**POC - INFRASTRUCTURE BUILD WITH TERRAFORM ON MULTI CLOUD**

**AWS CLOUD**

* **Setting Up Amazon EKS Cluster with Terraform**

**Overview:**

This poc provides a step-by-step guide for setting up an Amazon EKS (Elastic Kubernetes Service) cluster using Terraform. The setup includes configuring VPC, security groups, and the EKS cluster itself, and outputs key information needed for cluster management.

**Prerequisites:**

1. **AWS Account**: Ensure you have an AWS account with the necessary permissions to create and manage resources.
2. **Terraform**: Install Terraform version >= 1.3.2.
3. **AWS CLI**: Install and configure the AWS CLI on your local machine.

**Terraform Configuration**

1. **Provider and Version Requirements:** The Terraform configuration requires specific versions of providers to ensure compatibility.

terraform {

required\_version = ">= 1.3.2"

required\_providers {

random = {

source = "hashicorp/random"

version = "~> 3.1.0"

}

kubernetes = {

source = "hashicorp/kubernetes"

version = ">= 2.7.1"

}

aws = {

source = "hashicorp/aws"

version = ">= 3.68.0"

}

local = {

source = "hashicorp/local"

version = "~> 2.1.0"

}

null = {

source = "hashicorp/null"

version = "~> 3.1.0"

}

cloudinit = {

source = "hashicorp/cloudinit"

version = "~> 2.2.0"

}

}

}

1. Variables: Define the variables used in the configuration

variable "kubernetes\_version" {

default = 1.27

description = "Kubernetes version"

}

variable "vpc\_cidr" {

default = "10.0.0.0/16"

description = "CIDR range of the VPC"

}

variable "aws\_region" {

default = "us-east-1"

description = "AWS region"

}

1. VPC configuration: Define vpc and subnets resources.

module "vpc" {

source = "terraform-aws-modules/vpc/aws"

version = "5.9.0"

name = "yash-eks-vpc"

cidr = var.vpc\_cidr

azs = data.aws\_availability\_zones.available.names

private\_subnets = ["10.0.1.0/24", "10.0.2.0/24"]

public\_subnets = ["10.0.101.0/24", "10.0.102.0/24"]

enable\_nat\_gateway = true

enable\_vpn\_gateway = true

tags = {

Terraform = "true"

Environment = "dev"

}

}

1. Security Groups: configuring security groups and rules.

resource "aws\_security\_group" "all\_worker\_mgmt" {

name\_prefix = "all\_worker\_management"

vpc\_id = module.vpc.vpc\_id

}

resource "aws\_security\_group\_rule" "all\_worker\_mgmt\_ingress" {

description = "Allow inbound traffic from EKS"

from\_port = 0

protocol = "-1"

to\_port = 0

security\_group\_id = aws\_security\_group.all\_worker\_mgmt.id

type = "ingress"

cidr\_blocks = [

"10.0.0.0/8",

"172.16.0.0/12",

"192.168.0.0/16",

]

}

resource "aws\_security\_group\_rule" "all\_worker\_mgmt\_egress" {

description = "Allow outbound traffic to anywhere"

from\_port = 0

protocol = "-1"

to\_port = 0

security\_group\_id = aws\_security\_group.all\_worker\_mgmt.id

type = "egress"

cidr\_blocks = ["0.0.0.0/0"]

}

1. EKS Cluster Configuration: Define the EKS cluster and associated configurations.

module "eks" {

source = "terraform-aws-modules/eks/aws"

version = "20.20.0"

cluster\_name = local.cluster\_name

cluster\_version = var.kubernetes\_version

enable\_irsa = true

cluster\_endpoint\_public\_access = true

cluster\_addons = {

coredns = {}

eks-pod-identity-agent = {}

kube-proxy = {}

vpc-cni = {}

}

tags = {

cluster = "eks-terraform"

}

vpc\_id = module.vpc.vpc\_id

subnet\_ids = module.vpc.private\_subnets

eks\_managed\_node\_group\_defaults = {

ami\_type = "AL2\_x86\_64"

instance\_types = ["t3.large"]

capacity\_type = "ON\_DEMAND"

vpc\_security\_group\_ids = [aws\_security\_group.all\_worker\_mgmt.id]

}

eks\_managed\_node\_groups = {

node\_group1 = {

min\_size = 1

max\_size = 5

desired\_size = 2

}

}

enable\_cluster\_creator\_admin\_permissions = true

access\_entries = {

example = {

kubernetes\_groups = ["node\_group1"]

principal\_arn = "arn:aws:iam::088285363738:role/Eks\_cluster\_role"

policy\_associations = {

example = {

policy\_arn = "arn:aws:eks::aws:cluster-access-policy/AmazonEKSViewPolicy"

access\_scope = {

namespaces = ["default"]

type = "namespace"

}

}

}

}

}

}

1. Outputs: Define outputs for accessing key information.

output "cluster\_id" {

description = "EKS cluster ID."

value = module.eks.cluster\_id

}

output "cluster\_endpoint" {

description = "Endpoint for EKS control plane."

value = module.eks.cluster\_endpoint

}

output "cluster\_security\_group\_id" {

description = "Security group ids attached to the cluster control plane."

value = module.eks.cluster\_security\_group\_id

}

output "region" {

description = "AWS region"

value = var.aws\_region

}

output "oidc\_provider\_arn" {

value = module.eks.oidc\_provider\_arn

}

output "update\_kubeconfig\_command" {

description = "Command to update kubeconfig for EKS cluster."

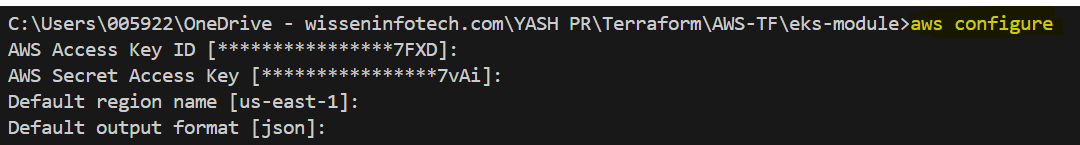
value = format("aws eks update-kubeconfig --name %s --region %s", module.eks.cluster\_id, var.aws\_region)

