**JAVA HOME CLOUD Terraform Project**

Terraform :- Terraform is a tool for automating Infrastructure provisioning or a tool for implementing infrastructure as code .

Using Terraform we can automatically setup infrastructure in the cloud

Infrastructure as code (IAC): is practise of provisioning the resources in the cloud using configuration files.

Advantages of IAC:

1. It Is automated , if you want to create similar environments like dev,test,prod kind of environments we can reuse them .
2. If there are any issues with the release you made recently for your infrastructure , we can revert back it to previous state,
3. Iif you caught with any bugs troubleshooting will becomes easy using source code management tools like git ,we can go and see what are the recent changes made to the infrastructure.

Terraform tool is from Hashicorp and this tool is written in Go Lang

.

Configure AWS IAM access KEY ID and secret Access key

In aws acoount go to IAM ->USERS->add user-> un:Terraform–ACESS KEY ID , SECRET ACCESS KEY 🡪access type->Programatic access ->Set permissions->attach existing policies directly->and choose administrative acces🡪create user.

Inorder to configure access keys and secret keys we need amazon cli in laptop

Run following commands in terminal 🡪 aws comfigure

Copy and paste access key id of Terraform user(–ACESS KEY ID)

Copy and paste the secret key of Terraform user(SECRET ACCESS KEY )

Region to procision the resources we neeed is Mumbai ap-south-1

The configurations are by default used by Terraform to create the resources in the aws account

Install Atom Editor

Create a Workspace : mkdir Terraform

cd Terraform

atom .

go to search bar and install packages 🡪terraform🡪 language Terraform (install)-🡪terraform-fmt(install)🡪go to settings (format on save ) is checked

Terraform supports multiple cloud providers,but we are using aws

Providers.tf

provider "aws" {

region = "ap-south-1"

}

Terraform knows that it has to create the resources in aws by above file

Once we write the Terraform file we need to perform Terraform init

Terraform init command will download the provider plugins in our case aws .

Creating resources using Terraform in aws

TO CREATE VPC

Terraform apply command picks all the Terraform files in the current folder and executes them

**Terraform State File:** All The resources created using Terraform scripts is maintained inside a file which is in JSON format default filename is Terraform ,default extension is . tfstate and the default location is within project workspace

Any change made to script and we apply terraform apply then terraform compares the terraform script against the tfstate file and performs the actions.

If we lose the tfstate file terraform loses the control over the created resources , and we perform terraform apply it will create one more vpc by forgetting what it has prepared before . we should never update or edit the tfstate file .

**Terraform Remote state using S3 Backend:**

Terraform by default maintains tfstate file in local machine , this is fine only one developer is working on the project , its going to be a problem when multiple developers wants to share same state file and same code base , in-order to solve this type of problems we want to use S3 as Remote state file location .

provider  "aws" {

  region = "ap-south-1" -🡪providers block

}

terraform {

  backend "s3" {

    bucket = "febterrafromproject" --🡪(s3 bucket name you have to create it in awsconsole)

    key    = "terraform.tfstate"

    region = "ap-south-1"

  }

  }

resource "aws\_vpc" "my\_vpc" {

  cidr\_block       = "10.0.0.0/16" 🡪resource block

  instance\_tenancy = "default"

  tags = {

    Name = "javahomevpc"

    Environment = "Dev" --🡪 resource tag block

  }

}

output "vpc\_cidr" {

    value = "${aws\_vpc.my\_vpc.cidr\_block}" ---> interpolation ${} getting the value of the resource using output argument

}

Why cant we create this bucket using terraform and refer that here?

That will not work here because ,before any resource created in terraform , it has to go and check tfstate file ,to do that this bucket must exists .

\*\*\*In reatime scenarios we need to create all this buckets outside terraform ,probably we can have a shell/pythin script .

\*\*\* first macke sure that the bucket is present and next run terraform apply

\*\*\* after adding remote backend details , we need to do again terraform init (----this is the second scenario where we use terraform init----it is initializing the backend , it says it found existing state file and asks is it fine to copy it to remote ---yes—and it succesfuuuly initialized the remote state file) o/p: go and check in aws console s3 bucket section in s3 bucket the state file will be present.

(it removes all the data in local tfstate file and places it in s3 bucket---now we can delete those tfstate file, tfstate backend file from local ---)

\*\*\*\*Because of s3 backend module configuration under providers, terraform will go to s3 bucket and checks the tfstate there not in local . it is highly recommended to enable s3 bucket versioning every time we change the state file a new version is created in the s3,so that we can revert back to the previous version ,we can pull that file and utilize it, its also good practise to enable encryption (for example if we are provisioning RDS its going to store RDS password , a file in a s3 bucket contains sensitive information should be eeencrypted \*\*\*

And now perform terraform apply

**Terraform Locking Remote state File:**

We successfully configured s3 backed for storing state file, Next problem would be locking it ( remote state file) , If we won’t lock this remote state file multiple developers applying concurrently it can create inconsistent state for us ,\*\*\*\*\*\* it is always important to lock the state file when a developer is currently performing operations on that and block the remaining users until the current operation completes.

**How to enable Locking when state files are stored in s3 :**

* **DynamoDB State Locking:** [dynamodb\_table](https://www.terraform.io/docs/language/settings/backends/s3.html#dynamodb_table) - (Optional) Name of DynamoDB Table to use for state locking and consistency. The table must have a primary key named LockID with type of string. If not configured, state locking will be disabled.

For this example there should be a table in dynamo db in aws for that go to aws console🡪search for dynamo db service and create table with name javahome-tf and and primary key as LockID and create a table.

