

Search IAM

Dashboard

Access management

Access reports

IAM dashboard

Security recommendations 2

Add MFA for root user

Root user has no active access keys

Update your access permissions for AWS Billing, Cost Management, and Account consoles

Add MFA

View affected policies

IAM resources

User groups

Users

Roles

Policies

Identity providers

1

0

15

2

0

AWS Account

Account ID

Account Alias

Sign-in URL for IAM users in this account

Quick Links

My security credentials

Tools

CloudShell

Feedback

Language

© 2023, Amazon Web Services India Private Limited or its affiliates.

Privacy

Terms

Cookie preferences

IAM > Users > Create user

Step 1

Specify user details

Step 2

Set permissions

Step 3

Review and create

Step 4

Retrieve password

Specify user details

User details

User name

ec2\_terraform

Provide user access to the AWS Management Console - optional

Are you providing console access to a person?

Console password

CloudShell

Feedback

Language

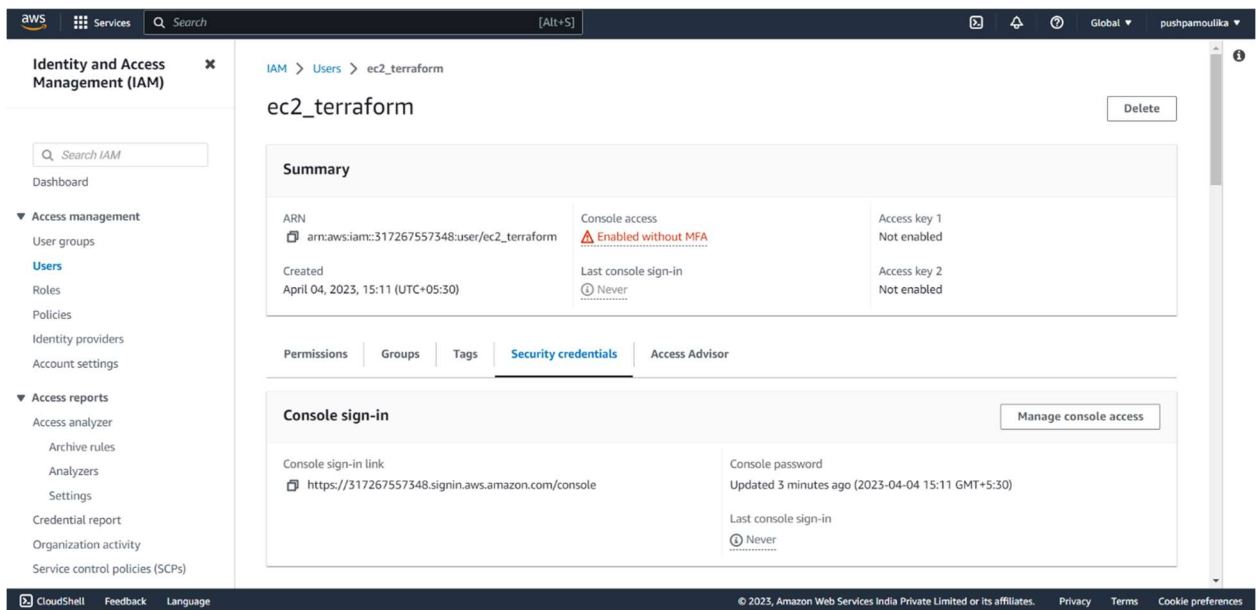
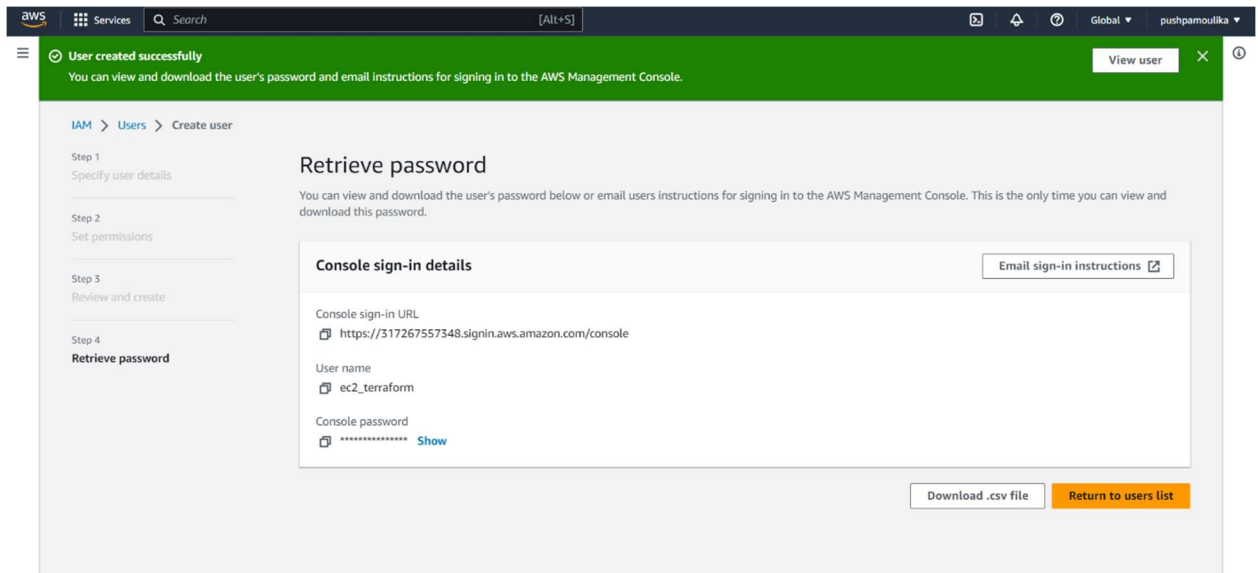
© 2023, Amazon Web Services India Private Limited or its affiliates.