}

**Steps to Deploy:**

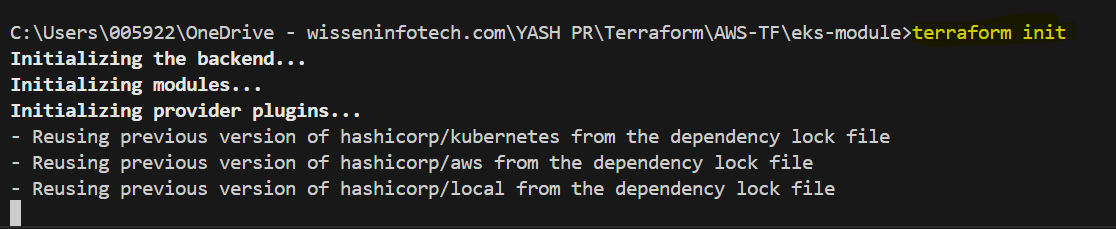
1. To interact AWS account with AWS CLI

$aws configure



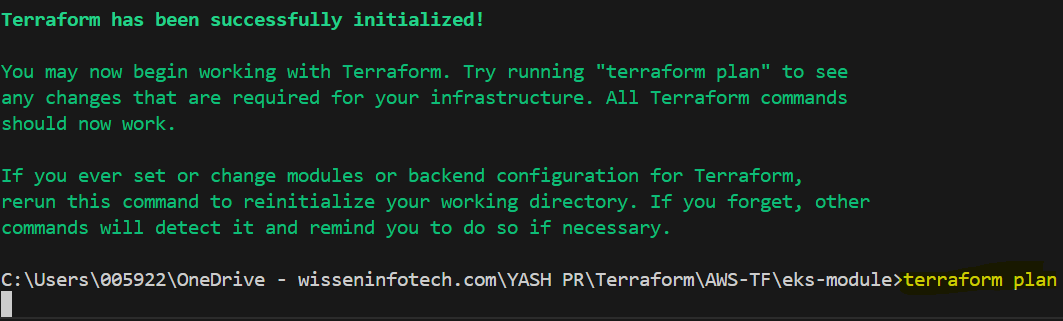
1. To initialize the working directory and download required providers.

$ terraform init



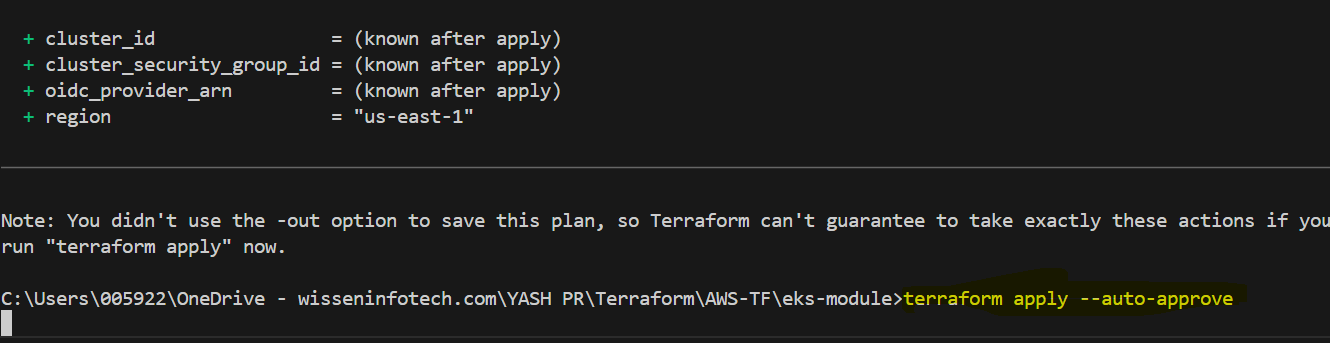
1. Execute terraform plan to see a preview of the changes Terraform will make.