Now we successfully created remote state file and created lock on it , to visualize the locking we need to do some changes to the terraform script , why because when its get locked is only it requires to update the resources

terraform init – because we have updated terraform locking details in script so we need to perform terraform init

terraform apply and now check in aws dynamo db items section there will be an entry and it will have the information who is currently performing this operation , here one entry is made but there is no lock , if any developer applies the code it will create one more entry in dynamo db and it will have the details about the user who got the lock on s3 remote state file

**Terraform Variables and tfvars**

In any programming language variables gives better maintained ability and code reusability.

wFor instance We have VPC and we want to use this vpc in dev, stage, prod environment and probably arguments to this vpc is different

provider  "aws" {

  region = "ap-south-1" ---🡪 Provisioner Block

}

terraform {

  backend "s3" {

    bucket = "febterrafromproject" -🡪 bucket name

    key    = "terraform.tfstate"

    region = "ap-south-1"

    dynamodb\_table = "javahome-tf" -🡪 dynamodb table name for remote state file locking to avoid conflicts

   }

  }

resource "aws\_vpc" "my\_vpc" {

  cidr\_block       = "10.0.0.0/16" --🡪 vpc Resource block

  instance\_tenancy = "default"

  tags = {

    Name = "javahomevpc" |------🡪 arguments for vpc resourcce block

    Environment = "Dev" |

    Location = "India"  |

  }

}

output "vpc\_cidr" {

    value = "${aws\_vpc.my\_vpc.cidr\_block}"   ---🡪 interpolation to get the vpc cidr value after creation of the resource

}

If we hardcode the values like this (**cidr\_block       = "10.0.0.0/16")**  it is very hard, if you point this in production you need to change all the attributes of the vpc resource , so we can use variables and we can refer those variables here , we have the option where we can change the values dynamically and passing them at command line .

For variables we have to create a separate file called variables.tf

variable "vpc\_cidr" {

    description ="choose cidr for vpc" |

    type = "string" (supported types for variables are string,map,list,boolean |

    default = "10.20.0.0/16" |---🡪 (these 3 properties are optional for variables, we can mention variables without using all these 3 properties also)

}

if we didn’t mention any type by default , it will take it as string, now we should refer this variable inside vpc resource section, in order to access the value from the variable we should use interpolations, var. is the prefix to access the variables followed by its name “${var.vpc\_cidr}” ---vpc\_cidr is the name of the variable in variable.tf

provider  "aws" {

  region = "ap-south-1"

}

terraform {

  backend "s3" {

    bucket = "febterrafromproject"

    key    = "terraform.tfstate"

    region = "ap-south-1"

    dynamodb\_table = "javahome-tf"

  }

  }

resource "aws\_vpc" "my\_vpc" {

  cidr\_block       = "${var.vpc\_cidr}"---🡪 reffering the variable using interpolation

  instance\_tenancy = "default"

  tags = {

    Name = "javahomevpc"

    Environment = "Dev"

    Location = "India"

  }

}

output "vpc\_cidr" {

    value = "${aws\_vpc.my\_vpc.cidr\_block}"

}

when we run the above script the value of the cidr is picked from variables.tf ie. 10.20.0.0/16.

suppose you want to change the value of cidr bock during run time we can do that by using it in command

terraform apply -var “variable name=its value”

eg: terraform apply -var “vpc\_cidr=10.30.0.0/16” –auto-approve

now terraform will consider this value 10.30.0.0/16 instead of this value 10.20.0.0/16 mentioned in the variables.tf

we can pass multiple variables ad fallows

terraform apply -var “variable name=its value” -var “variable name=its value”

eg: terraform apply -var “vpc\_cidr=10.30.0.0/16” -var “vpc\_cidr=10.40.0.0/16”

we can avoid the yes/no prompt by using –auto-approve

terraform apply -var “vpc\_cidr=10.30.0.0/16” –auto-approve

you have scenario to pass 20 to 30 variables in the command section, it will not be easy to pass them at command line ,so for this type we can declare our inputs in a separate file ,lets say we have 10 or 20 inputs and we want to change them during runtime ,keep them inside a file and pass a reference to that file and get things done .

name of the file can be anything but the extension should be .tfvars, I want to pass this **-var “vpc\_cidr=10.30.0.0/16”**  in a file rather than in a command

eg: dev.tfvars

vpc\_cidr = “10.30.0.0/16”

now in command use

eg: terraform apply -var-file=dev.tfvars –auto-approve

**Terraform Workspaces**

Maintaining multiple environments is a typical scenario, for terraform also we want to have dev, stage, prod environments which will have two different state files, but I want to have same set of configurations which is going to manage dev and production.(the config files will be same but I want to maintain different environments using same configuration file that’s where we use workspaces)

Terraform by default will maintain one default workspace, it will be created while initializing our terraform project

Cmd: terraform workspace list

To create workspaces

terraform workspace new dev

terraform workspace new prod

terraform maintains separate state file for dev and separate file for prod

to provision the resources in dev environment first move into that workspace

terraform workspace select dev

terraform apply –auto-approve

resources created in dev are isolated from the resources created in prod…we can use those details at the time of terraform building the scripts ..

for example we have tag environment hardcoded, so I want this value to be dynamic ,in dev I want it dev and in prod I want it to be prod , for that we can reference terraform workspace dynamically using “${terraform.workspace}”

resource "aws\_vpc" "my\_vpc" {

  cidr\_block       = "${var.vpc\_cidr}"---🡪 reffering the variable using interpolation

  instance\_tenancy = "default"

  tags = {

    Name = "javahomevpc"

    Environment = "Dev"

    Location = "India"

resource "aws\_vpc" "my\_vpc" {

  resource "aws\_vpc" "my\_vpc" {

cidr\_block       = "${var.vpc\_cidr}"---🡪 reffering the variable using interpolation

  instance\_tenancy = "default"

  tags = {

    Name = "javahomevpc"

    Environment = “${terraform.workspace}”------------🡪automatically gets the t.workspace

    Location = "India"

if your workspace is dev then environment value will be dev, if workspace is prod then the envi. Value will be prod .

**Terraform Using Loops**

Terraform supports loops, for example you want to create 10 ec2 instances , it can be created using loops , we use **count** attribute , here count attributes acts as loop.

**Terraform Conditions**

In terraform we can create certain resources conditionally, for example we have a specific resource that want to be created only in dev environment not in prod, we can do that using conditions.

provider  "aws" {

  region = "ap-south-1"

}

terraform {

  backend "s3" {

    bucket = "febterrafromproject"

    key    = "terraform.tfstate"

    region = "ap-south-1"

    dynamodb\_table = "javahome-tf"

  }

  }

resource "aws\_vpc" "my\_vpc" {

  count            = "${terraform.workspace == "dev"  ? 0 : 1}" --🡪 count acts as conditon

  cidr\_block       = "${var.vpc\_cidr}"

  instance\_tenancy = "default"

  tags = {

    Name = "javahomevpc"

    Environment = "${terraform.workspace}"

    Location = "India"

  }

}

# output "vpc\_cidr" {

#   value = "${aws\_vpc.my\_vpc.cidr\_block}"

# }

**Terraform Locals**

if we have an expression which is repeatedly used in the code we can declare them as part of local variables and refer them in the code such that in future, if you want to change the expression you don’t have to change expression at several places , you change at local variable level it reflects in other places

locals.tf

locals {

    vpc\_name = "${terraform.workspace == "dev" ? "javahome-dev" : "javahome-prod"}"

}

resource "aws\_vpc" "my\_vpc" {

  count            = "${terraform.workspace == "dev"  ? 0 : 1}"

  cidr\_block       = "${var.vpc\_cidr}"

  instance\_tenancy = "default"

  tags = {

    Name = "${local.vpc\_name}"

    Environment = "${terraform.workspace}"

    Location = "India"

  }

}

If the workspace is dev then I have to name my vpc as javahome-dev ,if workspace is other than dev then I want it to be javahome-prod.

 vpc\_name = "${terraform.workspace == "dev" ? "javahome-dev" : "javahome-prod"}"

This is my expression I can use it directly inside my resource section, however if I am repeating same expression several times, it is good practise to declare them as a local variable.