Privacy

Terms

Cookie preferences





aws

Services

Search

[Alt+S]

Global

pushpamoulika

IAM > Users > ec2\_terraform > Create access key

Step 1  
Access key best practices & alternatives

Step 2 - optional  
Set description tag

Step 3  
Retrieve access keys

## Access key best practices & alternatives

Avoid using long-term credentials like access keys to improve your security. Consider the following use cases and alternatives.

☒ **Command Line Interface (CLI)**  
You plan to use this access key to enable the AWS CLI to access your AWS account.

☐ **Local code**  
You plan to use this access key to enable application code in a local development environment to access your AWS account.

☐ **Application running on an AWS compute service**  
You plan to use this access key to enable application code running on an AWS compute service like Amazon EC2, Amazon ECS, or AWS Lambda to access your AWS account.

☐ **Third-party service**  
You plan to use this access key to enable access for a third-party application or service that monitors or manages your AWS resources.

☐ **Application running outside AWS**  
You plan to use this access key to enable an application running on an on-premises host, or to use a local AWS client or third-party AWS plugin.

☐ **Other**

CloudShell

Feedback

Language

© 2021 Amazon Web Services India Private Limited or its affiliates.

Privacy

Terms

Cookie preferences

aws

Services

Search

[Alt+S]

Global

pushpamoulika

IAM > Users > ec2\_terraform > Create access key

Step 1  
Access key best practices & alternatives

Step 2 - optional  
Set description tag

Step 3  
Retrieve access keys

## Set description tag - optional

The description for this access key will be attached to this user as a tag and shown alongside the access key.

