**Terraform:**

<https://developer.hashicorp.com/terraform/downloads>

<https://antonputra.com/amazon/create-aws-eks-fargate-using-terraform/>

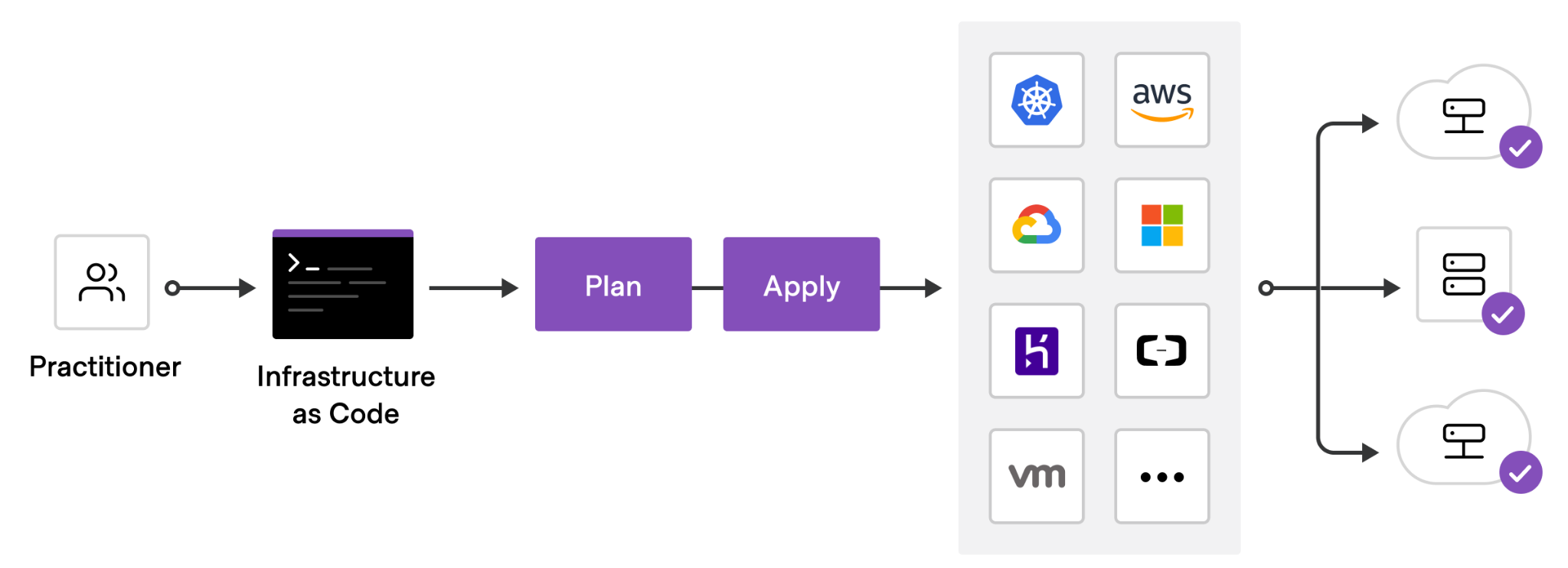
**/\* Terraform is an infrastructure as code tool that lets you build, change, and version cloud and on-prem resources safely and efficiently \*/**

Terraform is HashiCorp's infrastructure as code tool. It lets you define resources and infrastructure in human-readable, declarative configuration files, and manages your infrastructure's lifecycle. Using Terraform has several advantages over manually managing your infrastructure:

* Terraform can manage infrastructure on multiple cloud platforms.
* The human-readable configuration language helps you write infrastructure code quickly.
* Terraform's state allows you to track resource changes throughout your deployments.
* You can commit your configurations to version control to safely collaborate on infrastructure.

To deploy infrastructure with Terraform:

* Scope - Identify the infrastructure for your project.
* Author - Write the configuration for your infrastructure.
* Initialize - Install the plugins Terraform needs to manage the infrastructure.
* Plan - Preview the changes Terraform will make to match your configuration.
* Apply - Make the planned changes.



## **Track your infrastructure.**

Terraform keeps track of your real infrastructure in a **state file,** which acts as a source of truth for your environment. Terraform uses the state file to determine the changes to make to your infrastructure so that it will match your configuration.

**STEP1: Install Terraform**

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

# echo "deb [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

**/\* Installation part is over, Now go to the project\*/**

* **Next, aws configure and cross check**

# cd .aws/

# cat credentials

# aws configure --profile raghava ( AWS Configure for new profiles)

# cd .aws

# cat credentials

**Actual Project flow:**

Reference for code: [**https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/s3\_bucket**](https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/s3_bucket)

**STEP2: Create a “project” directory and inside that directory Initialise terraform**.

# mkdir myproject

# terraform init

* Create Global Directory ->tfstate\_s3 & tflock\_ddb

# mkdir global

# cd global/

# mkdir tfstate\_s3

# mkdir tflock\_ddb

* Inside the tfstate\_s3 directory

# cd tfstate\_s3/

* Create the mentioned files with extension and write the below code.

