\*\* Define the "AWS\_ACCESS\_KEY" and "AWS\_SECRET\_KEY" variables with default values:

# vars.tf

# Input Variables https://www.terraform.io/language/values/variables

variable "aws\_access\_key" {

}

variable "aws\_secret\_key" {

}

variable "apac\_region" {

default = "ap-south-1"

}

variable "cidr\_blocks" {

default = "0.0.0.0/0"

}

#Network Mask - 255.255.255.0 Addresses Available - 256

variable "vpc\_cidr" {

default = "10.0.1.0/24"

}

variable "public\_cidr" {

default = "10.0.1.0/28"

}

variable "private\_cidr" {

default = "10.0.1.16/28"

}

variable "instance\_type" {

default = "t2.micro"

}

#Providers are a logical abstraction of an upstream API. They help to understand API interactions and exposing provider resources such AWS, Google, Azure - https://registry.terraform.io/browse/providers.

provider "aws" {

region = var.apac\_region

}

\*\* devVPC.tf

# Providers are a logical abstraction of an upstream API. They help to understand API interactions and exposing provider resources such AWS, Google, Azure

provider "aws" {

region = var.apac\_region

}

# Query all available Availability Zone; we will use specific availability zone using index - The Availability Zones data source provides access to the list of AWS Availability Zones which can be accessed by an AWS account specific to region configured in the provider.

data "aws\_availability\_zones" "available" {}

# Provides a VPC resource

resource "aws\_vpc" "devVPC" {

cidr\_block = var.vpc\_cidr

enable\_dns\_hostnames = true

enable\_dns\_support = true

tags = {

Name = "dev\_terraform\_vpc"

}

}

# Public Subnet - Provides an VPC subnet resource

resource "aws\_subnet" "public\_subnet" {

cidr\_block = var.public\_cidr

vpc\_id = aws\_vpc.devVPC.id

map\_public\_ip\_on\_launch = true

availability\_zone = data.aws\_availability\_zones.available.names[1]

tags = {

Name = "dev\_terraform\_vpc\_public\_subnet"

}

}

# Private Subnet - Provides an VPC subnet resource

resource "aws\_subnet" "private\_subnet" {

cidr\_block = var.private\_cidr

vpc\_id = aws\_vpc.devVPC.id

map\_public\_ip\_on\_launch = false

availability\_zone = data.aws\_availability\_zones.available.names[1]

tags = {

Name = "dev\_terraform\_vpc\_private\_subnet"

}

}

#To access EC2 instance inside a Virtual Private Cloud (VPC) we need an Internet Gateway and a routing table connecting the subnet to the Internet Gateway

# Creating Internet Gateway

# Provides a resource to create a VPC Internet Gateway

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

vpc\_id = aws\_vpc.devVPC.id

tags = {

Name = "dev\_terraform\_vpc\_igw"

}

}

# Provides a resource to create a VPC routing table

resource "aws\_route\_table" "public\_route" {

vpc\_id = aws\_vpc.devVPC.id

route {

cidr\_block = var.cidr\_blocks

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

}

tags = {

Name = "dev\_terraform\_vpc\_public\_route"

}

}

# Provides a resource to create an association between a Public Route Table and a Public Subnet

resource "aws\_route\_table\_association" "public\_subnet\_association" {

route\_table\_id = aws\_route\_table.public\_route.id

subnet\_id = aws\_subnet.public\_subnet.id

depends\_on = [aws\_route\_table.public\_route, aws\_subnet.public\_subnet]

}

# Provides a security group resource - https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/security\_group

resource "aws\_security\_group" "sg\_allow\_ssh\_http" {

vpc\_id = aws\_vpc.devVPC.id

name = "dev\_terraform\_vpc\_allow\_ssh\_http"

tags = {

Name = "dev\_terraform\_sg\_allow\_ssh\_http"

}

}

# Ingress Security Port 22 (Inbound) - Provides a security group rule resource (https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/security\_group\_rule)

resource "aws\_security\_group\_rule" "ssh\_ingress\_access" {

from\_port = 22

protocol = "tcp"

security\_group\_id = aws\_security\_group.sg\_allow\_ssh\_http.id

to\_port = 22

type = "ingress"

cidr\_blocks = [var.cidr\_blocks]

}

# Ingress Security Port 80 (Inbound)

resource "aws\_security\_group\_rule" "http\_ingress\_access" {

from\_port = 80

protocol = "tcp"

security\_group\_id = aws\_security\_group.sg\_allow\_ssh\_http.id

to\_port = 80

type = "ingress"

cidr\_blocks = [var.cidr\_blocks]