Description tag value

Describe the purpose of this access key and where it will be used. A good description will help you rotate this access key confidently later.

accesskey

Maximum 256 characters. Allowed characters are letters, numbers, spaces representable in UTF-8, and: \_ . : / = + - @

Cancel

Previous

Create access key

CloudShell

Feedback

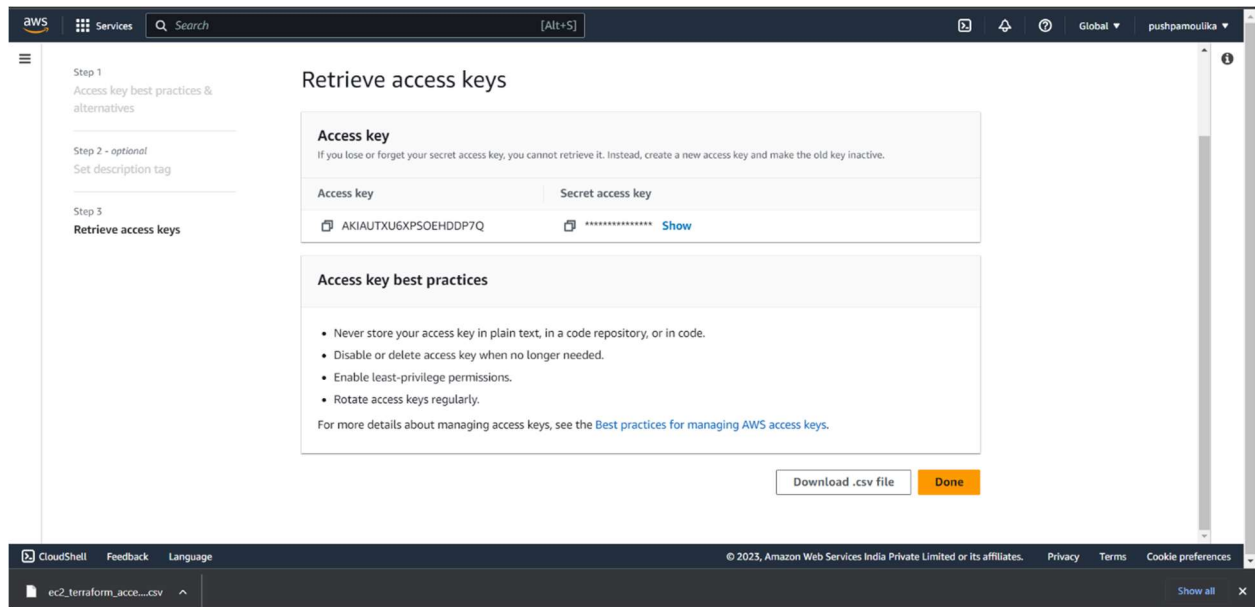
Language

© 2021 Amazon Web Services India Private Limited or its affiliates.

Privacy

Terms

Cookie preferences



**STEP2:** Now open PowerShell with run as administrator and give the command `aws configure`.

It asks for access key and secret key give them later it asks for region give default.

```
PS C:\Users\lenovo> aws configure
AWS Access Key ID [None]: AKIAUTXU6XPSOEHDDP7Q
AWS Secret Access Key [None]: GUL90DCm0iGEAUi5Wxjr dy80NoSE9sAFuT0iqF6t|
```

```
PS C:\Users\lenovo> aws configure
AWS Access Key ID [None]: AKIAUTXU6XPSOEHDDP7Q
AWS Secret Access Key [None]: GUL90DCm0iGEAUi5Wxjr dy80NoSE9sAFuT0iqF6t
Default region name [None]: default
Default output format [None]: default
PS C:\Users\lenovo> |
```

To generate the SSH key give the command `ssh-keygen`.