**Terraform Create Vpc**

In this project we are going to deploy a application in aws infrastructure , for that first we are building the network section

Providers.tf

provider  "aws" {

  region = "ap-south-1"

}

terraform {

  backend "s3" {

    bucket = "febterrafromproject"

    key    = "terraform.tfstate"

    region = "ap-south-1"

    dynamodb\_table = "javahome-tf"

  }

  }

Variables.tf

variable "vpc\_cidr" {

    description = "choose cidr for vpc"

    type        = "string"

    default     = "10.20.0.0/16"

}

variable "region" {

    description = "choose region for your stack"

    type        = "string"

    default     = "us-east-1" 🡪 using this because some aws services are not available in Mumbai region

}

Vpc.tf

resource "aws\_vpc" "my\_app" {

    cidr\_block       = "${var.vpc\_cidr}" -🡪refeering the value from variables.tf

  instance\_tenancy = "default"

  tags = {

    Name = "JavahomeVpc" 🡪we can give tags dynamically but we are hardcodding here

Environment = "${terraform.workspace}" 🡪using current workspace for envi key

}

}

resource "aws\_vpc" "my\_app" {

    cidr\_block       = "${var.vpc\_cidr}"

  instance\_tenancy = "default"

  tags = {

    Name = "JavahomeVpc"

    Environment = "${terraform.workspace}"

  }

}

**Terraform Creating Public subnets**

For high availability I want to have multiple subnets spanning multiple availability zones for making those subnets public we have to create internet gateway and configure them part of route tables , create a separate file for public subnets

Public-subnets.tf

locals {

  az\_names = "${data.aws\_availability\_zones.azs.names}" 🡪this expresion is repeating many times so we used it as local variables, so this gives all availability zones

pub\_sub\_ids = "${aws\_subnet.public.\*.id}" it is multiple subnets so we access them using \* ,----pu\_sub\_ids🡪 added this expression as local variable and reffered it in subnet association section

}

resource "aws\_subnet" "public" {

  count             = "${length(local.az\_names)}"----🡪 I take count variable to have public subnet in every a.zone,length—is used to find list of a.z’s,we are reffering local variable as local.variable name (eg.loacal.az\_names)

  vpc\_id            = "${aws\_vpc.my\_app.id}"

  cidr\_block        = "${cidrsubnet(var.vpc\_cidr, 8, count.index)}"-🡪we do want cidr blocks dynamically one option is declare a variable with multiple values other option is we have built-in options which can generate this cidrblocks dynamically at run time

  availability\_zone = "${local.az\_names[count.index]}" 🡪we reffered local variable here also

map\_public\_ip\_on\_launch = true🡪this option is for assigning public ip automatically for public subnets

  tags = {

    Name = "PublicSubnet-${count.index + 1}" 🡪to have different names for different az’s using count.index,indexes begins with 0, but we want to staert with 1 so we give count.index + 1

  }

}

Internet Gateway

resource "aws\_internet\_gateway" "igw" { igw🡪user given name for resource

  vpc\_id = "${aws\_vpc.my\_app.id}"-🡪reffering my\_app vpc

  tags = {

    Name = "JavaHome-internet-gateway" ----🡪this resource section is creating internet gateway

  }

}

Creating separate route table for public subnet associations

 resource "aws\_route\_table" "prt" { (prt- public route table)

  vpc\_id = "${aws\_vpc.my\_app.id}" (my\_app—logical name of vpc)

we are adding a route to a prt route table

  route {

    cidr\_block = "0.0.0.0/0"🡪this is internet gateway so all should be 0.0.0.0/0

    gateway\_id = "${aws\_internet\_gateway.igw.id}" 🡪igw is logical name of i.gateway

  }

  tags = {

    Name = "JavaHomePRT"

  }

}

Subnet association

resource "aws\_route\_table\_association" "pub\_sub\_association" {-🡪subnet association logical name

  count          = "${length(local.az\_names)}"🡪how to get all public subnet ids dynamically and loop it or count it

  subnet\_id      = "${local.pub\_sub\_ids[count.index]}"pu\_sub\_ids is alocal variable mentioned in the local variable section (\* returns back list of subnet ids,from that list from that list we should access one subnet at a time in the loop using count.index

route\_table\_id = "${aws\_route\_table.prt.id}" -🡪route table id prt,same for all subnets

}

locals {

  az\_names = "${data.aws\_availability\_zones.azs.names}"

  pub\_sub\_ids = "${aws\_subnet.public.\*.id}"

}

resource "aws\_subnet" "public" {

  count                   = "${length(local.az\_names)}"

  vpc\_id                  = "${aws\_vpc.my\_app.id}"

  cidr\_block              = "${cidrsubnet(var.vpc\_cidr, 8, count.index)}"

  availability\_zone       = "${local.az\_names[count.index]}"

  map\_public\_ip\_on\_launch = true

  tags = {

    Name = "PublicSubnet-${count.index + 1}"

  }

}

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

  vpc\_id = "${aws\_vpc.my\_app.id}"

  tags = {

    Name = "JavaHome-internet-gateway"

  }

}

 resource "aws\_route\_table" "prt" {

  vpc\_id = "${aws\_vpc.my\_app.id}"

  route {

    cidr\_block = "0.0.0.0/0"

    gateway\_id = "${aws\_internet\_gateway.igw.id}"

  }

  tags = {

    Name = "JavaHomePRT"

  }

}

resource "aws\_route\_table\_association" "pub\_sub\_association" {

  count          = "${length(local.az\_names)}"

  subnet\_id      = "${local.pub\_sub\_ids[count.index]}"

  route\_table\_id = "${aws\_route\_table.prt.id}"

}

We need multiple public subnets for high availability

How to get availability zones dynamically for that we use Data sources..

