Tools to write code (infrastructure as code tools)

Cloud formation tool- AWS

Resource manager- Azure

**Why Terraform:**

There are some tools like cloud formation, resource manage to automate the code. Like that all cloud providers has their own tools to create the infrastructure as a code. As a devops engineer, its difficult to learn all the tools and remember. So instead of learning all the tools we are introduced to terraform tool which will help us to create the infrastructure. We just need to tell terraform that on which service provider we need the infrastructure.

In terraform we will learn HCL. Then this hcl will be convert the code as per our requirement (aws, azure).

To configure terraform in AWS, first we need to have aws cli installed.

1st project in terraform for creating the EC2:

provider "aws"{

region = "us-east-1" # Set your desired AWS region

}

resource "aws\_instance" "example"

{

ami = "ami-0c55b159cbfafe1f0" # Specify an appropriate AMI ID

instance\_type = "t2.micro"

}

**Explanation of above code:**

1st line- here we need to mention for which service provider either its azure, GCP or AWS we are creating the terraform code)

2nd line- need to mention the region name where we want our resources to be created.

3rd line- need to mention the resources which we want to create. We need to create as many resource block as much we need to resource. For ex: if we are creating 5 resources we need to write 5 resource blocks like above.

To write the resource block syntaxes, we can make use of [Docs overview | hashicorp/aws | Terraform | Terraform Registry](https://registry.terraform.io/providers/hashicorp/aws/latest/docs) here we need to search for the resource which we want to create. We will be having basic to advanced syntax of the each resources.

For ex, if we need ec2 instance, search for the same and click on it.

To write code or use terraform we need to have visual studio code. Then inside the terminal we need to configure aws.

**Aws configure-** once we enter this command it will be asking few things like access key id, secret code, region and format. we need to provide all the details.

Terraform init: used to initialize terraform in a particular folder where we have our code.

**Terraform plan** – is a command which will tell the overview about what is going to happen. Once we run this command its recommended to check all the details thoroughly. It will help us in avoiding making mistakes.

**Terraform apply-** is used to perform the actual task and asks for the confirmation to proceed.

**Terraform tfstate file**: the state file will be used to record whatever the tasks we performed earlier and help us next time when we updating something on the old file.

**Terraform destroy:** used to the resource which was created earlier.

**Life cycle of the terraform:**

Terraform init🡪 terraform plan🡪 terraform apply🡪 terraform destroy

**Advanced terraform configuration :**

**Providers def:** Providers is a plugin/ act as medium which helps Terraform to understand where it (terraform) has to create the infrastructure.

*In the terraform code, we always starts with provider block where we can mention on which cloud provider we are going to create the infrastructure.*

There are different syntaxes for different cloud providers. We can find all the providers list under [Browse Providers | Terraform Registry](https://registry.terraform.io/browse/providers)

There are 3 different categories providers in terraform.

1. Official- maintained by hasicorp. Ex:GCP,aws,azure
2. Partner- ex: alibaba, oracle
3. Community – open source will maintain

**Interview perspective,** there are two main things.

How we will write providers for multi region or multi cloud providers.

**MULTI REGION EX:** here we can make use of **alias** keyword.

provider "aws" {

alias = "us-east-1"

region = "us-east-1"

}

provider "aws" {

alias = "us-west-2"

region = "us-west-2"

}

resource "aws\_instance" "example" {

ami = "ami-0123456789abcdef0"

instance\_type = "t2.micro"

provider = "aws.us-east-1"

}

resource "aws\_instance" "example2" {

ami = "ami-0123456789abcdef0"

instance\_type = "t2.micro"

provider = "aws.us-west-2"

}

Alias can be named as anything we want. Here we need to remember, while writing the resource block.

**Provider =cloud provider+ alias\_ name**

**MULTI Providers:**

Here the provider syntax will be different for different providers. We can refer the links for the same. So based on that we will be writing the provider block for multi providers.

**Here are some other examples of providers:**

azurerm - for Azure- [azurerm\_virtual\_machine | Resources | hashicorp/azurerm | Terraform | Terraform Registry](https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/virtual_machine)

**The provider block for azure:**

provider "azurerm" {

subscription\_id = "your-azure-subscription-id"

client\_id = "your-azure-client-id"

client\_secret = "your-azure-client-secret"

tenant\_id = "your-azure-tenant-id"

}

google - for Google Cloud Platform

kubernetes - for Kubernetes

openstack - for OpenStack

vsphere - for VMware vSphere

**VARIABLEs**: are used to avoid hardcoding the values. There are 2 types of variables in terraform.

1. Input variable- the value which we are passing to terraform.
2. Output variable- the value which terraform gives

**Ex: input variable syntax for instance creation:**

Variable “instance type” {

Description= “EC2 instance type”

Type= string

Default type=”t2.micro”

}

**Ex: output variable syntax for instance creation:**

**Output “public ip”{**

Description: “public ip of the EC2 instance”

Value=”resource type.resource name.public ip *(aws\_instance.instance name.public ip)*

**Structure of the terraform project:**

main.tf

provider.tf

input.tf

output.tf

main.tf

terraform.tfvars

**terraform.tfvars-** Complete parameterize of our code: (tfvars)

We will write all the resource type, instance type inside the terraform.tfvars file. So based on the environment and requirement we can just go inside the terraform.tfvars and change accordingly. Whenever we are doing terraform apply the input variables will be taking the data from terraform.tfvars

**Conditional Expressions: like if else condition**

SYNTAX: condition ? true\_val : false\_val

**Ex:** cidr\_blocks = var.environment == "production" ? [var.production\_subnet\_cidr] : [var.development\_subnet\_cidr]

**Builtin functions:** there are some builtin functions available in terraform. Refer document

**Modules:**

**What is module in terraform?**

In terraform there is a chance that every resource or every line of the code is written inside the main.tf. like ec2,vpc, cloud formation, s3 and etc. if any one of the resource has any bug or something it’s difficult to find and fix. Also there are some other challenges like maintaining, ownership, testing. So terraform introduces modules /modular to make it simple.

