**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