# sudo touch vars.tf

# sudo touch main.tf

# sudo touch outputs.tf

**# sudo touch vars.tf**

variable "region" {

description = "The region into which the VPC is deployed"

type = string

default = "ap-south-1"

}

variable "profile" {

description = "profile to be used for the Authentication purpose."

type = string

default = "raghava"

}

variable "environment" {

description = "profile to be used for the Authentication purpose."

type = string

default = "dev"

}

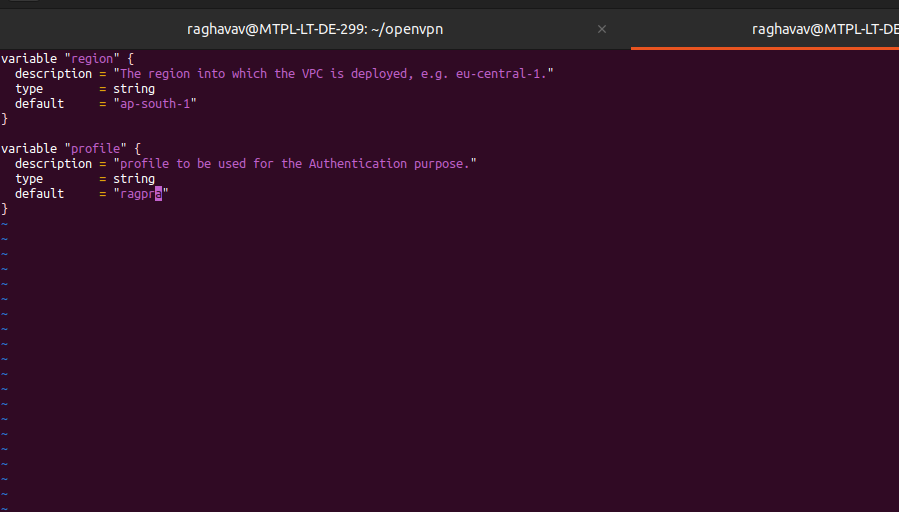
variable "project" {

description = "profile to be used for the Authentication purpose."

type = string

default = "poc"

}



**# sudo vim main.tf**

/\*

This script handles:

- Creation of base s3 bucket for terraform states

- So Create S3 bucket

\*/

terraform {

required\_providers {

aws = {

source = "hashicorp/aws"

version = "~> 4.0"

}

}

required\_version = ">= 1.2" #https://github.com/hashicorp/terraform/releases

}

provider "aws" {

region = var.region

profile = var.profile

}

resource "aws\_s3\_bucket" "terraform\_state\_bucket" {

bucket = "${var.project}-${var.environment}-tfstate"

tags = {

Name = "project-tfstate"

Environment = var.environment

}

}

resource "aws\_s3\_bucket\_public\_access\_block" "terraform\_state\_access" {

bucket = aws\_s3\_bucket.terraform\_state\_bucket.id

block\_public\_acls = true

block\_public\_policy = true

ignore\_public\_acls = true

restrict\_public\_buckets = true

}

resource "aws\_s3\_bucket\_versioning" "terraform\_state\_versioning" {

bucket = aws\_s3\_bucket.terraform\_state\_bucket.id

versioning\_configuration {

status = "Enabled"

}

}

resource "aws\_s3\_bucket\_server\_side\_encryption\_configuration" "terraform\_state\_encryption" {

