Terraform is a proper infrastructure as a code tool, we can have version control by Github

Consistence infrastructure, infra will work in all Env, automated infra creation

CRUD-create, read, update, delete

Inventory management- we can easily see what infra we have

Cost management- create when required stop or delete when not required

Terraform modules – we can create modules and other projects can also use it to create resource from scratch

Declarative way of creating infra: whatever we write we will get if have correct syntax

No need to follow sequence in terraform

Terraform is a multi-cloud solution it can work many popular tools

Download terraform from browser-extract-terraform.exe-keep it in c/software folder

System env variables-add path in user env variables

Install AWS CLI

Terraform syntax:

HCL syantax – hashi corp configuration language

Variables

Loops

Conditions

Data Types

We need to tell terraform on which provider it should work, the version available in provider is a provider version

Create user in aws create access and secret key run aws configure

We need to run terraform command where ever .tf files are available that you need to deploy

Terraform init – it will initialize the provider

Terrafrom plan- it is a plan on what rewources are goin to create, we will review plan and if everything Is good we will go ahead for next steps

Terraform apply – it will create resources

Resource “what resource to create” “name your resource” {

}

Variables in terraform:

Variable “name of variable” {

Type = data-type

Default= “Srikanth-sg”

}

Defining variables in terraform:

We can create a variable.tf file and we can pass

Variables provided in variables.tf can be overridden by terraform.tfvars

Providing variables in command line is first preference

We can create tfvars with any name like roboshop.tfvars

We can pass variables in cmd like terraform plan –var=”instance\_type=t3.medium”

We can pass variable file name in cmd like terraform plan –var-file=”roboshop.tfvars”

# terraform will search the varibles values as per below flow

# First preference will be command line

# Second preference will be -var-file="" which we need to run with terraform plan and provide file name

# Third preference will be terraform.tfvars

# Last preference will be Environment variables

Conditions in terraform:

Conditions is used to if conditions is true run this or else run this

We can use condition for selecting instance type

Instance\_type= var.instance\_name == “MongoDB” ? “t3.small” :”t2.micro”

Loops in terraform:

Count based loop

For each loop

Output in terraform:

Terraform out put block is used to get the output from the created resources  
output is used to create another resource by using output as input of other

For example we will use ec2 IP out put to create Rout2 53 records

Functions in terraform:

Max , min function, join ()

Length function: we can use this function to get the length of list of instance\_names  
for ex: count = length(var.instance\_names)

We can use what functions are there in terraform we cant create by our own

Starts with function

Locals in terraform:

Locals is just like variables with some extra capabilities

You can keep functions and expressions, conditions in locals

Data sources in terraform:

Data sources is used to get some details from aws by querying them,

For ex I want ami id but ami id will change if any patch update is done do in that case we can query ami id by providing ami details other than id as an filter

Loops:

Count based loop used mostly iterating list

For each loop is used mostly to iterate map

State and remote state in terraform:

Whatever we write it will create – it is declarative

Declarative is nothing but our desired state

Current state- what terraform is created is current tfstate

Matching Desired state – current state

Terraform will ensure the terraform files matches tfstate and aws cloud

Whatever will be there in tfstate

In collaboration env if tfstate is in local then errors may come

We need a central state file to keep tfstate file

We can create S3 bucket and store tfstate file in bucket

We need to locking when ever changes are happening

We use dynamo db for locking S3 bucket

Dynamo db-table create-partition key (LockID)-create table

Keep backend S3 configuration we can give path bucket name in provider and dynamodb table name

Locking is required for ex if im working on terraform and another person also working on same terraform files so it will create discrepancies so locking is required

We have one file generated called terraform.tfstate.backup – this is previous state backup not the current state

Multiple environments in terraform:

**Dealing with different env: -**

1. Tfvars:

Create different S3 buckets and dynamodb locking for Dev and Prod differently and keep them in their directories

There should be different key for separate terraform project

We can create different variable for dev and prod like dev.tfvars

In Jenkins pipeline we can select in drop down weather we deploy dev or prod

1. Workspasec
2. Different repos for different env (Dev, Prod)

When we plan and apply we should give variable keys as terraform plan –var-file=dev/dev.tfvars

Terraform workspace show – This command will show the workspace which we are in

Terraform workspace select dev – This command will select and open the dev env

Terraform workspace list – This will list all workspace list

Terraform workspace new prod – This will create new workspace with prod

Terraform init -reconfigure – This will use once we are in one env (dev) and want to plan and deploy other env (prod) we need to use this command

terraform init -backend-config=dev/backend.tf – when we created new workspace with dev name and need to init we need to use this command

terraform plan -var-file=dev/dev.tfvars – This is used after the backend-config is done for dev workspace

terraform apply -var-file=dev/dev.tfvars – This is used to apply to create infra with dev

terraform destroy -var-file=prod/prod.tfvars – This is used to destroy infra from dev workspace

terraform init -reconfigure -backend-config=dev/backend.tf – If we are switching from prod workspace to dev workspace we need to use this command

**Provisioners : -**

Localexec and remoteexec

Local exec will run where you are running terraform command

Remoteexec: first you need to connect to server and remote exec will in the server

Infra creation is taken care by terraform and configuration management is taken care by ansible, we need to integrate terraform and ansible with provisioners

**Terraform modules : -**

Reuse

DRY- Don’t repeat yourself

No need to write infra for every project, we can reuse with module

Code reuse , no need to write code for

We can give variables as default input as empty so that when ever who is using modules users can override

If the user want to override variables from module user can give variable with variable definition in the main.tf of ec2.tf in user execution

Always provider will not be in module it will be user execution

We can even create variable.tf in user execution

If variable default is commented In module then user should give

If variable in module is default = {} then optional

We can merge(var.commom\_tags, var.resource\_tags)

Data types : - when we want to ask aws to provide details we will use data types to query the resources as below

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

    #all\_availability\_zones = true

    state = "available"

}

In the above with data we are querying awz availability zones

Local.tf : - in local we can keep the common code and we can call it as below

locals {

  name = "${var.project\_name}-${var.environment}"

  az\_names = slice(data.aws\_availability\_zones.azs.names,0,2)

}

Local.name

Local.az\_names

Merge : -

We can merge any two tags as below

 tags = merge(

        var.common\_tags,

        var.vpc\_tags,

        {

            Name = local.name

        }

    )

Slice : -

With slice we can add the data fetched from aws and we can select the first two outputs provided by aws with 0,2 as in below command

Count: whenever we use count in any resource block it will be as list if one or two or three elements in it

az\_names = slice(data.aws\_availability\_zones.azs.names,0,2)

VPC: virtual private cloud

Subnet, which have access and route to internet gateway is called public subnet

Subnet which don’t have access and route to internet gateway is called private subnet

We can create route table and configure routes

NAT gateway is purely a outgoing traffic not an incoming traffic

We can create in public subnet 1a, add in private, and database route tables

VPC Peering:

To connect vpc in same region, same account different account same account different region etc

Transit gateway is used to connect on premises and Aws