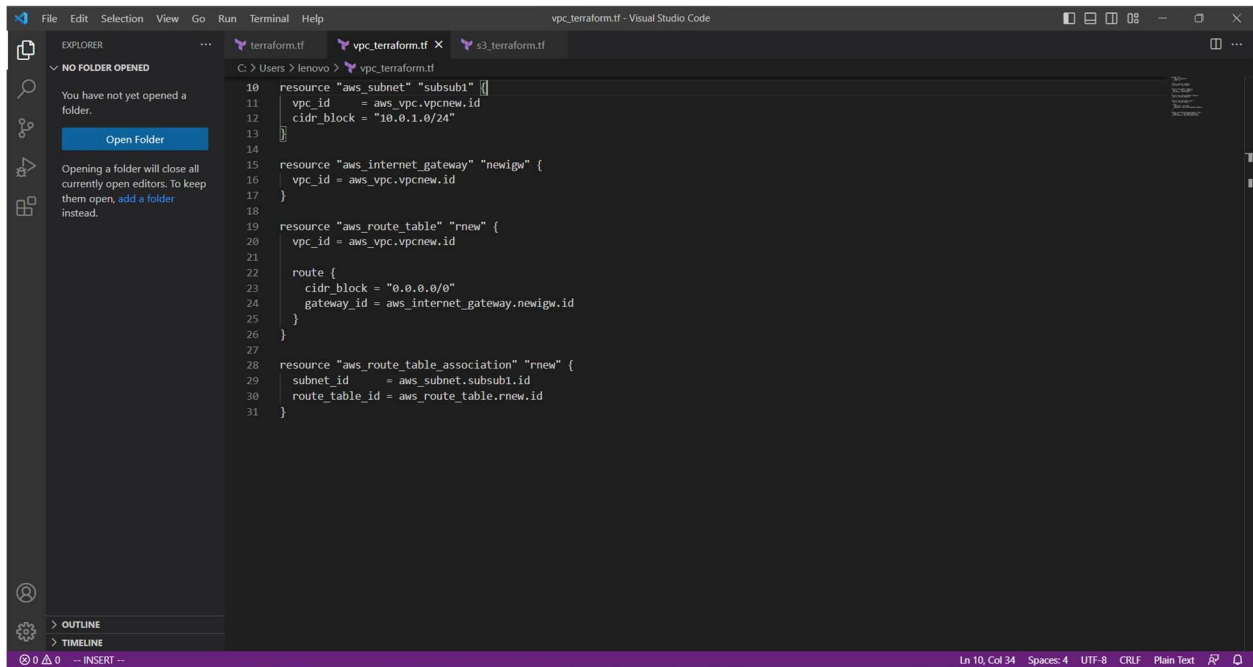


```

PS C:\Users\lenovo> ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (C:\Users\lenovo/.ssh/id_rsa): sshfile.txt
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in sshfile.txt
Your public key has been saved in sshfile.txt.pub
The key fingerprint is:
SHA256:TtahFL3QAHDZ4b7qhDHQqHnvxgCcSypZ3Frkgu9tBxg lenovo@LAPTOP-F00K5AP6
The key's randomart image is:
+---[RSA 3072]-----+
|
|..o++=
|o....o.o
|.o++.. o...
|o*E.+ o o..
|==o*o S .
|+o=..+ + .
|.. +o.. o
|..=...
|o.o
+-----[SHA256]-----+
PS C:\Users\lenovo>

```

**STEP3:** Now create a folder in a desired path and open it in visual studio code and type the following code.



```

C:\Users\lenovo> vpc_terraform.tf
10 resource "aws_subnet" "subsub1" {
11   vpc_id = aws_vpc.vpcnew.id
12   cidr_block = "10.0.1.0/24"
13 }
14
15 resource "aws_internet_gateway" "newgw" {
16   vpc_id = aws_vpc.vpcnew.id
17 }
18
19 resource "aws_route_table" "rnew" {
20   vpc_id = aws_vpc.vpcnew.id
21 }
22   route {
23     cidr_block = "0.0.0.0/0"
24     gateway_id = aws_internet_gateway.newgw.id
25   }
26 }
27
28 resource "aws_route_table_association" "rnew" {
29   subnet_id = aws_subnet.subsub1.id
30   route_table_id = aws_route_table.rnew.id
31 }

```

**STEP 4:** Now run the following command **“terraform init”** Terraform init command is used Initializes a Terraform working directory by downloading the necessary provider plugins and modules.

```

Windows PowerShell
PS C:\Users\lenovo> terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v4.61.0

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.

```

**STEP5:** Now after the terraform init command give the following command **“terraform plan”** terraform plan command is used generates an execution plan that shows what changes Terraform will make to your infrastructure. This is a useful command to run before applying changes to your infrastructure.