}

# Ingress Security Port 8080 (Inbound)

resource "aws\_security\_group\_rule" "http8080\_ingress\_access" {

from\_port = 8080

protocol = "tcp"

security\_group\_id = aws\_security\_group.sg\_allow\_ssh\_http.id

to\_port = 8080

type = "ingress"

cidr\_blocks = [var.cidr\_blocks]

}

# Egress Security (Outbound)

resource "aws\_security\_group\_rule" "egress\_access" {

from\_port = 0

protocol = "-1"

security\_group\_id = aws\_security\_group.sg\_allow\_ssh\_http.id

to\_port = 0

type = "egress"

cidr\_blocks = [var.cidr\_blocks]

}

\*\* The following is a script to create an AMI in Amazon EC2 using Packer. AMI will have Java and Jenkins installed in it. Create the jenkinsami-packer.json file:

{

"variables": {

"aws\_access\_key": "",

"aws\_secret\_key": ""

},

"builders": [

{

"type": "amazon-ebs",

"access\_key": "{{user `aws\_access\_key`}}",

"secret\_key": "{{user `aws\_secret\_key`}}",

"region": "ap-south-1",

"source\_ami": "ami-0c6615d1e95c98aca",

"instance\_type": "t2.micro",

"ssh\_username": "ec2-user",

"ami\_name": "packer-jenkins-ami-{{timestamp}}"

}

],

"provisioners": [

{

"type": "shell",

"inline": [

"sudo yum update –y",

"sudo yum remove java -y",

"sudo yum install java-1.8.0-openjdk-devel -y",

"sudo wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat/jenkins.repo",

"sudo rpm --import https://pkg.jenkins.io/redhat/jenkins.io.key",

"sudo yum install jenkins -y"

]

}

]

}

Execute packer build -var "aws\_access\_key=XXXXXXXXXXXXXXXXXXXX" -var "aws\_secret\_key=XXXXXXXXXXXXXXXXXXXX" jenkinsami-packer.json.

#Get latest AMI ID based on Filter - Here AMI created using Packer

data "aws\_ami" "packeramis" {

owners = ["10xxxxxxxxxx"] #change the owner ID as per your account

most\_recent = true

filter {

name = "name"

values = ["packer-jenkins\*"]

}

}

#Create an Instance using latest Packer AMI

resource "aws\_instance" "jenkins-instance" {

ami = data.aws\_ami.packeramis.id

instance\_type = var.instance\_type

key\_name = "terraform"

vpc\_security\_group\_ids = [aws\_security\_group.sg\_allow\_ssh\_http.id]

subnet\_id = aws\_subnet.public\_subnet.id

tags = {

Name = "dev\_terraform\_jenkins\_instance"

}

}

The following is the Terraform script that creates security group for EFS. Note the name of security group that we will refer to in the upcoming script:

# Ingress Security Port 2049 (Inbound)

resource "aws\_security\_group" "sg\_jenkins\_efs" {

name\_prefix = "sg\_jenkins\_efs"

vpc\_id = aws\_vpc.devVPC.id

ingress {

from\_port = 2049

to\_port = 2049

protocol = "tcp"

cidr\_blocks = [var.cidr\_blocks]

}

}

Create EC2 instance using Terraform and once instance is available, execute the following commands in EC2 instance:

#!/bin/bash

sudo yum update -y

sudo yum install nfs-utils

#Mount EFS Mount Access point

sudo mkdir /root/.jenkins

sudo mount -t nfs4 -o nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2,noresvport fs-0d6103dd6806be283.efs.ap-south-1.amazonaws.com:/ /root/.jenkins

The following code will create EFS, access point, and mount target:

# Provides an Elastic File System (EFS) File System resource to store JENKINS\_HOME

resource "aws\_efs\_file\_system" "jenkins\_home\_efs" {

creation\_token = "jenkins\_home\_efs"

tags = {

Name = "dev\_terraform\_jenkins\_home"

}

}

# Provides an Elastic File System (EFS) mount target

resource "aws\_efs\_mount\_target" "jenkins\_mount\_target" {

file\_system\_id = aws\_efs\_file\_system.jenkins\_home\_efs.id

subnet\_id = aws\_subnet.public\_subnet.id

security\_groups = [aws\_security\_group.sg\_jenkins\_efs.id]

}

# Provides an Elastic File System (EFS) access point

resource "aws\_efs\_access\_point" "jenkins\_access\_point" {

file\_system\_id = aws\_efs\_file\_system.jenkins\_home\_efs.id

root\_directory {

path = "/"