bucket = aws\_s3\_bucket.terraform\_state\_bucket.bucket

rule {

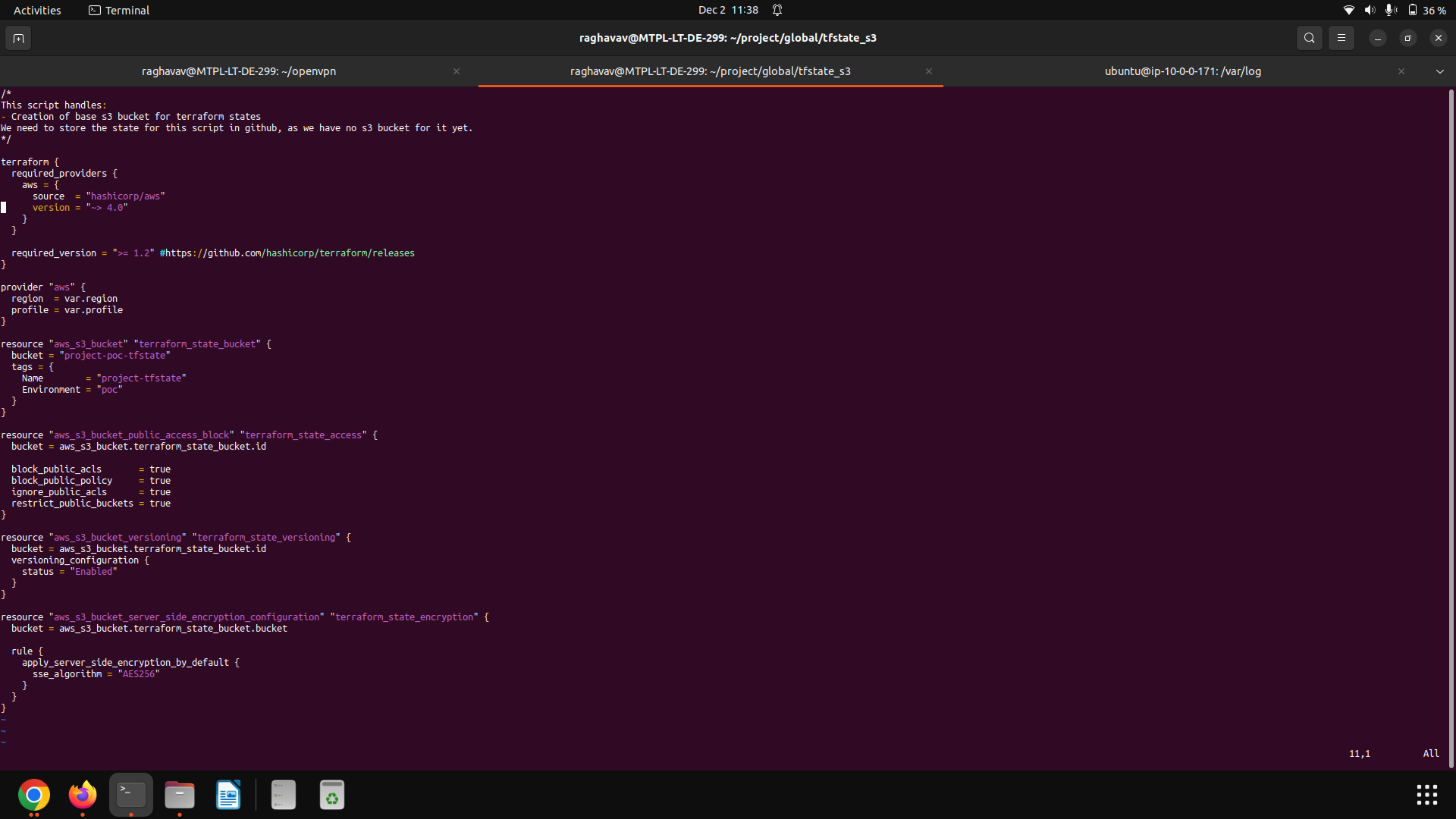
apply\_server\_side\_encryption\_by\_default {

sse\_algorithm = "AES256"

}

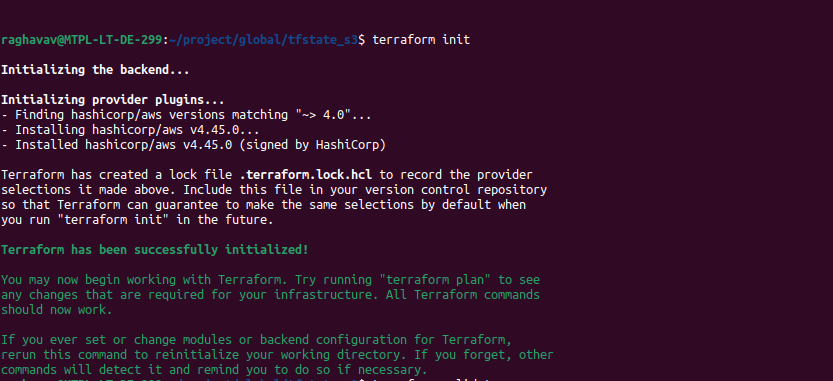
}

}



* Next, Terraform initialize, Validate, Plan and apply.