The screenshot displays the Visual Studio Code editor with a Terraform configuration file named `vpc.tf` open. The file defines an AWS VPC resource. The terminal window shows the output of the `terraform plan` command, indicating that the `aws_instance.sushma[0]` resource will be created. The plan details the configuration for the instance, including its AMI, architecture, availability zone, and various system settings.

```

File Edit Selection View Go Run Terminal Help
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

EXPLORER
  OPEN EDITORS
    ter_file.tf C:\Users\HP\
    X vpc.tf C:\Users\HP\
  NO FOLDER OPENED
  OUTLINE
  TIMELINE

vpc.tf
C:\Users\HP> vpc.tf
1 provider "aws" {
2     region = "us-east-1"
3     alias = "vpc"
4 }
5
6 resource "aws_vpc" "vpcnew" {
7     cidr_block = "10.0.0.0/16"
8 }

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\HP> terraform plan -out terraform.out
aws_key_pair.key.name: Refreshing state... [id=keykey]
aws_instance.sushma[0]: Refreshing state... [id=i-0e4716e0c1c511b12]
aws_instance.sushma[1]: Refreshing state... [id=i-0f1f9f0e0c63f10ee]

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

Terraform will perform the following actions:

# aws_instance.sushma[0] will be created
+ resource "aws_instance" "sushma" {
+   ami              = "ami-04581fbf744a7d11f"
+   arch             = (known after apply)
+   associate_public_ip_address = (known after apply)
+   availability_zone = (known after apply)
+   cpu_core_count   = (known after apply)
+   cpu_threads_per_core = (known after apply)
+   disable_api_stop = (known after apply)
+   disable_api_termination = (known after apply)
+   ebs_optimized    = (known after apply)
+   get_password_data = false
+   host_id          = (known after apply)
+   host_resource_group_arn = (known after apply)
+   iam_instance_profile = (known after apply)
+   id               = (known after apply)
+   instance_initiated_shutdown_behavior = (known after apply)
+   instance_state   = (known after apply)
+   instance_type     = "t3.micro"
  
```

**STEP6:** Now after the terraform plan command give the following command **“terraform apply”**  
Applies the changes specified in the Terraform configuration to your infrastructure.

The image shows a Windows 10 desktop with a Visual Studio Code editor window open. The editor is displaying a Terraform configuration file named 'ter.tf' located at 'C:\Users\JP>'. The file content is as follows:

```
provider "aws" {
  region = "us-east-1"
  access_key = "AKIA27Z2BN4GNCQW5H9"
  secret_key = "D8eghu0j+alHy9wulp4RactN/95q26GfgUAZ05"
}
```

```
public key      = "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQGCgEfghvZG9s0L2mcqel0r1/c00N924w/vwU/4P0ck0ds/48g80V0T2qC9XAAwHMEH3wE0R18qXA1pggc1C3MLD/wYtHw
Fufr/VXU0pQdU7W2w2p4PhYtYCEp3f4f8F/0007ZU02q20v/mcAMH6G-105vH82ds8Q8T5tkU/6S00w9w/1lp0ubk1/1tn+7Uefbts0wA75f4f99g3/0j1d8BAaek1b4dvC1B1lwuk1g6Q0Z0g/uaden
6rUkq05V8m+350s8F3NBu0G8E7CQ2dA0L0X06+0kexST+P0058G2kg4qk1u1xpVY60Zvcadd0Q06TVYH0T99Kv5EQw0WZCvZV8e0yELA6eaJ10J09su/v90U0VPcc5X8C//fuJ8kqP8agge1Sh0cS
9F0zV6sG5K1Sed0x8u8m/5l0n891+0d1q20n/6n6F9G50p0x20R5u5r- hpLAP0T-0F35H3L"
tags_all       = (known after apply)
```