**Terraform Data Sources**

Data sources helps us to import certain information, which is declared outside from terraform configurations

datasources.tf

#Decalare the datasource

data "aws\_availability\_zones" "azs"{

}

Data sources is going to return list of zones in current regions.

terraform apply –auto-approve

and now check in vpc section public subnets we can find 6 public subnets formed as 0,1,----5.

**We need to make this public subnet for that we need to create internet gateway**

Internet Gateway

resource "aws\_internet\_gateway" "igw" { igw🡪user given name for resource

  vpc\_id = "${aws\_vpc.my\_app.id}"-🡪reffering my\_app vpc

  tags = {

    Name = "JavaHome-internet-gateway" ----🡪this resource section is creating internet gateway

  }

}

**Creating separate Route table for public subnet Associations**

 resource "aws\_route\_table" "prt" { (prt- public route table)

  vpc\_id = "${aws\_vpc.my\_app.id}" (my\_app—logical name of vpc)

we are adding a route to a prt route table

  route {

    cidr\_block = "0.0.0.0/0"🡪this is internet gateway so all should be 0.0.0.0/0

    gateway\_id = "${aws\_internet\_gateway.igw.id}" 🡪igw is logical name of i.gateway

  }

  tags = {

    Name = "JavaHomePRT"

  }

}

**Subnet Association**

locals {

  az\_names = "${data.aws\_availability\_zones.azs.names}" 🡪this expresion is repeating many times so we used it as local variables, so this gives all availability zones

pub\_sub\_ids = "${aws\_subnet.public.\*.id}" it is multiple subnets so we access them using \* ,----pu\_sub\_ids🡪 added this expression as local variable and reffered it in subnet association section

}

Subnet association

resource "aws\_route\_table\_association" "pub\_sub\_association" {-🡪subnet association logical name

  count          = "${length(local.az\_names)}"🡪how to get all public subnet ids dynamically and loop it or count it

  subnet\_id      = "${local.pub\_sub\_ids[count.index]}"pu\_sub\_ids is alocal variable mentioned in the local variable section (\* returns back list of subnet ids,from that list from that list we should access one subnet at a time in the loop using count.index

route\_table\_id = "${aws\_route\_table.prt.id}" -🡪route table id prt,same for all subnets

}

terraform apply –auto-approve

o/p:Go to aws console and in vpc section, check internet gateway Is there and in Route tables JavaHomePRT route table is there not .

\*\*\*\*6 subnets got associated and ther should be 1 route which has integration to internet gateway \*\*\*\*\*\*\*\*\*\*\*

**Terraform Private subnets**

Now we want to create private subnets ,create a file called private-subnets.tf in the same folder

Private-subnets.tf

resource "aws\_subnet" "private" {🡪p.subnet resource logical name private

  count             = "${length(slice(local.az\_names, 0, 2))}"🡪cmg to p.subnets we can leave it as public.subnet ,it is not wrong to create 6 p.subnets means each pvt.subunet for a.zone, but I want to create 2 pvt.subnets and when we gona setup RDS ,if we want to enable multi Az’s we maximum need 2 pvt.subnets there, we are introducing slice function here (using slice we can create sublist (slice(local.az\_names, 0, 2)) slice,azs list is the first argument ,based on how many elements we want from this list we need to specify those values ,from the az’s list pick 0 as the index means pvt.subnet 0, pvt.subnet 1, ending index is exclusive as we said its going to give index 0 and index 1, which the count is 2.

  vpc\_id            = "${aws\_vpc.my\_app.id}"

  cidr\_block        = "${cidrsubnet(var.vpc\_cidr, 8, count.index + length(local.az\_names))}" -🡪 we need to begin from where we left at public.subnets ,copy the public subnets + length(local.az\_names) and it as this "${cidrsubnet(var.vpc\_cidr, 8, count.index + length(local.az\_names))}" hence it will pick from where the public subnets ends

  availability\_zone = "${local.az\_names[count.index]}"

  tags = {

    Name = "PrivateSubnet-${count.index + 1}"-🡪p.subnet resource tag name PrivateSubnet

  }

}

resource "aws\_subnet" "private" {

  count             = "${length(slice(local.az\_names, 0, 2))}"

  vpc\_id            = "${aws\_vpc.my\_app.id}"

  cidr\_block        = "${cidrsubnet(var.vpc\_cidr, 8, count.index + length(local.az\_names))}"

  availability\_zone = "${local.az\_names[count.index]}"

  tags = {

    Name = "PrivateSubnet-${count.index + 1}"

  }

}

Here we are not going- to do any route table association because we have default route table which doesn’t have internet gateway configured, so its going to be private by default

So we count.index + length(local.az\_names))}" expresiion here we can make it as local variable and we can refer it

terraform apply -auto-approve

**Terraform Enable Subnet settings For auto assigning Public Ip**

For public subnets I want to assign public ip automatically I want to do subnet settings for this

For that go to Terraform documentation subnet terraform in google , go to Argument Refference

* [[map\_public\_ip\_on\_](https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/subnet" \l "map_customer_owned_ip_on_launch)](https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/subnet#map_public_ip_on_launch)[launch](https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/subnet" \l "map_customer_owned_ip_on_launch) - (Optional) Specify true to indicate that instances launched into the subnet should be assigned a public IP address. Default is false.

I want this settings enabled for public subnet

In public-subnet.tf in public subnet section

resource "aws\_subnet" "public" {

  count                   = "${length(local.az\_names)}"

  vpc\_id                  = "${aws\_vpc.my\_app.id}"

  cidr\_block              = "${cidrsubnet(var.vpc\_cidr, 8, count.index)}"

  availability\_zone       = "${local.az\_names[count.index]}"