$ terraform plan

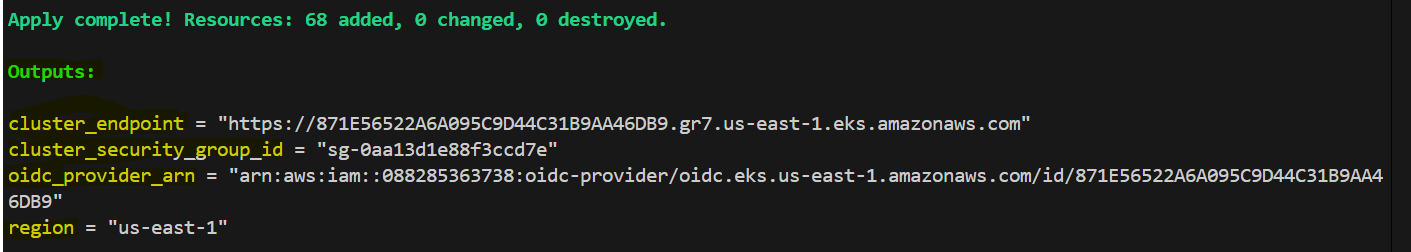


1. Apply the configuration to create the resources.

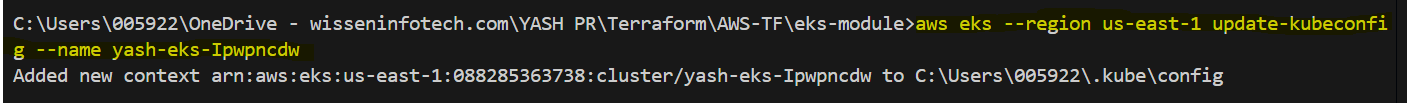
$ terraform apply



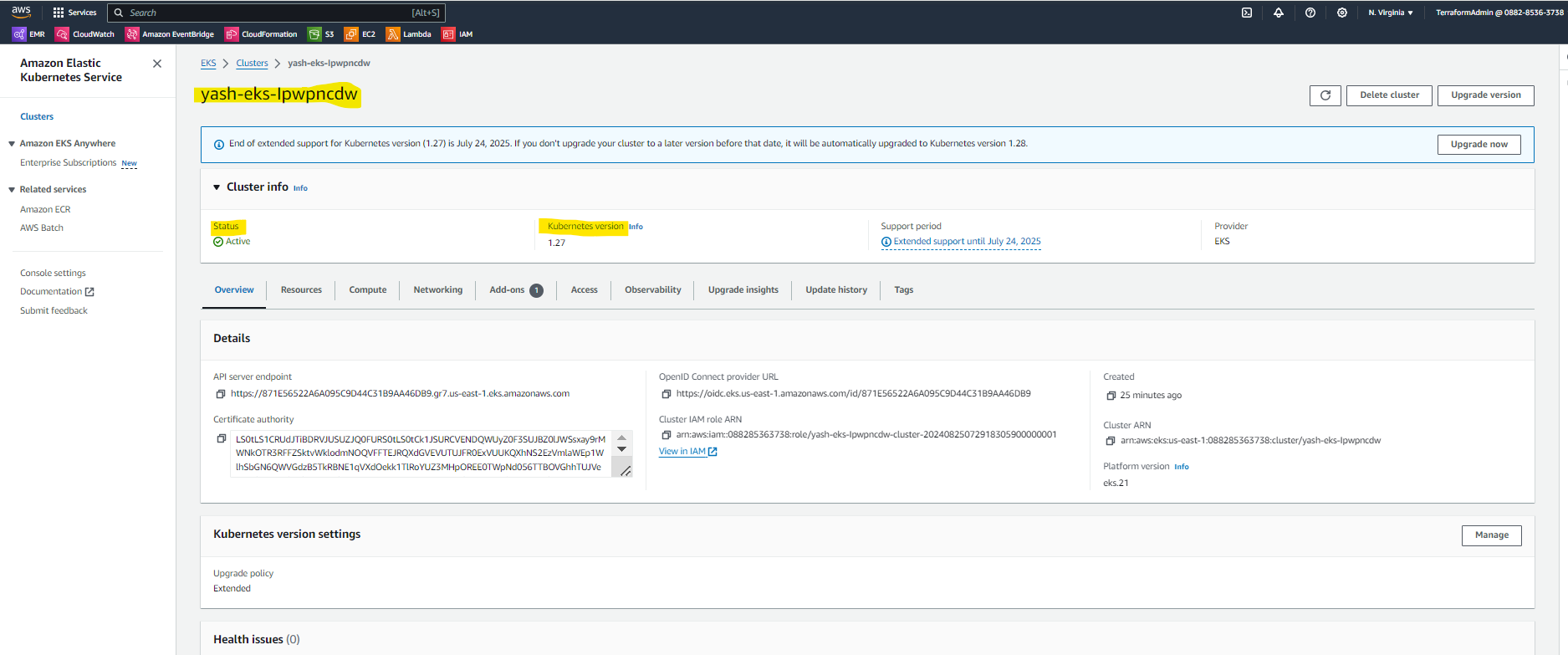
1. Use the output command to update your kubeconfig for kubectl.



$ aws eks update-kubeconfig --name <cluster\_id> --region <aws\_region>

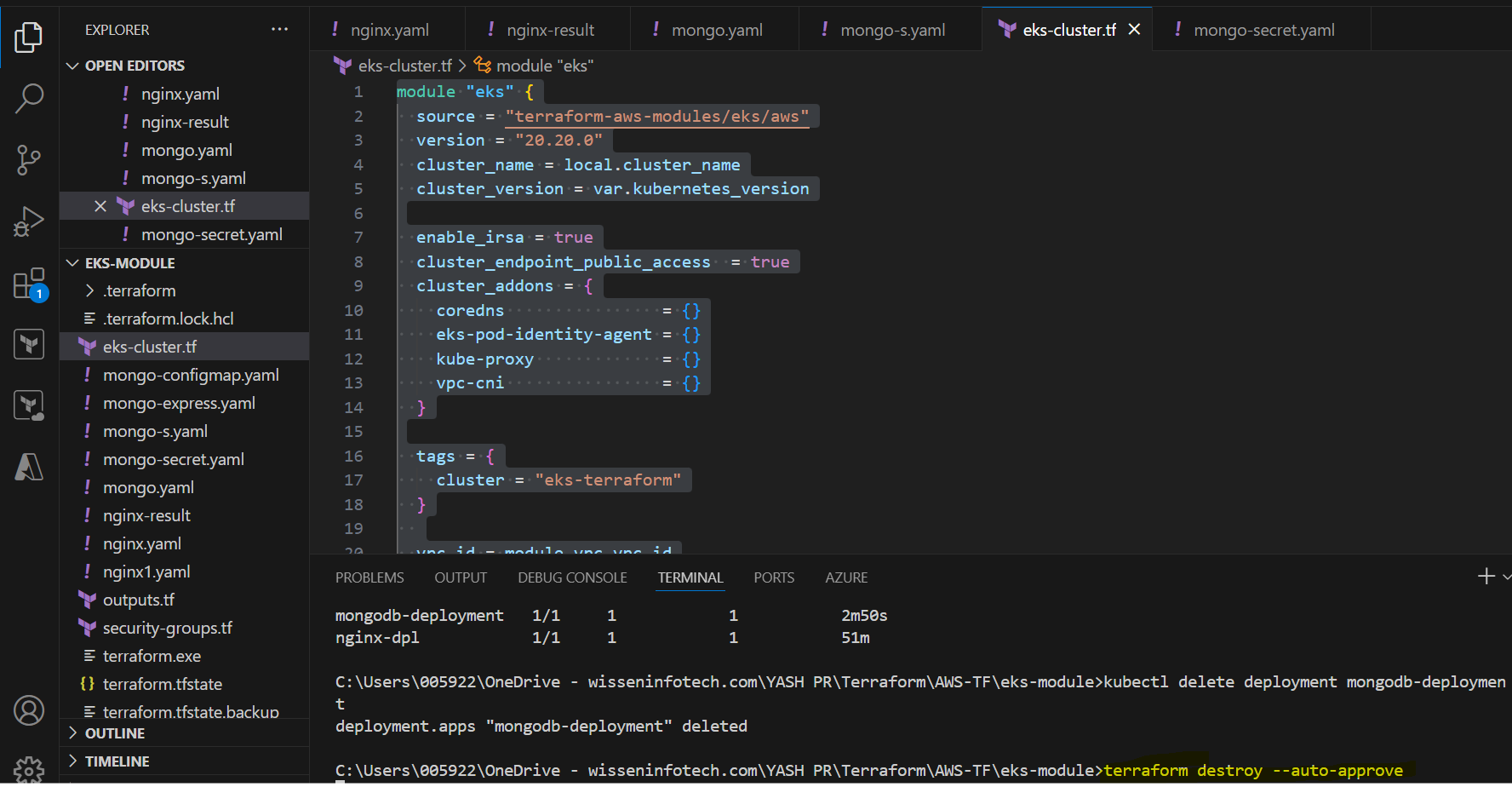


Login to AWS Management Console: EKS Cluster was created successfully using terraform templates.