**# terraform init**



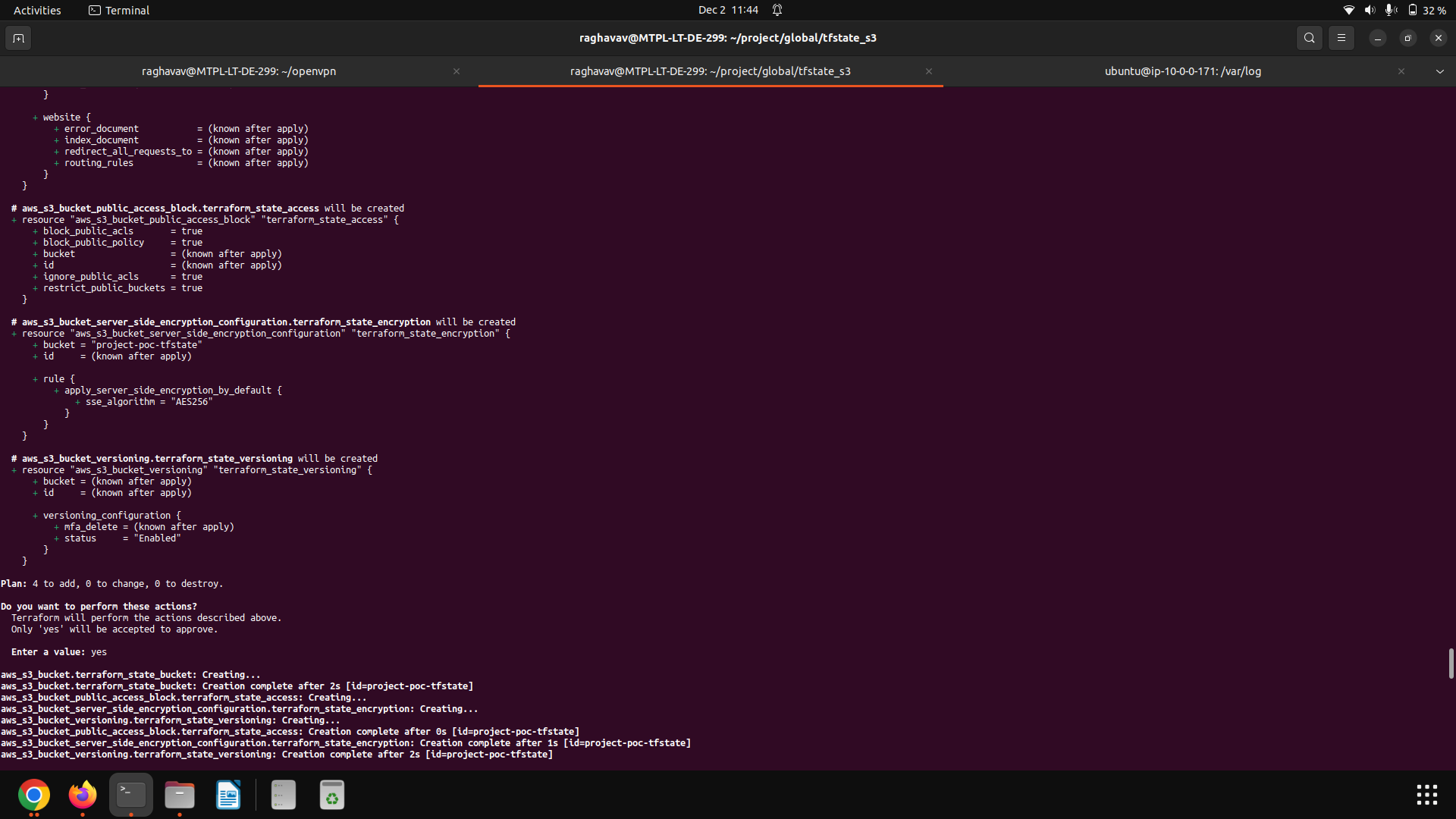
**# terraform validate**



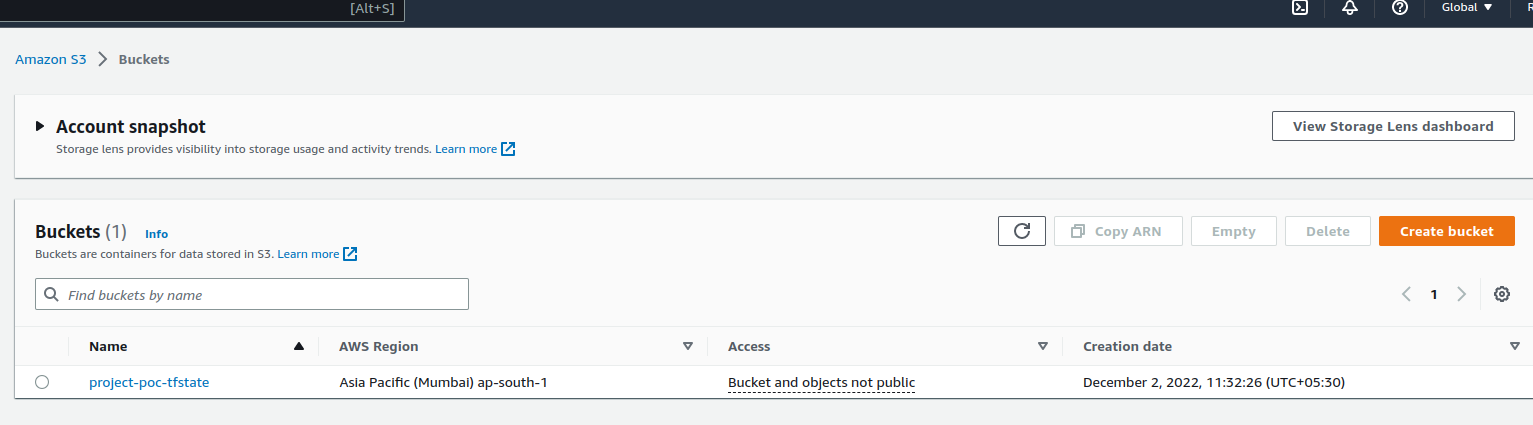
# **terraform fmt** —----- It will format/Indentation setup will take care.

**# terraform plan—----- List out the resources which we are going to create**

**# terraform init && terraform plan && terraform apply**



Output, S3 Created



**INFO: # terraform apply -auto-approve**

**STEP3: Now tflock file:**

\*\* Create tflock file and create Dynamodb Table for the tflock file\*\*\*

Which is used to lock the state of the Terraform apply, Plan\*\*\*

It helps to control the conflicts of Applying the terraform infra by different users………. Whenever the user apply/Plan it will read the state from DynamoDb table, if the state is locked it won’t run the apply command.  
  
  
**Next, Inside the tflock\_ddb directory create the files, mai.tf, vars.tf.**

# cd tflock\_ddb

# sudo touch vars.tf

# sudo touch main.tf

**# sudo vim vars.tf**

variable "region" {

description = "The region into which the VPC is deployed, e.g. eu-central-1."