  map\_public\_ip\_on\_launch = true-------🡪

  tags = {

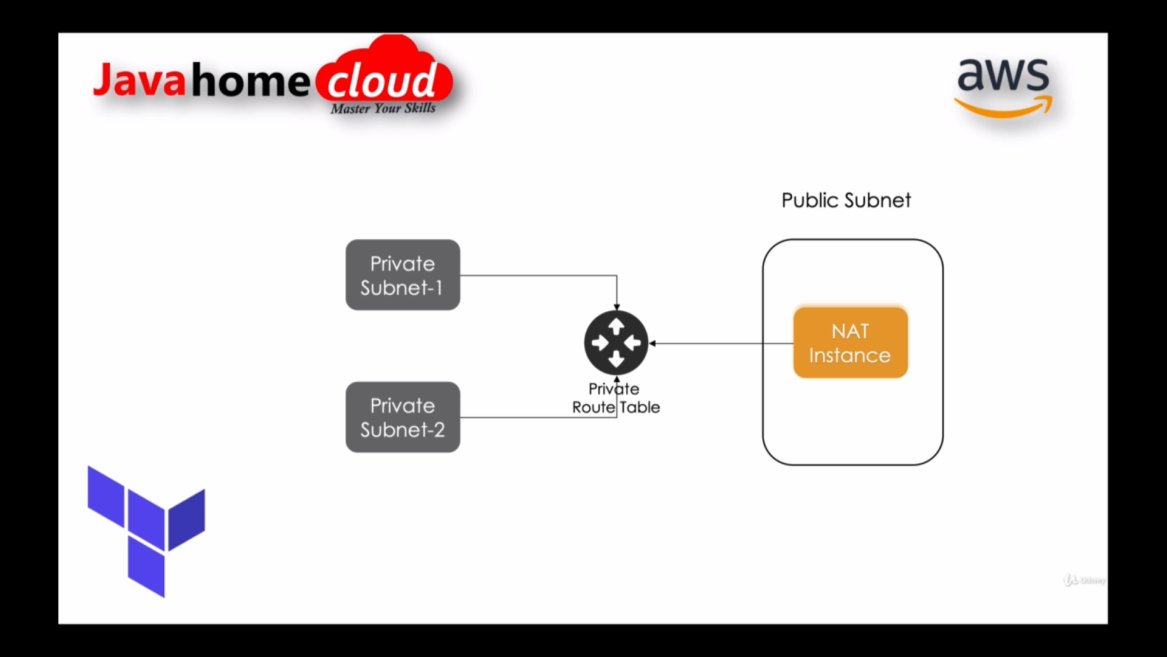
    Name = "PublicSubnet-${count.index + 1}"

  }

}

**Terraform Configue NAT instance**

Configuring nat instance, we want to configure internet access to private subnets , we are going to use nat instances for that, we do have options to use nat gateways , for this demo we are going with nat instances



We already have two private subnets,

1.we should create route table for this 2 pvt.subnets.

2.And launch nat instance in public subnet .

3. And configure internet access to private subnets.

For launching nat instance go to ec2 terraform documentation, we need ami, instance type, tags, you also should mention which subnet it should go into(in our case public subnet) , we do need public ip then we should disable source-destination check .

In priate-subnet.tf

resource "aws\_subnet" "private" {

  count             = "${length(slice(local.az\_names, 0, 2))}"

  vpc\_id            = "${aws\_vpc.my\_app.id}"

  cidr\_block        = "${cidrsubnet(var.vpc\_cidr, 8, count.index + length(local.az\_names))}"

  availability\_zone = "${local.az\_names[count.index]}"

  tags = {

    Name = "PrivateSubnet-${count.index + 1}"

  }

}

Nat instance section:

resource "aws\_instance" "nat" {

  ami               = "${var.nat\_amis[var.region]}"🡪to get nat instances ami ids we declared variables in variables.tf, for evry region we should have different ami id because ami is region specific,we have given 2 ami ids , so it supports 2 regions,if you want to support all regions then give all regions ami ids, so ami ids are accessed from variables.tf [var.region] represents current region

  instance\_type     = "t3.micro"

  subnet\_id         = "${local.pub\_sub\_ids[0]}"🡪public subnet id ,we have multiple subnet ids, so this expresiion is already in local variable so we reffred it from local variable,from that lis first one 0—reprsents first index

  source\_dest\_check = false

  tags = {

    Name = "JavaHomeNat"🡪tag as javahomenat

  }

}

Creating route table and add nat instance to the route table:we are creating one route table it contains one route with nat instance and cidr block 0.0.0.0/0 open to internet

resource "aws\_route\_table" "privatert" { 🡪private route table

  vpc\_id = "${aws\_vpc.my\_app.id}"🡪vpc id pointing to my\_app

  route {

    cidr\_block  = "0.0.0.0/0"-🡪open to internet

    instance\_id = "${aws\_instance.nat.id}" 🡪 instance id pointing to nat instance

  }

  tags = {

    Name = "JavaHomePrivateRT"-🡪private route table tag

  }

}

Associating the above route table with private subnets

resource "aws\_route\_table\_association" "private\_rt\_association" {🡪private rt association

  count          = "${length(slice(local.az\_names, 0, 2))}"🡪we have multiple private subnets so we have count

  subnet\_id      = "${aws\_subnet.private.\*.id[count.index]}"-🡪pointing to private subnet,it is list of pvt.subnets so we use \*.id[count.index]

  route\_table\_id = "${aws\_route\_table.privatert.id}"🡪pointing to above created private route table

}

In variable.tf we declared nat ami ids

variable "vpc\_cidr" {

    description = "choose cidr for vpc"

    type        = "string"

    default     = "10.20.0.0/16"

}

variable "region" {

    description = "choose region for your stack"

    type        = "string"

    default     = "ap-south-1"

}

variable "nat\_amis" {

  type    = "map"

  default =

  us-east-1 = "ami-00a9d4a05375b2763"

  us-east-2 = "amo-00d1f8201864cc10c" -🡪 nat ami ids

}

Private-subnet.tf

resource "aws\_subnet" "private" {

  count             = "${length(slice(local.az\_names, 0, 2))}"

  vpc\_id            = "${aws\_vpc.my\_app.id}"

  cidr\_block        = "${cidrsubnet(var.vpc\_cidr, 8, count.index + length(local.az\_names))}"

  availability\_zone = "${local.az\_names[count.index]}"

  tags = {

    Name = "PrivateSubnet-${count.index + 1}"

  }

}

resource "aws\_instance" "nat" {

  ami               = "${var.nat\_amis[var.region]}"

  instance\_type     = "t3.micro"

  subnet\_id         = "${local.pub\_sub\_ids[0]}"

  source\_dest\_check = false

  tags = {

    Name = "JavaHomeNat"

  }

}

resource "aws\_route\_table" "privatert" {

  vpc\_id = "${aws\_vpc.my\_app.id}"

  route {

    cidr\_block  = "0.0.0.0/0"

    instance\_id = "${aws\_instance.nat.id}"

  }

  tags = {

    Name = "JavaHomePrivateRT"

  }

}

resource "aws\_route\_table\_association" "private\_rt\_association" {

  count          = "${length(slice(local.az\_names, 0, 2))}"

  subnet\_id      = "${aws\_subnet.private.\*.id[count.index]}"

  route\_table\_id = "${aws\_route\_table.privatert.id}"

}

terraform apply –auto-approve

o/p: go to aws console🡪public subnets created, route tables javahomeprivate route table should be created and 2 subnets associated ,in Routes we have Nat instance configured.