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

Saved the plan to: tfplan

To perform exactly these actions, run the following command to apply:

```
terraform apply "tfplan"
```

```
PS C:\Users\JP> terraform apply "tfplan"
aws_key_pair.key.name: Creating...
aws_instance.sushma[1]: Creating...
aws_instance.sushma[0]: Creating...
aws_instance.sushma[1]: Still creating... [10s elapsed]
aws_instance.sushma[0]: Still creating... [10s elapsed]
aws_key_pair.key.name: Still creating... [10s elapsed]
aws_key_pair.key.name: Creation complete after 11s [1d:keykey]
aws_instance.sushma[0]: Still creating... [20s elapsed]
aws_instance.sushma[1]: Still creating... [20s elapsed]
aws_instance.sushma[1]: Still creating... [31s elapsed]
aws_instance.sushma[0]: Still creating... [31s elapsed]
aws_instance.sushma[0]: Creation complete after 35s [id=i-0e4714ee1c51b1b2]
aws_instance.sushma[1]: Creation complete after 35s [id=i-0f19fd0ec63f10ee]
```

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.

```
PS C:\Users\JP>
```

The status bar at the bottom of the Visual Studio Code window shows 'Restricted Mode' and 'In 9 Col 22 Spaces 4 UTF-8 CR LF Java'.

## REPOSITORY DETAILS

**main.tf:** This file contains the main Terraform code that defines the VPC, subnets, route tables, and other resources.

**variables.tf:** This file defines the input variables that are used in the main.tf file. This allows you to customize the VPC configuration, such as the VPC CIDR block or the number of subnets.

**outputs.tf:** This file defines the output variables that are generated by the main.tf file, such as the VPC ID or the public and private subnets.

**terraform.tfvars:** This file contains the values for the input variables defined in variables.tf.

**provider.tf:** This file specifies the provider details, such as AWS credentials and region.

**modules/:** This directory contains any Terraform modules used in the project, such as a module for creating EC2 instances.

**main.tf:** This file contains the main Terraform code that defines the VPC, subnets, route tables, and other resources.

## OUTCOME

The outcome of a VPC using Terraform project is a fully functional and customizable VPC infrastructure in Amazon Web Services (AWS). With the help of Terraform, you can quickly and easily create and manage VPC resources, including subnets, route tables, security groups, and EC2 instances. This allows you to set up a secure and isolated network environment that is tailored to the needs of your application. By using Terraform to define and manage your VPC infrastructure as code, you can ensure consistency and repeatability, as well as easy collaboration among team members. Moreover, Terraform enables you to easily update, modify, or delete VPC resources, and provides a clear view of the entire infrastructure state. Overall, the outcome of a VPC using Terraform project is a reliable, scalable, and maintainable VPC infrastructure that can support your application's growth and evolution.

aws

Services

Search

[Alt+S]

Oregon

pushpamoulika

VPC dashboard

EC2 Global View

Filter by VPC:

Select a VPC

Virtual private cloud

Your VPCs

Subnets

Route tables

Internet gateways

Egress-only internet gateways

Carrier gateways

DHCP option sets

Elastic IPs

Managed prefix lists

Endpoints

Endpoint services

NAT gateways

Peering connections

Security

Network ACLs

Your VPCs (2)

Info

Filter VPCs

Actions

Create VPC

	Name	VPC ID	State	IPv4 CIDR	IPv6 CIDR	DHC
<input type="checkbox"/>	-	vpc-0fa9f44cf4f843b70	Available	10.0.0.0/16	-	dopt
<input type="checkbox"/>	-	vpc-067051e6fddec25bc	Available	172.31.0.0/16	-	dopt

Select a VPC above

https://us-west-2.console.aws.amazon.com/vpc/home?region=us-west-2#vpcs

© 2023, Amazon Web Services India Private Limited or its affiliates. Privacy Terms Cookie preferences

aws

Services

Search

[Alt+S]

Oregon

pushpamoulika

VPC dashboard

EC2 Global View

Filter by VPC:

Select a VPC

Virtual private cloud

Your VPCs

Subnets

Route tables

Internet gateways

Egress-only internet gateways

Carrier gateways

DHCP option sets

Elastic IPs

Managed prefix lists

Endpoints

Endpoint services

NAT gateways

Peering connections

Security

Network ACLs

Subnets (1/5)

Info

Filter subnets

Actions

Create subnet

	Name	Subnet ID	State	VPC	IPv4 CIDR	IPv6 CIDR
<input type="checkbox"/>	-	subnet-00c1d06d78a24669a	Available	vpc-067051e6fddec25bc	172.31.0.0/20	-
<input type="checkbox"/>	-	subnet-03f978835253011c4	Available	vpc-067051e6fddec25bc	172.31.32.0/20	-
<input type="checkbox"/>	-	subnet-0458d0f0450d07906	Available	vpc-067051e6fddec25bc	172.31.16.0/20	-
<input checked="" type="checkbox"/>	-	subnet-03dd7f41f03586e16	Available	vpc-0fa9f44cf4f843b70	10.0.1.0/24	-
<input type="checkbox"/>	-	subnet-0b1e7d64e023c9b6a	Available	vpc-067051e6fddec25bc	172.31.48.0/20	-

subnet-03dd7f41f03586e16

Details

Flow logs

Route table

Network ACL

CIDR reservations

Sharing

Tags

Details

Subnet ID

subnet-03dd7f41f03586e16

Available IPv4 addresses

Subnet ARN

arn:aws:ec2:us-west-2:317267557348:subnet/subnet-03dd7f41f03586e16

State

Available

Availability Zone

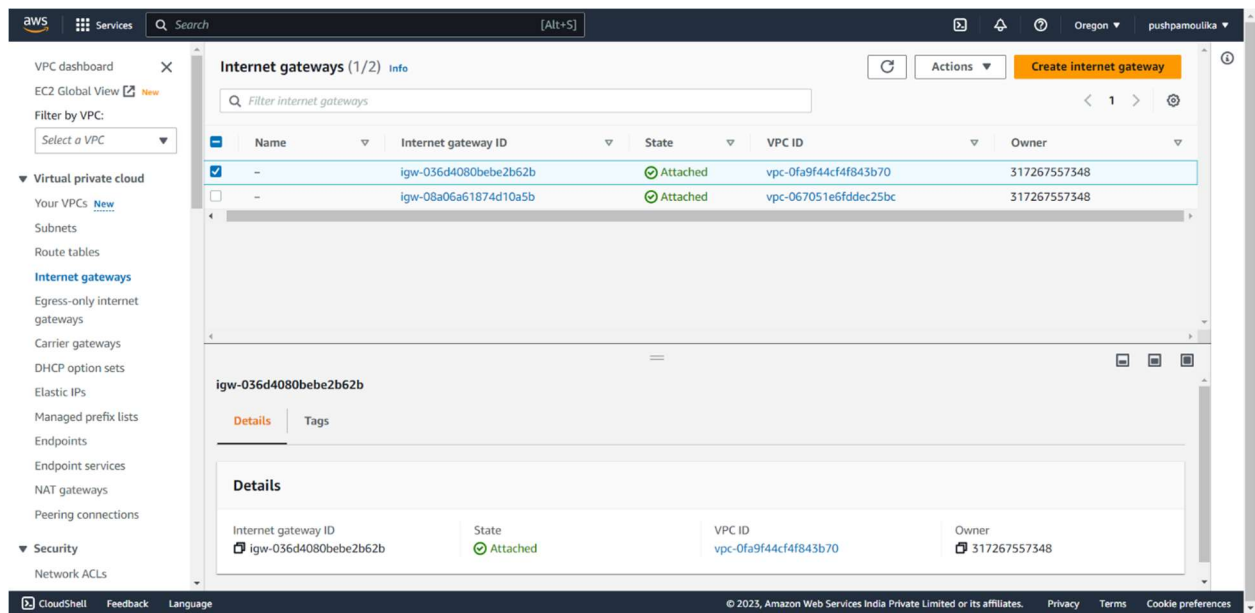
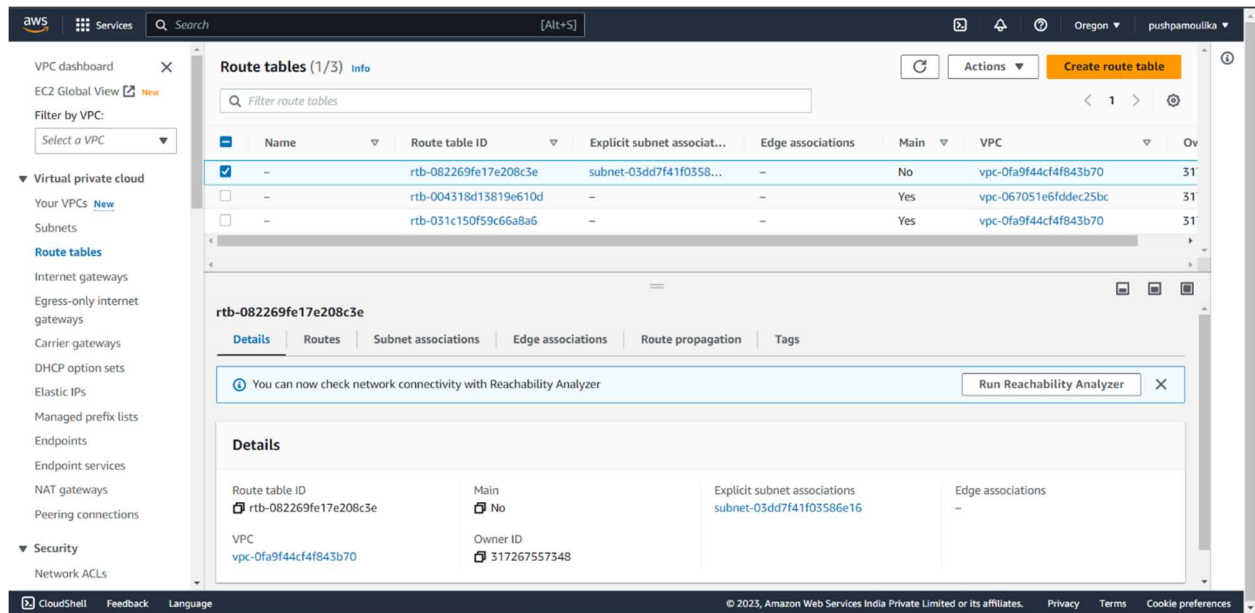
IPv4 CIDR