type = string

default = "ap-south-1"

}

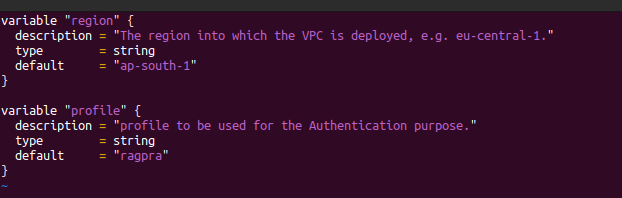
variable "profile" {

description = "profile to be used for the Authentication purpose."

type = string

default = "raghava"

}



**# sudo vim main.tf**

/\*

This script handles:

- Creation of a dynamo db table to acquire tf state locks for this environment

Attention: This cannot use locking yet!

\*/

terraform {

required\_providers {

aws = {

source = "hashicorp/aws"

version = "~> 4.0"

}

}

backend "s3" {

bucket = "project-poc-tfstate"

key = "global/tflock/terraform.tfstate"

region = "ap-south-1"

}

required\_version = ">= 1.2" #https://github.com/hashicorp/terraform/releases

}

provider "aws" {

region = var.region

profile = var.profile

}

resource "aws\_dynamodb\_table" "terraform\_locks" {

# Creates a dynamoDB table to store our tf state locks

name = "project-poc-tflock"

billing\_mode = "PAY\_PER\_REQUEST"

hash\_key = "LockID"

attribute {

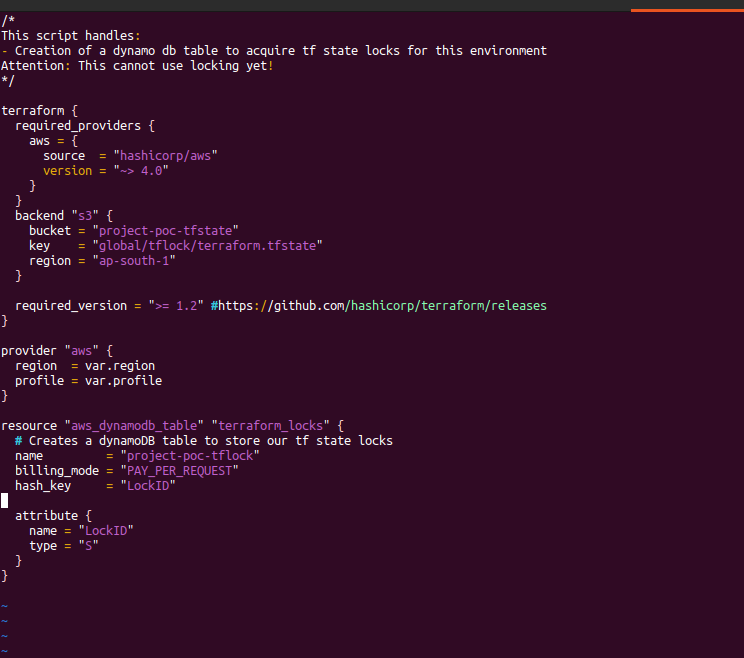
name = "LockID"

type = "S"