That’s how we setup nat instances using terraform.

**Terraform Configue Security Groups For NAT Instance**

We have to secure nat instances using security groups

Private-subnets.tf

resource "aws\_subnet" "private" {

  count             = "${length(slice(local.az\_names, 0, 2))}"

  vpc\_id            = "${aws\_vpc.my\_app.id}"

  cidr\_block        = "${cidrsubnet(var.vpc\_cidr, 8, count.index + length(local.az\_names))}"

  availability\_zone = "${local.az\_names[count.index]}"

  tags = {

    Name = "PrivateSubnet-${count.index + 1}"

  }

}

resource "aws\_instance" "nat" {

  ami                    = "${var.nat\_amis[var.region]}"

  instance\_type          = "t3.micro"

  subnet\_id              = "${local.pub\_sub\_ids[0]}"

  source\_dest\_check      = false

  vpc\_security\_group\_ids = ["${aws\_security\_group.nat\_sg.id}"]—associating security groups with nat instance security group id followed by logical name

  tags = {

    Name = "JavaHomeNat"

  }

}

resource "aws\_route\_table" "privatert" {

  vpc\_id = "${aws\_vpc.my\_app.id}"

  route {

    cidr\_block  = "0.0.0.0/0"

    instance\_id = "${aws\_instance.nat.id}"

  }

  tags = {

    Name = "JavaHomePrivateRT"

  }

}

resource "aws\_route\_table\_association" "private\_rt\_association" {

  count          = "${length(slice(local.az\_names, 0, 2))}"

  subnet\_id      = "${aws\_subnet.private.\*.id[count.index]}"

  route\_table\_id = "${aws\_route\_table.privatert.id}"

}

Securing nat instances using security group

resource "aws\_security\_group" "nat\_sg" { -🡪sgroup logica name nat\_sg

  name        = "nat\_sg"

  description = "Allow trafiic for private subnets"

  vpc\_id      = "${aws\_vpc.my\_app.id}"-🡪 vpc id my\_app

  egress { 🡪egress outbound traffic, where servers in private subnet controlled

    from\_port   = 0

    to\_port     = 0-🡪 from-to port 0 means allow all ports

    protocol    = "-1" -1 means all protocols

    cidr\_blocks = ["0.0.0.0/0"] 🡪cidr blocks 0 means keep it open for internet

  }

  tags = {

    Name = "nat\_security\_groups"

  }

}

resource "aws\_subnet" "private" {

  count             = "${length(slice(local.az\_names, 0, 2))}"

  vpc\_id            = "${aws\_vpc.my\_app.id}"

  cidr\_block        = "${cidrsubnet(var.vpc\_cidr, 8, count.index + length(local.az\_names))}"

  availability\_zone = "${local.az\_names[count.index]}"

  tags = {

    Name = "PrivateSubnet-${count.index + 1}"

  }

}

resource "aws\_instance" "nat" {

  ami                    = "${var.nat\_amis[var.region]}"

  instance\_type          = "t3.micro"

  subnet\_id              = "${local.pub\_sub\_ids[0]}"

  source\_dest\_check      = false

  vpc\_security\_group\_ids = ["${aws\_security\_group.nat\_sg.id}"]

  tags = {

    Name = "JavaHomeNat"

  }

}

resource "aws\_route\_table" "privatert" {

  vpc\_id = "${aws\_vpc.my\_app.id}"

  route {

    cidr\_block  = "0.0.0.0/0"

    instance\_id = "${aws\_instance.nat.id}"

  }

  tags = {

    Name = "JavaHomePrivateRT"

  }

}

resource "aws\_route\_table\_association" "private\_rt\_association" {

  count          = "${length(slice(local.az\_names, 0, 2))}"

  subnet\_id      = "${aws\_subnet.private.\*.id[count.index]}"

  route\_table\_id = "${aws\_route\_table.privatert.id}"

}

resource "aws\_security\_group" "nat\_sg" {

  name        = "nat\_sg"

  description = "Allow trafiic for private subnets"

  vpc\_id      = "${aws\_vpc.my\_app.id}"

  egress {

    from\_port   = 0

    to\_port     = 0

    protocol    = "-1"

    cidr\_blocks = ["0.0.0.0/0"]

  }

  tags = {

    Name = "nat\_security\_groups"

  }

}

terraform apply –auto-approve

o/p: it has created security group, and it updated nat instance for associating this security group .

nat instance pointing to nat security group with the rules we added.

**Terraform Ec2 instances**

**Launching ec2 instances in public subnets**

we create ec2 instances in public subnet and use user-data to install apache web server and launch a website .

I want to parameterized instance type, tags, ami’s, so open variables.tf file

variables.tf

variable "vpc\_cidr" {

    description = "choose cidr for vpc"

    type        = "string"

    default     = "10.20.0.0/16"

}

variable "region" {

    description = "choose region for your stack"

    type        = "string"

    default     = "us-east-1"

}

variable "web\_ec2\_count" { ---reffering this variable in web-ec2.tf for web servers, for having 2 web server instances in 2 azones

    description = "choose number of instances for web "

    type        = "string"

    default     = "2" -🡪2 instances ,we can change it dynaamically

}

variable "nat\_amis" {

  type    = "map"

  default =

  us-east-1 = "ami-00a9d4a05375b2763"

  us-east-2 = "amo-00d1f8201864cc10c"

}

Mapping of amis for webservers from vraibles.tf

variable "web\_amis" { -🡪mapping amis for web servers

  type    = "map"

  default =

  us-east-1 = "ami-00a9d4a05375b2763"

  us-east-2 = "amo-00d1f8201864cc10c"

}

Parameterizing web server amis,tags,instance,type

variable "web\_instance\_type" {--🡪web servers instance type

    description = "choose instance type for your web"

    type        = "string"

    default     = "t2.micro"----🡪 instance type

}

Supplying tags for web servers from variables

variable "web\_tags" { -🡪 web tags name

  type    = "map"

  default =

  Name = "Webserver"