*Whenever we creating modules, we no need to create .tfvars file.*

So by creating modules we can reuse the file whenever needed. The only thing is we need to wrire main.tf as per our requirement.

There are some modules available in public, we can use those on our own risk. [Terraform AWS modules | Terraform Registry](https://registry.terraform.io/namespaces/terraform-aws-modules)

**Module syntax:**

module “module\_name”{

source= “path where the file is available”

variables

}

**Terraform State:**

**What is statefile in terraform?**

Statefile is the heart of the terraform used to record or store all the information which infrastructure we create.

So whenever we are doing some update on existing infrastructure, terraform first will check the statefile and look for what has been done earlier, then compares the new file. After that terraform will update only the new changes to the infrastructure instead of creating another new one.

Statefile is also used while destroying the infrastructure.

**Drawbacks:**

1. Saves the sensitive information which is not needed.
2. If we are using version control, every time people checkout the code we are responsible to upload the statefile along with our code. And there is a chance that a devops engineers forgets to do that. In that case our entire configuration will be messed up.
3. The second drawback of using statefile in VC it may be accessed and modified by multiple people if it is not restricted.

**These drawbacks can be fixed.**

Terraform has a concept called remote backend. Using this remote backend we can store our statefile in any other places/ resources like S3 bucket. By doing this, we can avoid above drawbacks.

We can restrict the access of S3 bucket.

Also when devops engineer modifies something while checks in no need to upload the statefile again and again as those changes will be automatically updated in our statefile which is stored inside the S3 bucket.

*When we do terraform init, terraform.lock.hcl file will be created automatically. Then once we do terraform apply terraform.tfstate file will be created.*

**Terraform show** is a command used to show the output of terraform.

**Creation of backend in terraform:**

**Go to** [Backend Configuration - Configuration Language | Terraform | HashiCorp Developer](https://developer.hashicorp.com/terraform/language/settings/backends/configuration). Search for the resource which we need to create, just copy paste then use it.

1. Create a new file called backend.tf

Syntax:

terraform {

backend "s3" {

bucket = "mybucket"

key = "path/to/my/key" “latha/terraform.tfstate

region = "us-east-1"

}

}

**Locking mechanism in terraform**:

It’s useful when more than one person tries to update something on the same file. So to avoid any confusion terraform will lock the statefile till the first person completes his update then the lock will be released, then next person can proceeds with his changes or updates.

Only one person will be able to make the changes at a time.

Aws has concept called dynamo db which is used to implement the locking mechanism. For that we need to create one resource for dynamo db configuration.

**Terraform Provisioners:**

*Ingress- inbound, egress- outbound*

Using provisioners we can deploy applications in server/instance. There are 3 types of provisioners available.

1. File
2. Local-exec
3. Remote-exec

Are resources or concept in terraform lets us to copy or execute the code/or something during the creation or deletion.

By default, provisioner will run at the time of creation if we are not mentioning anything particularly. Under the provider block we can provide when the provisioner should run.

In terraform, without provisioners we cannot install/ run anything on EC2 instance. We can connect instance use either Ansible, shell and other options to do it which is critical/ tough compared to provisioners.

To overcome this, provisioners provides two types which is remote-exec and local-exec.

Remote-exec🡪 at the time of ec2 instance creation we can connect to instance and run any shell command to perfom any task.

Local-exec🡪 terminal is capable of showing only limited lines of output files to the users. But when we use local-exec terraform will create one separate file where all the output info will be stored inside that.

There is one another type called as file provisioners which is used for copying any files from our local.

We can refer the link for writing any provisioners. [Provisioner: remote-exec | Terraform | HashiCorp Developer](https://developer.hashicorp.com/terraform/language/resources/provisioners/remote-exec)

**Terraform workspaces:**

Scenario why terraform workspaces? Xyz team comes with a request for creating ec2 and s3 bucket for their project in dev environment.

So devops team created a module and gave readme file to the developers, using that file anyone can create the resources. But it cant be used for different environments like staging, preprod and etc as the statefile will get confused and it will only overriden the file with recent changes.

To overcome this terraform introduces workspaces which will maintain one single statefile inside one folder.for each environment one folder will be created along with the statefile within it.

To create a workspace- terraform workspace new dev/stage/etc…

To switch between the environments- terraform workspace select dev/stage/etc…

Whenever we use the workspace for different environment, different statefile will be created under each environment.

Instead of changing the terraform.tfvars file, we can write different .tfvars file for each environment like dev.tfvars, stage.tfvars, etc…

While running terraform apply just mention terraform apply –var-file=dev.tfvars

There is one another way is, without using the terraform.tfvars file under the main.tf inside the variable declaration, we can use map function.

variable "instance\_type" {

    description = "value of instance type"

    type = map(string)

    default = {

      "dev"="t2.micro"

      "stage"="t2.medium"

      "prod"="t2.xlarge"

    }

  instance\_type=lookup(var.instance\_type,terraform.workspace,"t2.micro")

Secret management in terraform: (to handle the sensitive information)

Refer the github for installation steps. Once the installation is done enable the security inbound rules for 8200.

As soon as we edit the inbound rules, we will be able to access the vault application.

Security engines are nothing but different types of secrets that we create in hashicorp. Or to store our key value pair we use KV secret engine.