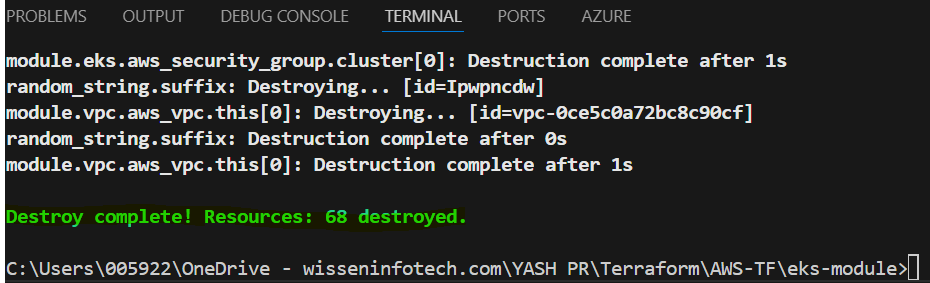


1. To destroy all resources which is created by using destroy command (–auto-approve) gives you the auto approval to deploy the resources.

$terraform destroy –auto-approve



Resources are Successfully destroyed



* **Proof of Concept: AWS SNS and CloudWatch Event Integration Using Terraform**

**Overview:** It demonstrates how to set up an Amazon SNS topic, apply a policy to it, create a subscription, and configure a CloudWatch Event Rule to send notifications to the SNS topic when EC2 instance state changes.

1. **Terraform and provider Configuration:** Set up the AWS provider to manage resources in the us-east-1 region.

terraform {

required\_providers {

aws = {

source = "hashicorp/aws"

version = "~> 5.0"

}

}

}

# Configure the AWS Provider

provider "aws" {

region = "us-east-1"

}

1. SNS Topic: Create an SNS topic with a delivery policy.

resource "aws\_sns\_topic" "ec2sns" {

name = "my-topic"

delivery\_policy = <<EOF

{

"http": {

"defaultHealthyRetryPolicy": {

"minDelayTarget": 20,

"maxDelayTarget": 20,

"numRetries": 3,

"numMaxDelayRetries": 0,

"numNoDelayRetries": 0,

"numMinDelayRetries": 0,

"backoffFunction": "linear"

},

"disableSubscriptionOverrides": false,

"defaultThrottlePolicy": {

"maxReceivesPerSecond": 1

}

}

}

EOF

}

1. S**NS Topic Policy:** Attach a policy to the SNS topic allowing CloudWatch Events to publish messages.

resource "aws\_sns\_topic\_policy" "default" {

arn = aws\_sns\_topic.ec2sns.arn

policy = data.aws\_iam\_policy\_document.sns\_topic\_policy.json

}

data "aws\_iam\_policy\_document" "sns\_topic\_policy" {

statement {

effect = "Allow"

actions = ["SNS:Publish"]

principals {

type = "Service"

identifiers = ["events.amazonaws.com"]

}

resources = [aws\_sns\_topic.ec2sns.arn]

}

}

1. **SNS Subscription:** Subscribe an email endpoint to the SNS topic.

resource "aws\_sns\_topic\_subscription" "targetmail" {

topic\_arn = aws\_sns\_topic.ec2sns.arn

protocol = "email"

endpoint = "yaswanth889@gmail.com"

}

1. C**loudWatch Event Rule:** Create a CloudWatch event rule to trigger on EC2 instance state changes.

resource "aws\_cloudwatch\_event\_rule" "event" {

name = "MyEC2StateChangeEvent"

description = "MyEC2StateChangeEvent"

event\_pattern = <<EOF

{

"source": [

"aws.ec2"

],

"detail-type": [

"EC2 Instance State-change Notification"

]

}

EOF

}

1. **CloudWatch Event Target:** Configure the CloudWatch event rule to send events to the SNS topic.

resource "aws\_cloudwatch\_event\_target" "sns" {

rule = aws\_cloudwatch\_event\_rule.event.name

target\_id = "SendToSNS"

arn = aws\_sns\_topic.ec2sns.arn

}

1. **Outputs:** Define outputs to easily retrieve ARNs of the SNS topic and CloudWatch Event Rule.

output "topic\_arn" {

description = "ARN of SNS tpoic"

value = aws\_sns\_topic.ec2sns.arn

}

output "event\_name" {

description = "ARN of CloudWatch Rule"