10.0.1.0/24

Availability Zone ID

CloudShell Feedback Language

© 2023, Amazon Web Services India Private Limited or its affiliates. Privacy Terms Cookie preferences



At the end of the VPC using Terraform project, you should have a fully provisioned and functional VPC infrastructure in your chosen cloud provider (such as AWS, Azure, or Google Cloud Platform). This infrastructure should include all necessary resources such as VPC, subnets, routing tables, security groups, and any additional resources required for your application.

Once the infrastructure is set up, you can deploy your application and start using the VPC to host your application. You can also continue to use Terraform to manage and update the infrastructure over time, making it easier to scale and maintain your application.

The final outcome of the VPC using Terraform project is a reliable and flexible network infrastructure that enables your application to run efficiently and securely in the cloud.

## **CONCLUSION**

In conclusion, using Terraform for creating and managing a Virtual Private Cloud (VPC) in Amazon Web Services (AWS) offers many benefits, such as improved infrastructure consistency, reduced human error, and increased automation. Terraform allows you to define your infrastructure as code, enabling you to rapidly and consistently create and manage your cloud infrastructure. With the help of Terraform, you can easily create and configure VPC resources, including subnets, route tables, security groups, and EC2 instances. Moreover, Terraform enables you to easily update, modify, or delete VPC resources, and provides a clear view of the entire infrastructure state. As such, Terraform is a powerful tool that can significantly enhance the management and provisioning of VPCs in AWS, making it an essential technology for cloud infrastructure management.