}

}

Create a userdata.tpl file with the following content and change the file system ID in the userdata.tpl file to the new file system created by terraform script above:

#!/bin/bash

sudo yum update -y

sudo yum install nfs-utils

#Mount EFS Mount Access point

sudo mkdir /root/.jenkins

sudo mount -t nfs4 -o nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2,noresvport fs-0d6103dd6806be283.efs.ap-south-1.amazonaws.com:/ /root/.jenkins

Add the template\_file block in the main terraform file and provide its reference in instance configuration, as follows:

#The template\_file data source usually loaded from an external file.

data "template\_file" "init" {

template = file("${path.module}/userdata.tpl")

}

#Create an Instance using latest Packer AMI and apply User Data - This allows instances to be created, updated, and deleted - https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance

resource "aws\_instance" "jenkins-instance" {

ami = data.aws\_ami.packeramis.id

instance\_type = var.instance\_type

key\_name = "terraform"

vpc\_security\_group\_ids = [aws\_security\_group.sg\_allow\_ssh\_http.id]

subnet\_id = aws\_subnet.public\_subnet.id

user\_data = data.template\_file.init.rendered

tags = {

Name = "dev\_terraform\_jenkins\_instance"

}

}

Let’s create launch configuration using the aws\_launch\_configuration resource and autoscaling group using aws\_autoscaling\_group.

We will also define policies to scale resources up and down. Create autoscaling.tf and copy the following script into it:

resource "aws\_launch\_configuration" "nginx\_launch\_config" {

image\_id = data.aws\_ami.packeramis.id

instance\_type = var.instance\_type

security\_groups = [aws\_security\_group.sg\_allow\_ssh\_http.id]

user\_data = data.template\_file.init.rendered

lifecycle {

create\_before\_destroy = true

}

}

resource "aws\_autoscaling\_group" "nginx\_autoscaling\_group" {

launch\_configuration = aws\_launch\_configuration.nginx\_launch\_config.id

vpc\_zone\_identifier = [aws\_subnet.public\_subnet.id]

health\_check\_type = "ELB"

min\_size = 2

max\_size = 5

load\_balancers = [aws\_elb.nginx-elb.id]

tag {

key = "Name"

value = "dev\_terraform\_nginx\_instance\_asg"

propagate\_at\_launch = true

}

}

resource "aws\_autoscaling\_policy" "nginx\_cpu\_policy\_scaleup" {

name = "nginx\_cpu\_policy\_scaleup"

autoscaling\_group\_name = aws\_autoscaling\_group.nginx\_autoscaling\_group.name

adjustment\_type = "ChangeInCapacity"

scaling\_adjustment = 1

cooldown = "120"

}

resource "aws\_autoscaling\_policy" "nginx\_cpu\_policy\_scaledown" {

name = "nginx\_cpu\_policy\_scaledown"

autoscaling\_group\_name = aws\_autoscaling\_group.nginx\_autoscaling\_group.name

adjustment\_type = "ChangeInCapacity"

scaling\_adjustment = -1

cooldown = "120"

}

Let’s create Elastic load balancer and configure listener as well as health check details. Create a file named elb.tf:

# Elastic Load Balancer resource, also known as a Classic Load Balancer - https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/elb

resource "aws\_elb" "nginx-elb" {

name = "nginx-elb"

subnets = [aws\_subnet.public\_subnet.id]

security\_groups = [aws\_security\_group.sg\_allow\_ssh\_http.id]

listener {

instance\_port = 80

instance\_protocol = "http"

lb\_port = 80

lb\_protocol = "http"

}

health\_check {

healthy\_threshold = 2

unhealthy\_threshold = 2

timeout = 2

target = "HTTP:80/"

interval = 30

}

tags = {

Name = "nginx\_elb"

}

}

Add load\_balancers = [aws\_elb.nginx-elb.id] in the auto scaling group that we created earlier to link ELB and Auto scaling group:

resource "aws\_autoscaling\_group" "nginx\_autoscaling\_group" {

launch\_configuration = aws\_launch\_configuration.nginx\_launch\_config.id

vpc\_zone\_identifier = [aws\_subnet.public\_subnet.id]

health\_check\_type = "ELB"

min\_size = 2

max\_size = 5

load\_balancers = [aws\_elb.nginx-elb.id]

tag {

key = "Name"

value = "dev\_terraform\_nginx\_instance\_asg"

propagate\_at\_launch = true

}

}