value = aws\_cloudwatch\_event\_rule.event.arn

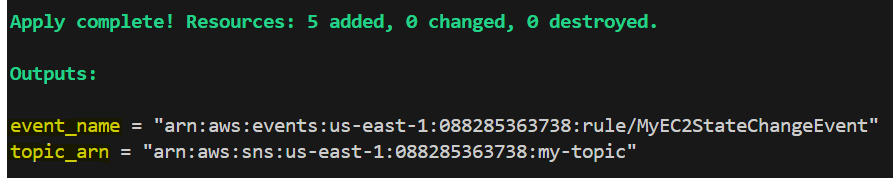
}

**Run below terraform commands to deploy resources in aws:**

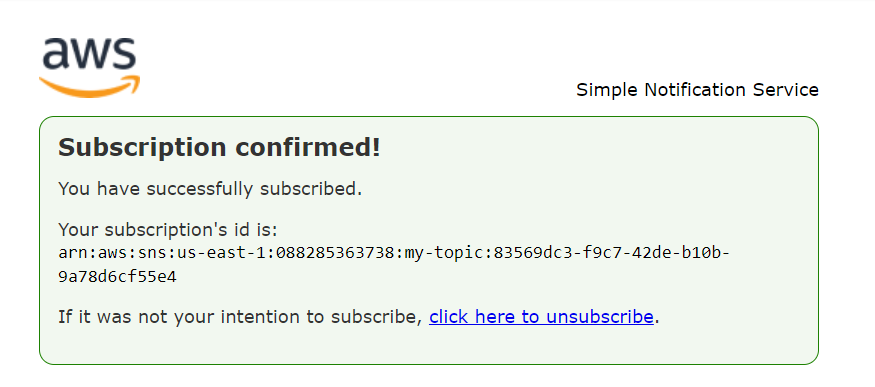
$terraform init

$terraform plan -out tfstate

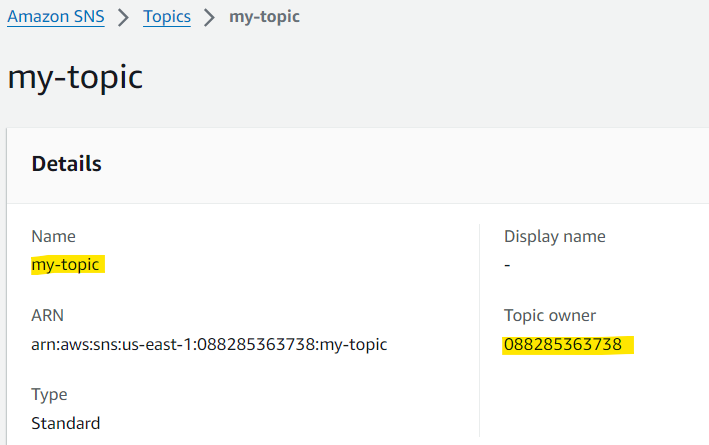
$terraform apply --auto-approve

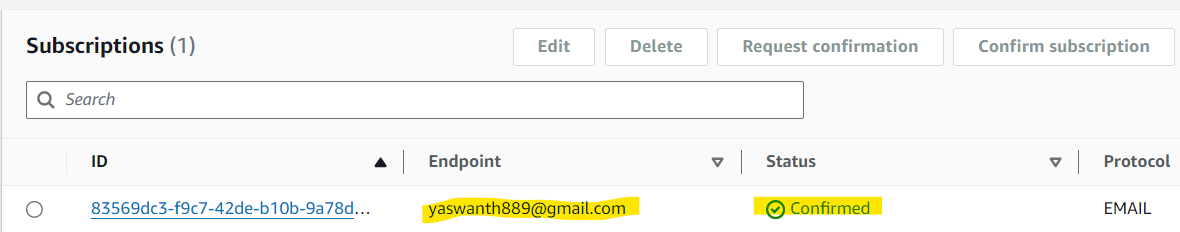


* **Confirm Subscription**: Check your email for a subscription confirmation message from SNS and confirm the subscription by clicking the link provided.

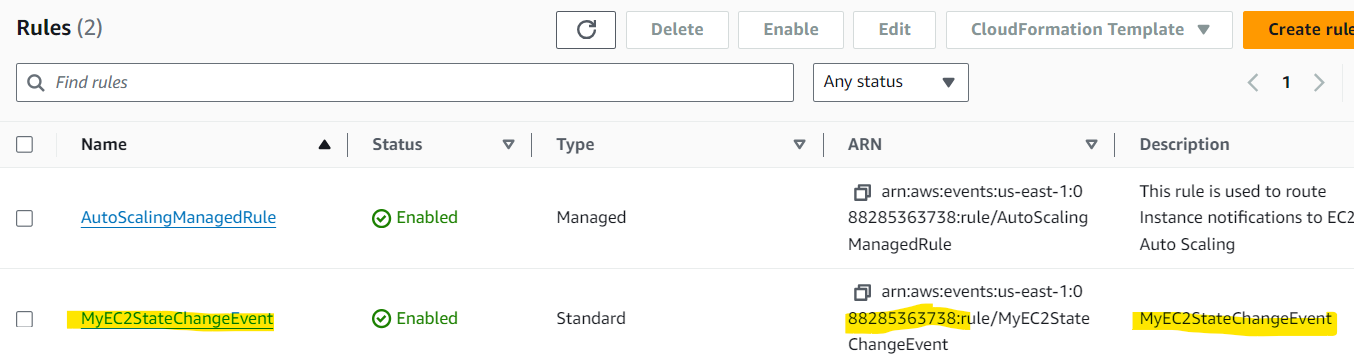


* **Verify Setup:** Verify that the notification is received in the subscribed email.
  + **SNS Topic**: Ensure the SNS topic, subscription is created and the policy is correctly applied.

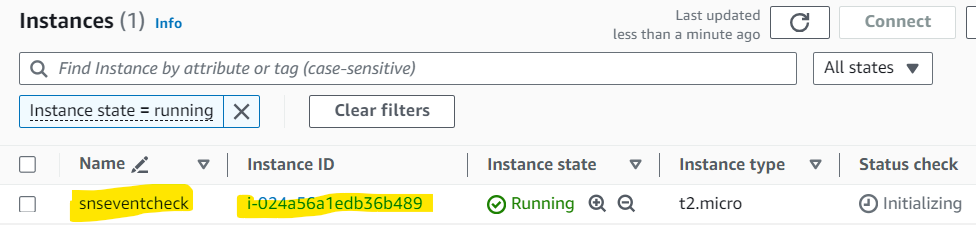




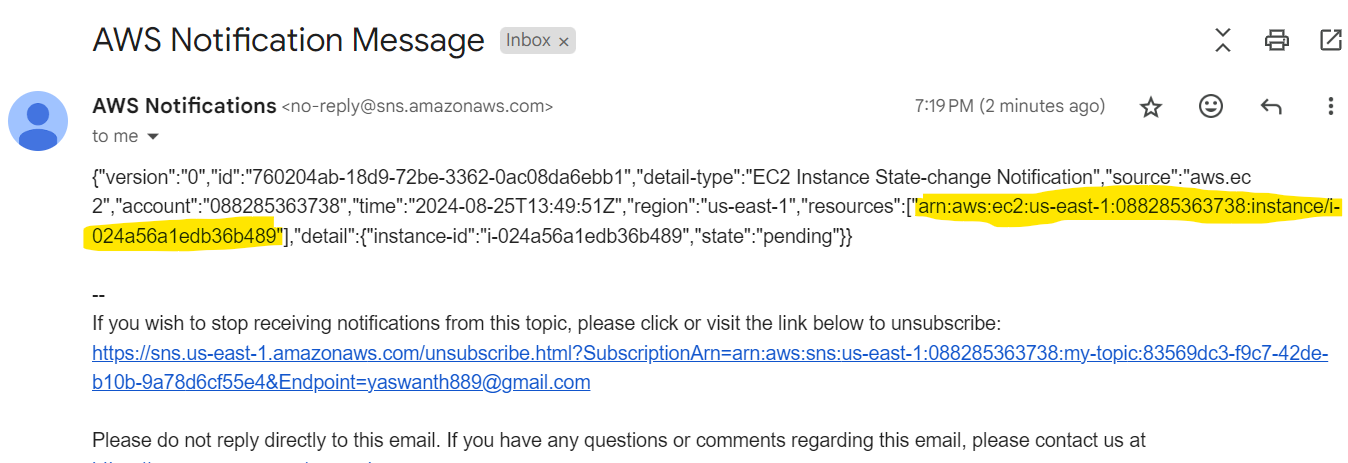
* + **CloudWatch Event**: Verify that the CloudWatch event rule is created and is properly configured to send notifications to the SNS topic.