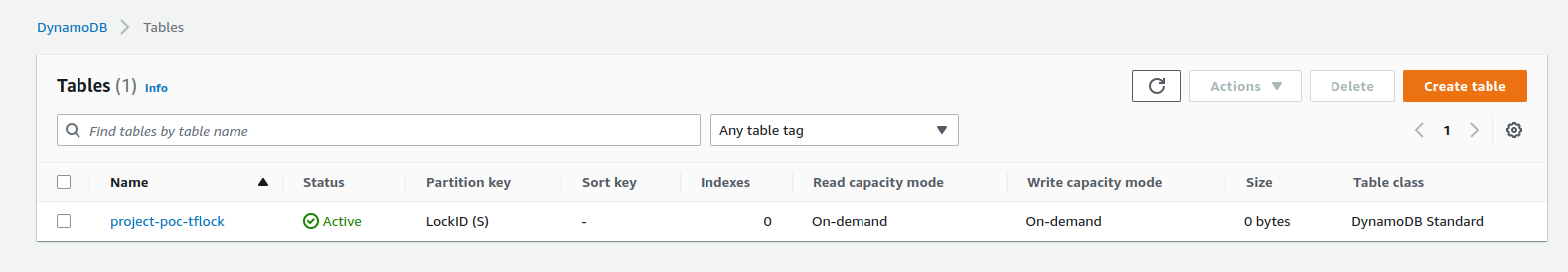
}

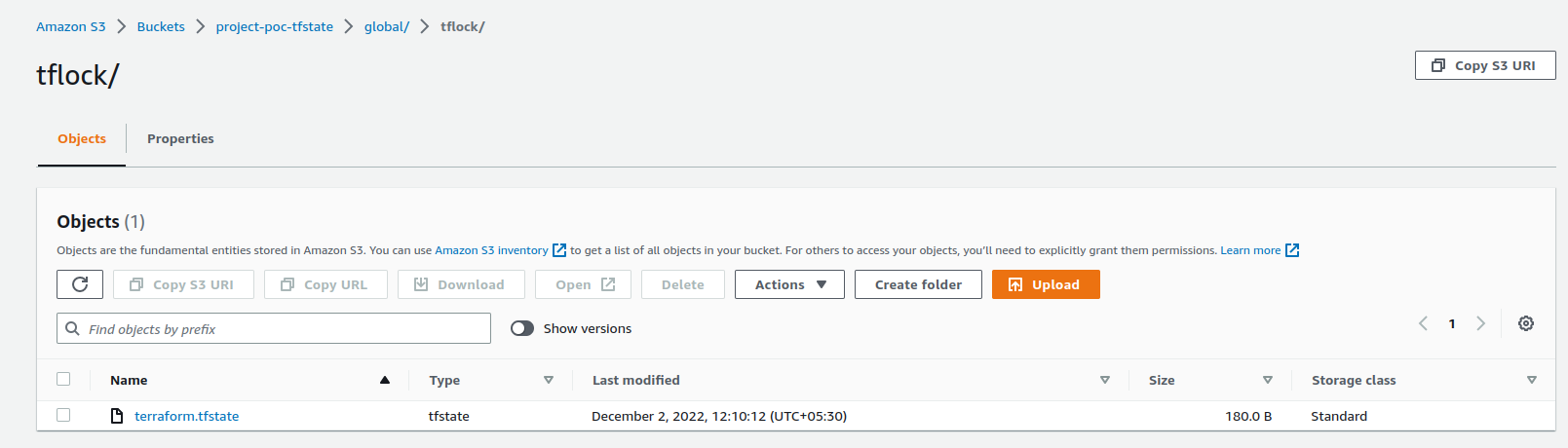
}

/\* NOTE: HASH\_KEY and Attribute NAME  
Partition key – **A simple primary key, composed of one attribute known as the partition key**. DynamoDB uses the partition key's value as input to an internal hash function. The output from the hash function determines the partition (physical storage internal to DynamoDB) in which the item will be stored. \*/



# terraform init && terraform plan && terraform apply



* Tfstate file in S3 bucket  
    
  

**STEP4: Now, Create a AWS Modules:**

Inside the project directory, Create a **“aws\_modules”** directory. ( Here we can keep all the aws resources which we are going to create further).