}

Inside variables, we cant use terraform interpolation

web-ec2.tf

locals {----🡪as part of web tags we don’t need environment(dev,prod) which points to current workspace, so we are managing that by using local variable, we call it as env\_tags which is a map to environment = terraformworkspace

  env\_tags = {

    Environment = "${terraform.workspce}"

  }

---mapping env tags with web tags for that we have built in terraform function for merging called merge which merges two maps ,to get merge function we need to refer terraform interpolation documentation

  web\_tags = "${merge(var.web\_tags, local.env\_tags)}"

}

resource "aws\_instance" "web" {

  count         = "${var.web\_ec2\_count}"🡪count represents having 2 instances in 2 different a.zones, and count kept as variable, reffering from variables.

  ami           = "${var.web\_amis[var.region]}"-🡪reffering/mapping amis from variables.tf, [var.region] refers current region

  instance\_type = "${var.web\_instance\_type}"-🡪reffering instance type from variables.tf

  subnet\_id     = "${local.public\_subnet\_ids[count.index]}"🡪 linking web instances to public subnets, we need get subnet ids, but we have multiple subnets so we need to get one by one so we use count [count.index], and we get public subnet ids by reffering local.public\_subnet\_ids—from public-subnets.tf local variable section.

  tags = {

    Name = "${local.web\_tags}"-🡪this tags coming from variable and the one declared in locals in web-ec2.tf

  }

}

locals {

  env\_tags = {

    Environment = "${terraform.workspce}"

  }

  web\_tags = "${merge(var.web\_tags, local.env\_tags)}"

}

resource "aws\_instance" "web" {

  count         = "${var.web\_ec2\_count}"

  ami           = "${var.web\_amis[var.region]}"

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

  subnet\_id     = "${local.public\_subnet\_ids[count.index]}"

  tags = {

    Name = "${local.web\_tags}"

  }

}

terraform apply –auto-approve

o/p:we have created ec2 instances in public subnets

**Terraform Ec2 user-data**

we have created ec2 instances in public subnets, now we want to put web server and a sample website on that ec2 instances, we are doing that using user data, we want to do that using user data script

create a separate folder in atom editor named scripts and create a file apache.sh which is a shell script

#!/bin/sh

yum -y install httpd

systemctl enable httpd

systemctl start httpd.service

echo "<h1> Murali Java Home app</h1>" > /var/www/html/index.html

now we need to pass this scripts where you use user data in web-ec2.tf

web-ec2.tf

locals {

  env\_tags = {

    Environment = "${terraform.workspce}"

  }

  web\_tags = "${merge(var.web\_tags, local.env\_tags)}"

}

resource "aws\_instance" "web" {

  count         = "${var.web\_ec2\_count}"

  ami           = "${var.web\_amis[var.region]}"

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

  subnet\_id     = "${local.public\_subnet\_ids[count.index]}"

  tags          = "${local.web\_tags}"

passing the apache.sh script using user data

file function and provide path of the file scripts/apache.sh, file function reads the content from script and assigns that to user data

  user\_data     = "${file("scripts/appache.sh")}"----

}

Varaibeles.tf

locals {

  env\_tags = {

    Environment = "${terraform.workspce}"

  }

  web\_tags = "${merge(var.web\_tags, local.env\_tags)}"

}

resource "aws\_instance" "web" {

  count         = "${var.web\_ec2\_count}"

  ami           = "${var.web\_amis[var.region]}"

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

  subnet\_id     = "${local.public\_subnet\_ids[count.index]}"

  tags          = "${local.web\_tags}"

  user\_data     = "${file("scripts/appache.sh")}"

}

terraform apply –auto-approve

**Terraform creating s3 bucket**

creating s3 bucket using Terraform

s3.tf

resource "aws\_s3\_bucket" "my\_bucket" { -----🡪resource aws bucket, logica=l name my\_bucket

  bucket = "${var.my\_app\_s3\_bucket}"-🡪reffered in varaible.tf under web\_instance section

  acl    = "private"

  region ="${var.region}" 🡪 region referred from variable.tf

  tags = {

    Name        = "javahome-app-de"

    Environment =

  }

}

I want this bucket in the region where we are creating our app

Variables.tf

variable "vpc\_cidr" {

    description = "choose cidr for vpc"

    type        = "string"

    default     = "10.20.0.0/16"

}

variable "region" {

    description = "choose region for your stack"

    type        = "string"

    default     = "us-east-1"

}

variable "web\_ec2\_count" {

    description = "choose number of instances for web"

    type        = "string"

    default     = ""

}

variable "nat\_amis" {

  type    = "map"

  default =

  us-east-1 = "ami-00a9d4a05375b2763"

  us-east-2 = "amo-00d1f8201864cc10c"

}

variable "web\_amis" {

  type    = "map"

  default =

  us-east-1 = "ami-00a9d4a05375b2763"

  us-east-2 = "amo-00d1f8201864cc10c"

}

variable "web\_instance\_type" {

    description = "choose instance type for your web"

    type        = "string"

    default     = "t2.micro"

}

variable "my\_app\_s3\_bucket"{---------🡪 variable name

  default ="java-home-app-dev"

}

variable "web\_tags" {

  type    = "map"

  default =

  Name = Webserver"

}

terraform apply –auto-approve

o/p: s3 Bucket got created

s3.tf

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

  bucket = "${var.my\_app\_s3\_bucket}"

  acl    = "private"

  region ="${var.region}"

  tags = {

    Name        = "javahome-app-de"

    Environment =

  }

}

**Terraform – Create IAM ROLE**

Creating IAM role for ec2 instances, using iam role applications running on ec2 instances will get access to s3 for uploading objects.

We need to create

1. Assume Role, then
2. IAM policy
3. IAM Role, Followed by create IAM Role itself by linking assume role and iam policy .