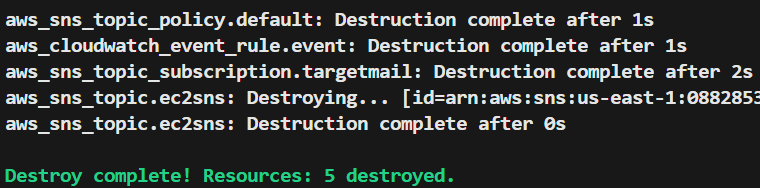
* **Trigger an EC2 Instance State Change**: Change the state of an EC2 instance (e.g., start, stop, terminate) to generate an event.



* **Check SNS Notifications**: Verify that the notification is received in the subscribed email.



After all done $terraform destroy --auto-approve -------- To destroy all resources



**AZURE CLOUD**

* **Proof of Concept: Setting Up an Azure environment with load balancer and two virtual machines running Nginx.**

**Overview:** This poc setting up a load balancer and two virtual machines running Nginx using Terraform. The setup includes configuring VN, security groups, network interface and VM.

**Prerequisites:**

* **Terraform:** Installed on your local machine.
* **Azure CLI:** Installed and configured for authentication.

1. **Bash script:** install the nginx

#!/bin/bash

# Update package list

sudo apt-get update

# Install Nginx

sudo apt-get install nginx -y

# Start Nginx

systemctl enable nginx

systemctl start nginx

# Write a custom index.html file

echo "<html><body><h1>Hello from Terraform vm1</h1></body></html>" > /var/www/html/index.html

1. Terraform and provider configuration

terraform {

required\_providers {

azapi = {

source = "azure/azapi"

version = "~>1.5"

}

azurerm = {

source = "hashicorp/azurerm"

version = "=2.84.0"

}

}

required\_version = ">= 1.1.3"

}

provider "azurerm" {

features {}

}

1. **Create resource group:** Azure Resource Group, which is a container for organizing and managing Azure resources.

**resource "azurerm\_resource\_group" "example" {**

**name = "LoadBalancerRG"**

**location = "Central India"**

**}**

1. **Configure Virtual network:** virtual private network includes subnets, security groups, network interface to deploy resources inside the AZURE VN.

resource "azurerm\_virtual\_network" "example" {

name = "example-network"

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 = "mysubnet"

resource\_group\_name = azurerm\_resource\_group.example.name

virtual\_network\_name = azurerm\_virtual\_network.example.name

address\_prefixes = ["10.0.2.0/24"]

}

# Create Network Security Group and rules

resource "azurerm\_network\_security\_group" "example" {

name = "lb-sg-SecurityGroup1"

location = azurerm\_resource\_group.example.location

resource\_group\_name = azurerm\_resource\_group.example.name

security\_rule {

name = "Allowanyhttp"

priority = 210

direction = "Inbound"

access = "Allow"

protocol = "tcp"

source\_port\_range = "\*"

destination\_port\_range = "80"

source\_address\_prefix = "\*"

destination\_address\_prefix = "\*"

}

security\_rule {

name = "allow-ssh"

priority = 110

direction = "Inbound"

access = "Allow"

protocol = "Tcp"

source\_port\_range = "\*"

destination\_port\_range = "22"

source\_address\_prefix = "\*"

destination\_address\_prefix = "\*"

}

}

resource "azurerm\_network\_interface" "nic1" {

name = "example-nic1"

location = azurerm\_resource\_group.example.location

resource\_group\_name = azurerm\_resource\_group.example.name

ip\_configuration {

name = "internal"

subnet\_id = azurerm\_subnet.example.id

private\_ip\_address\_allocation = "Dynamic"

}

}

# Create network interface

resource "azurerm\_network\_interface" "nic2" {

name = "example-nic2"

location = azurerm\_resource\_group.example.location

resource\_group\_name = azurerm\_resource\_group.example.name

ip\_configuration {

name = "internal"

subnet\_id = azurerm\_subnet.example.id

private\_ip\_address\_allocation = "Dynamic"

}

}

1. **Intergrate security group with network interface** : create resource network interface security group association

resource "azurerm\_network\_interface\_security\_group\_association" "az\_nsg\_security1" {

network\_interface\_id = azurerm\_network\_interface.nic1.id

network\_security\_group\_id = azurerm\_network\_security\_group.example.id

}

resource "azurerm\_network\_interface\_security\_group\_association" "az\_nsg\_security2" {

network\_interface\_id = azurerm\_network\_interface.nic2.id

network\_security\_group\_id = azurerm\_network\_security\_group.example.id

}

1. **create resource public ip:**

resource "azurerm\_public\_ip" "example" {

name = "Public-IP"

location = azurerm\_resource\_group.example.location

resource\_group\_name = azurerm\_resource\_group.example.name

allocation\_method = "Static"

sku = "Standard"

}

1. **loadbalancer configuration**: configuring loadbalancer resource which includes frond end ip configuration, backend address pool, probe and rules.

resource "azurerm\_lb" "example" {

name = "virtualLoadBalancer"

location = azurerm\_resource\_group.example.location

resource\_group\_name = azurerm\_resource\_group.example.name

sku = "standard"

sku\_tier = "Regional"

frontend\_ip\_configuration {

name = "PublicIPAddress"

public\_ip\_address\_id = azurerm\_public\_ip.example.id

}

}

resource "azurerm\_lb\_backend\_address\_pool" "example" {

loadbalancer\_id = azurerm\_lb.example.id

name = "BackEndAddressPool"

}

resource "azurerm\_network\_interface\_backend\_address\_pool\_association" "az\_pool\_association\_nic1" {

backend\_address\_pool\_id = azurerm\_lb\_backend\_address\_pool.example.id

network\_interface\_id = azurerm\_network\_interface.nic1.id

ip\_configuration\_name = "internal"

}

resource "azurerm\_network\_interface\_backend\_address\_pool\_association" "az\_pool\_association\_nic2" {

backend\_address\_pool\_id = azurerm\_lb\_backend\_address\_pool.example.id

network\_interface\_id = azurerm\_network\_interface.nic2.id

ip\_configuration\_name = "internal"