* Inside the **aws\_modules** directory create an individual directory for each resource. example: VPC, EC2, S3, EKS etc..
* Create a directory for resource “VPC”

# cd aws\_modules

# mkdir vpc

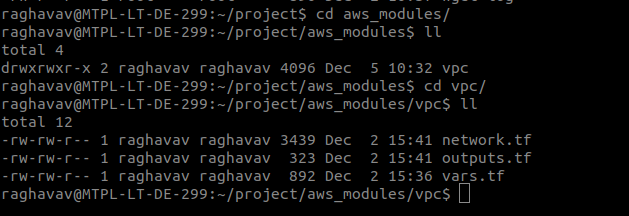
# cd vpc

* Inside the VPC directory, Create the files mentioned.

# touch network.tf

# touch vars.tf

# touch outputs.tf



**# sudo vim vars.tf**

############ GLOBAL PROJECT RELATED VARIABLES ############################

variable "environment" {

description = "Write here description for the variable."

type = string

default = ""

}

variable "project" {

type = string

default = ""

}

variable "costcentre" {

description = "Write description for the variable."

type = string

default = ""

}

variable "iac\_type" {

description = "Write description for the variable."

type = string

default = ""

}

#################### COMMON VARIABLES ############################################

variable "azs" {

type = string

default = ""

}

#################### VPC/NETWORK VARIABLES ############################

variable "vpc\_cidr" {

type = string

default = ""

}

variable "pub\_sub" {

type = string

default = ""

}

variable "pri\_sub" {

type = string

default = ""

}

**# sudo vim network.tf**

resource "aws\_vpc" "project\_vpc" {

cidr\_block = var.vpc\_cidr

enable\_dns\_hostnames = true

enable\_dns\_support = true

tags = {

"Name" = "${var.project}-${var.environment}-vpc"

"Environment" = var.environment

"CostCentre" = var.costcentre

}

}

resource "aws\_internet\_gateway" "igw" {

vpc\_id = aws\_vpc.project\_vpc.id

tags = {

"Name" = "${var.project}-${var.environment}-igw"

"Environment" = var.environment

"CostCentre" = var.costcentre

}

}

resource "aws\_subnet" "pub\_sub" {

count = length(var.azs)

vpc\_id = aws\_vpc.project\_vpc.id

cidr\_block = cidrsubnet(var.vpc\_cidr, 8, count.index)

availability\_zone = string(var.azs, count.index)

map\_public\_ip\_on\_launch = true

tags = {

"Name" = "${var.project}-${var.environment}-pubsub${count.index}"

"Environment" = var.environment

"CostCentre" = var.costcentre

}

}

resource "aws\_subnet" "pri\_sub" {

count = length(var.azs)

vpc\_id = aws\_vpc.project\_vpc.id

cidr\_block = cidrsubnet(var.vpc\_cidr, 8, count.index + 10)

availability\_zone = string(var.azs, count.index)

map\_public\_ip\_on\_launch = false

tags = {

"Name" = "${var.project}-${var.environment}-prisub${count.index}"

"Environment" = var.environment

"CostCentre" = var.costcentre

}

}

resource "aws\_eip" "nat\_eip" {

vpc = true

depends\_on = [

aws\_internet\_gateway.igw

]

tags = {

"Name" = "${var.project}-${var.environment}-nateip"

"Environment" = var.environment

"CostCentre" = var.costcentre

}

}

resource "aws\_nat\_gateway" "nat-gtw" {

allocation\_id = aws\_eip.nat\_eip.id

subnet\_id = aws\_subnet.pub\_sub[0].id

connectivity\_type = "public"

depends\_on = [

aws\_internet\_gateway.igw

]

tags = {

"Name" = "${var.project}-${var.environment}-natgw"

"Environment" = var.environment

"CostCentre" = var.costcentre

}

}

resource "aws\_route\_table" "pub-rt" {

vpc\_id = aws\_vpc.project\_vpc.id

tags = {

"Name" = "${var.project}-${var.environment}-pubrt"

"Environment" = var.environment

"CostCentre" = var.costcentre

}

}

resource "aws\_route" "pub-route" {

route\_table\_id = aws\_route\_table.pub-rt.id

destination\_cidr\_block = "0.0.0.0/0"

gateway\_id = aws\_internet\_gateway.igw.id

}

resource "aws\_route\_table\_association" "pub-rta" {

count = 3

route\_table\_id = aws\_route\_table.pub-rt.id

subnet\_id = aws\_subnet.pub\_sub.\*.id[count.index]