Create Iam folder inside scripts folder and kept 2 json files

Web-ec2-assume-role.json

{

  "Version": "2012-10-17",

  "Statement": [

    {

      "Effect": "Allow",

      "Principal": {

        "Service": "ec2.amazonaws.com",

     },

      "Action": "sts:AssumeRole"

    }

  ]

}

Web-ec2-policy.json

{

  "Version": "2012-10-17",

  "Statement": {

    "Sid": "Smt1563521329921",

    "Action":[

      "s3:PutObject"

    ],

    "Effect": "Allow",

    "Resource": "${s3\_bucket\_arn}"-🡪 I want to replace this s3 bucket arn dynamically based on the workspace we are using, so we cant keep this as a plain text or static file, it should be dynamic, we have terraform template files using s3\_bucket\_arn this piece of information can be dynamically replaced

  }

}

Web-ec2-policy.json

{

  "Version": "2012-10-17",

  "Statement": {

    "Sid": "Smt1563521329921",

    "Action":[

      "s3:PutObject"

    ],

    "Effect": "Allow",

    "Resource": "${s3\_bucket\_arn}"

  }

}

Lets also have separate file for IAM role ec2-iam.tf is created in terraform scripts folder

Search in Google terraform data template\_file, we need to pass the file which has place holders so in our case we have bucket name there and the values

## [Example Usage](https://registry.terraform.io/providers/hashicorp/template/latest/docs/data-sources/file" \l "example-usage)

data "template\_file" "init" {

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

vars = {

consul\_address = "${aws\_instance.consul.private\_ip}"

}

}

Right now we are referring to template file

data "template\_file" 🡪template file "init"🡪 logical name

template = "${file("${path.module}/init.tpl")}"🡪 we need to pass the file which has place holders so in our case we have bucket name there.

vars = {

consul\_address = "${aws\_instance.consul.private\_ip}"

}

--🡪 And the Values, otherwise the variables that should be replaced inside the file init.tpl file

ec2-iam.tf

data "template\_file" "s3\_web\_policy" { -🡪s3 web policy

  template = "${file("scripts/iam/web-ec2-policy.json")}"-🡪 name of the template file (web-ec2-policy.json template file name), we have this in path scripts/iam/web-ec2-policy.json

  vars = {

    s3\_bucket\_arn = "${arn:aws:s3:::${var.my\_app\_s3\_bucket}/\*"     s3\_bucket\_arn 🡪 this variable is from scripts/iam/web-ec2-policy.json

I want to grant PutObject access only on a specific s3 bucket, so I need to pick specific arn and I need to put / \* which indicates that it can put objects into specific bucket .

"${arn:aws:s3:::${var.my\_app\_s3\_bucket}/\*" --🡪

This template file is reading this scripts/iam/web-ec2-policy.json, which has this variable s3\_bucket\_arn  and we are passing that variable with this value arn:aws:s3:::${var.my\_app\_s3\_bucket, this data will return back a plain text of replacinf that place holder.

  }

}

To Create IAM role policy and attach that iam role policy to ec2 instance, google iam role policy for terraform

ROLE POLICY

resource "aws\_iam\_role\_policy" "s3\_test\_policy" {

  name = "s3\_test\_policy" 🡪 name of the iam role policy

  role = "${aws\_iam\_role.s3\_test\_role.id}"🡪role it should associate with is assume\_role\_policy  which is in the below assume role section

  policy = "${data.template\_file.s3\_web\_policy.rendered}" -🡪 I need to load the policy, I need to get that translated policy document from data.template file ,data. Type of data.logical name (data.template\_file.s3\_web\_policy.rendered) we use rendered such that it will give back the rendered json document for us, which has the bucket name dynamically replaced

}

resource "aws\_iam\_role" "s3\_test\_role" {

  name = "s3\_test\_role"

  assume\_role\_policy = "${file("scripts/iam/web-ec2-assume-role.json")}" 🡪 lets read that from web-ec2-assume-role.json file and the path of the file, we are loading role from this path and assigning it to the assume\_role\_policy

}

now the role is created and now attach this role to ec2 instance , for attaching iam role to ec2 instance we have to create instance profile, search terraform instance profile in Google.

Attaching iam role to ec2 instance, use this instance profile inside ec2-iam.tf for linking iam role

resource "aws\_iam\_instance\_profile" "test\_profile" {

name = "test\_profile"

role = “${aws\_iam\_role.s3\_ec2\_role.name}”

}

aws\_iam\_instance\_profile lets take this and go to web-ec2.tf

ec2-iam.tf

data "template\_file" "s3\_web\_policy" {

  template = "${file("scripts/iam/web-ec2-policy.json")}"

  vars = {

    s3\_bucket\_arn = "${arn:aws:s3:::${var.my\_app\_s3\_bucket}/\*"

  }

}

resource "aws\_iam\_role\_policy" "s3\_test\_policy" {

  name = "s3\_test\_policy"

  role = "${aws\_iam\_role.s3\_test\_role.id}"

  policy = "${data.template\_file.s3\_web\_policy.rendered}"

}

resource "aws\_iam\_role" "s3\_test\_role" {

  name = "s3\_test\_role"

  assume\_role\_policy = "${file("scripts/iam/web-ec2-assume-role.json")}"

}

resource "aws\_iam\_instance\_profile" "test\_profile" {

name = "test\_profile"

role = “${aws\_iam\_role.s3\_ec2\_role.name}”

}

Web-ec2.tf

locals {

  env\_tags = {

    Environment = "${terraform.workspce}"

  }

  web\_tags = "${merge(var.web\_tags, local.env\_tags)}"

}

resource "aws\_instance" "web" {

  count                = "${var.web\_ec2\_count}"

  ami                  = "${var.web\_amis[var.region]}"

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

  subnet\_id            = "${local.public\_subnet\_ids[count.index]}"

  tags                 = "${local.web\_tags}"

  user\_data            = "${file("scripts/appache.sh")}"

TYPE logical-name

  iam\_instance\_profile =  "${aws\_iam\_instance\_profile.s3\_ec2\_profile.name}"

-🡪 IAM instance profile

}

Web-ec2.tf

locals {

  env\_tags = {

    Environment = "${terraform.workspce}"

  }

  web\_tags = "${merge(var.web\_tags, local.env\_tags)}"

}

resource "aws\_instance" "web" {

  count                = "${var.web\_ec2\_count}"

  ami                  = "${var.web\_amis[var.region]}"

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

  subnet\_id            = "${local.public\_subnet\_ids[count.index]}"

  tags                 = "${local.web\_tags}"

  user\_data            = "${file("scripts/appache.sh")}"

  iam\_instance\_profile =  "${aws\_iam\_instance\_profile.s3\_ec2\_profile.name}"

}

terraform apply –auto-approve 🡪 role got successfully created and it got attached to the ec2 instances, to verify go to aws management console ec2 section🡪running instances🡪web server instance🡪actions 🡪 instance settings 🡪attach/replace IAM Role 🡪 and s3\_ec2\_profile attached successfully.