}

resource "azurerm\_lb\_probe" "example" {

resource\_group\_name = azurerm\_resource\_group.example.name

loadbalancer\_id = azurerm\_lb.example.id

name = "http-running-probe"

protocol = "http"

port = 80

request\_path = "/"

interval\_in\_seconds = 15

number\_of\_probes = 2

}

resource "azurerm\_lb\_rule" "example" {

resource\_group\_name = azurerm\_resource\_group.example.name

name = "example-lb-rule"

loadbalancer\_id = azurerm\_lb.example.id

protocol = "Tcp"

frontend\_port = 80

backend\_port = 80

frontend\_ip\_configuration\_name = azurerm\_lb.example.frontend\_ip\_configuration[0].name

backend\_address\_pool\_ids = [azurerm\_lb\_backend\_address\_pool.example.id]

probe\_id = azurerm\_lb\_probe.example.id

}

1. configuring azure virtual machine:

resource "tls\_private\_key" "secureadmin\_ssh" {

algorithm = "RSA"

rsa\_bits = 4096

}

resource "azurerm\_linux\_virtual\_machine" "server1" {

name = "vm-machine1"

resource\_group\_name = azurerm\_resource\_group.example.name

location = azurerm\_resource\_group.example.location

size = "Standard\_F2"

admin\_username = "adminuser"

network\_interface\_ids = [azurerm\_network\_interface.nic1.id]

admin\_ssh\_key {

username = "adminuser"

public\_key = tls\_private\_key.secureadmin\_ssh.public\_key\_openssh

}

custom\_data = filebase64("userdata.sh") # Base64 encoded user data

os\_disk {

caching = "ReadWrite"

storage\_account\_type = "Standard\_LRS"

}

source\_image\_reference {

publisher = "Canonical"

offer = "0001-com-ubuntu-server-jammy"

sku = "22\_04-lts"

version = "latest"

}

}

resource "azurerm\_linux\_virtual\_machine" "server2" {

name = "vm-machine2"

resource\_group\_name = azurerm\_resource\_group.example.name

location = azurerm\_resource\_group.example.location

size = "Standard\_F2"

admin\_username = "adminuser"

network\_interface\_ids = [azurerm\_network\_interface.nic2.id]

admin\_ssh\_key {

username = "adminuser"

public\_key = tls\_private\_key.secureadmin\_ssh.public\_key\_openssh

}

custom\_data = filebase64("userdata2.sh") # Base64 encoded user data

os\_disk {

caching = "ReadWrite"

storage\_account\_type = "Standard\_LRS"

}

source\_image\_reference {

publisher = "Canonical"

offer = "0001-com-ubuntu-server-jammy"

sku = "22\_04-lts"

version = "latest"

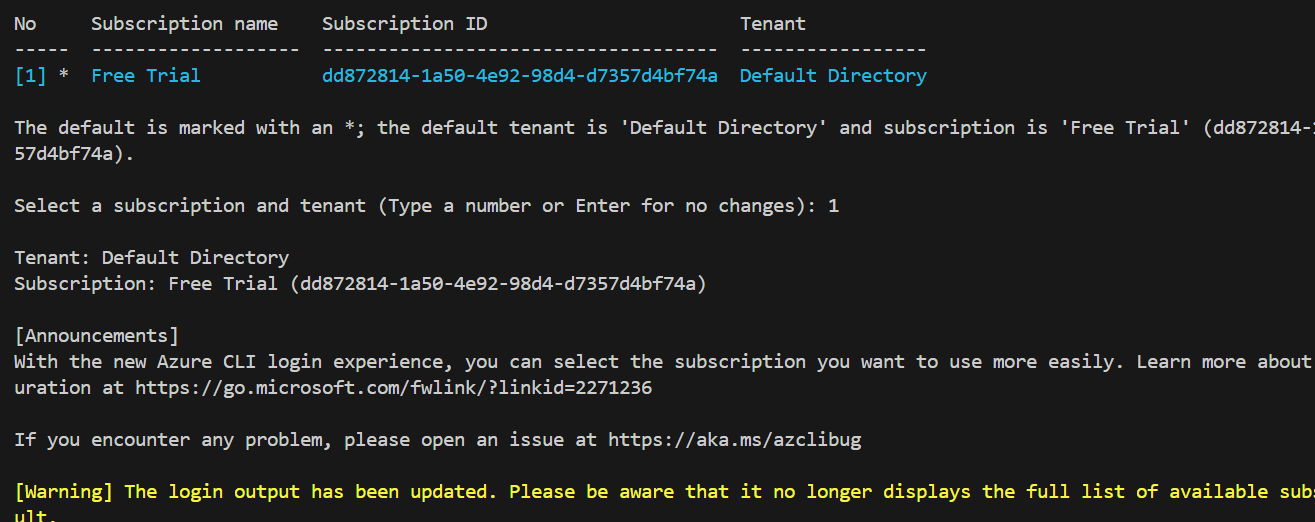
}

}

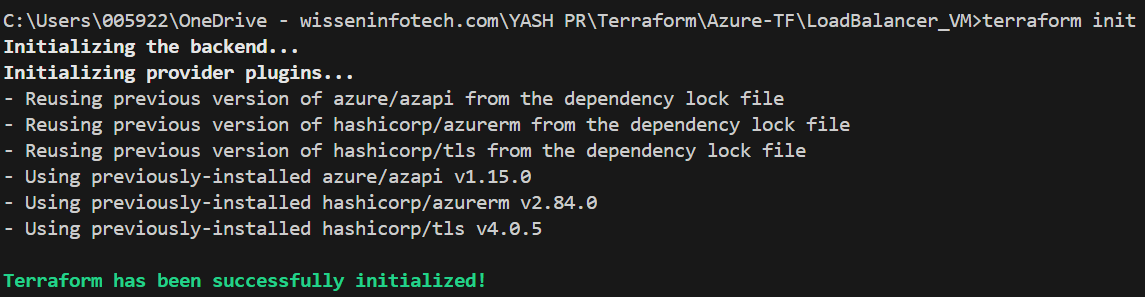
**Steps to Deploy:**

1. **configure azure cli with required permissions to create resource in azure portal**