}

resource "aws\_route\_table" "pri-rt" {

vpc\_id = aws\_vpc.project\_vpc.id

tags = {

"Name" = "${var.project}-${var.environment}-prirt"

"Environment" = var.environment

"CostCentre" = var.costcentre

}

}

resource "aws\_route" "pri-rt" {

count = 1

route\_table\_id = aws\_route\_table.pri-rt.id

destination\_cidr\_block = "0.0.0.0/0"

nat\_gateway\_id = aws\_nat\_gateway.nat-gtw.id

}

resource "aws\_route\_table\_association" "pri-rta" {

count = 3

route\_table\_id = aws\_route\_table.pri-rt.id

subnet\_id = aws\_subnet.pri\_sub.\*.id[count.index]

}

/\*

resource "aws\_flow\_log" "vpc\_flowlogs" {

log\_destination = "${var.project}-${var.environment}-objects"

log\_destination\_type = "s3"

traffic\_type = "ALL"

vpc\_id = aws\_vpc.project\_vpc.id

}\*/

**# sudo vim outputs.tf**

output "vpc\_id" {

value = aws\_vpc.project\_vpc.id

description = "The ID of the VPC."

}

output "pub\_subIDs" {

value = aws\_subnet.pub\_sub.\*.id

description = "The ID of the Public subnets."

}

output "pri\_subIDs" {

value = aws\_subnet.pri\_sub.\*.id

description = "The ID of the Private subnets."

}

**/\* We have created VPC module, Like VPC module we can create whatever the modules we need \*/**

**STEP5:** Now, Create a **“dev\_env”** directory inside the Project directory **\*/**

Create a file mentioned below.

# touch main.tf

# touch vars.tf

# touch outputs.tf

**# sudo vim vars.tf**

#################### DECLARE ENVIRONMENT SPECIFIC VARIABLES ############################

variable "environment" {

description = "Dev environment for testing."

type = string

default = "poc"

}

variable "AWS\_REGION" {

type = string

default = "ap-south-1"

}

variable "azs" {

type = list(string)

default = ["ap-south-1a", "ap-south-1c", "ap-south-1b"]

}

variable "vpc\_cidr" {

description = "Write description for the variable."

type = string

default = "172.16.0.0/16"

}

variable "pub-sub" {

type = list(string)

default = [

"172.16.0.0/24",

"172.16.1.0/24",

"172.16.2.0/24",

"172.16.3.0/24"

]

}

variable "pri-sub" {

type = list(string)

default = [

"172.16.10.0/24",

"172.16.11.0/24",

"172.16.12.0/24",

"172.16.13.0/24"

]

}

**# sudo vim main.tf**

/\*

######################## Using S3 and DynamoDB for backend + locking ##################

terraform {

backend "s3" {

bucket = "project-poc-tfstate"

key = "tfstate/vpc/terraform.tfstate"

region = "ap-south-1"

dynamodb\_table = "project-poc-tflock"

encrypt = true

}

required\_version = ">= 1.2"

required\_providers {

aws = {

version = "~> 4.0"

}

}

}

\*/

provider "aws" {

region = "${var.AWS\_REGION}"

}

######################## Independent services (Create before Dependent services) ##################

module "aws\_vpc" {

source = "../aws\_modules/vpc"

vpc\_cidr = "${vpc\_cidr}"

iac\_type = "terraform"

project = "dev"

environment = "dev"

costcentre = "raghava"

pub\_sub = "${pub-sub}”

pri\_sub = “${pri-sub}”

}

**Terraform Commands:**

# terraform init ( Install plugins)

# terraform init -upgrade ( to upgrade plugins)

# terraform plan --var-file=../../variables/vpc/bizom.tf ***–auto-approve***

( To Skip Interactive Approval)

# terraform destroy --var-file=../../variables/vpc/bizom.tf

# terraform destroy -target='resource.name’

# terraform apply --var-file=../../variables/vpc/bizom.tf

## **IAM OIDC provider Using Terraform**

You can associate an **IAM role** with a **Kubernetes service account**. This service account can then provide **AWS permissions** to the containers in any pod that uses that service account. With this feature, you no longer need to **provide extended permissions** to all Kubernetes nodes so that pods on those nodes can **call AWS APIs**.

data "tls\_certificate" "eks" {

url = aws\_eks\_cluster.cluster.identity[0].oidc[0].issuer

}

resource "aws\_iam\_openid\_connect\_provider" "eks" {

client\_id\_list = ["sts.amazonaws.com"]

thumbprint\_list = [data.tls\_certificate.eks.certificates[0].sha1\_fingerprint]

url = aws\_eks\_cluster.cluster.identity[0].oidc[0].issuer

}

# aws iam list-open-id-connect-providers