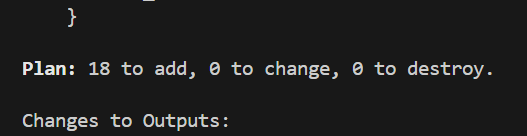
**$az login**

****

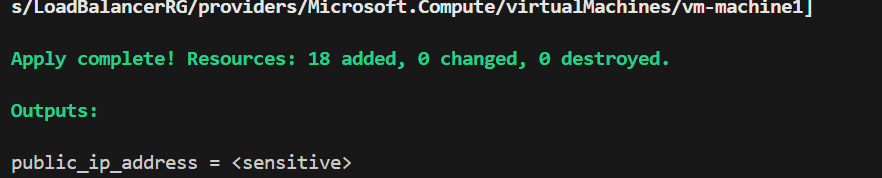
1. **$terraform init**

****

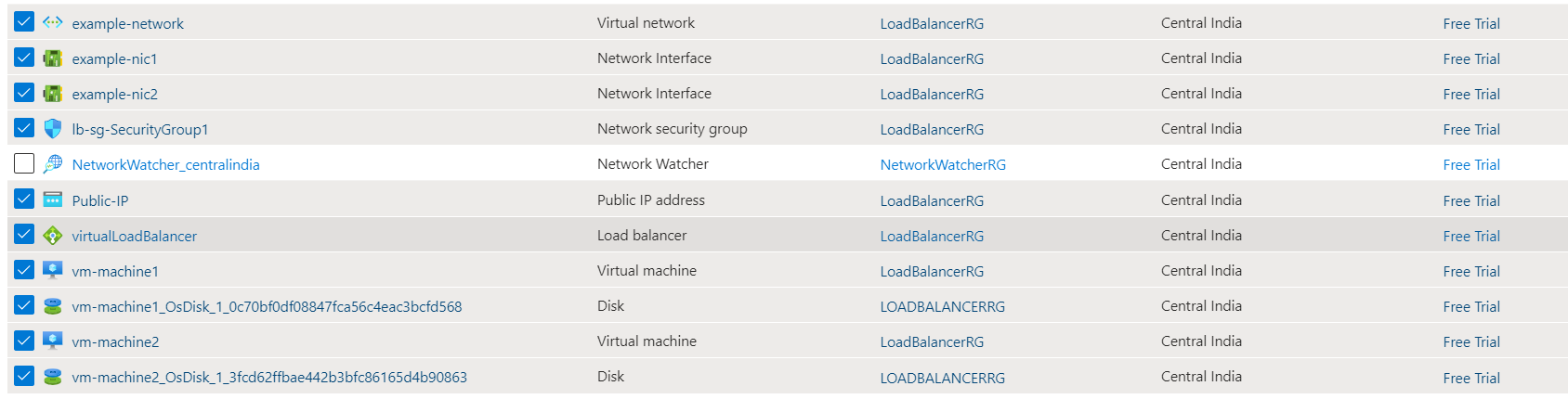
1. **$terraform plan –out tfstate**

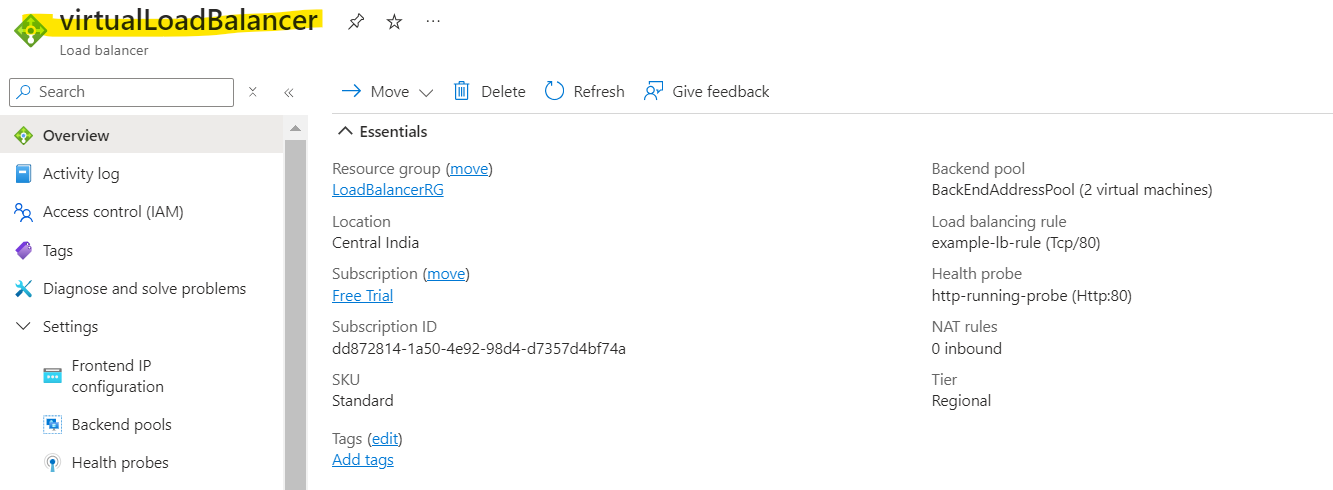
****

1. **$terraform apply –auto-approve**

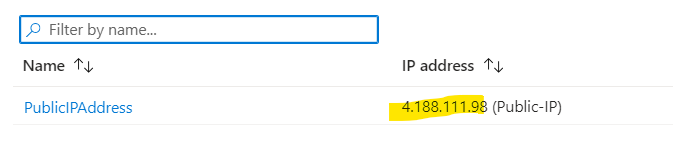
****

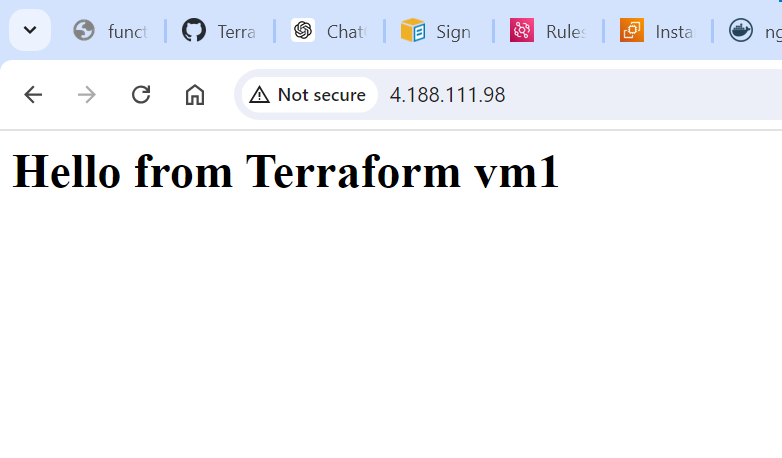
**Connect to azure portal and verify resources are created or not**

****

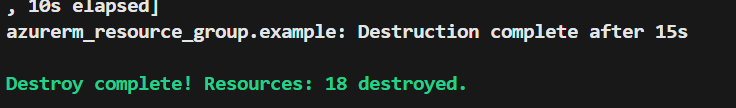
****

**Open load balancer and copy public ip which is assigned to azure LB and past it in web browser**

****

****

1. **$terraform destroy –auto-approve**

****