Terraform

1) What is Terraform?

→ It is an Open Source IAC tool developed by hashicorp which is used to Plan, Apply and Destroy infrastructure using HCL (Hashicorp Configuration Language).

2) What is Terraform Module?

→ A module is a collection of .tf and/or .tf.json files kept together in a directory. (Terraform configuration files in a single directory).
Modules are the main way to package and reuse resource configurations with Terraform.

3) What is Terraform Dependency Lock File?

→ terraform.lock.hcl, and this name is intended to signify that it is a lock file for various items that Terraform caches in the .terraform subdirectory of your working directory. Terraform automatically creates or updates the dependency lock file each time you run the terraform init command.

4) What is Resources in Terraform?

→Resources in terraform are it provides all the resources in aws Provider. Each resource block describes one or more infrastructure objects, such as virtual networks, compute instances, or higher-level components such as DNS records.
Example:-

```
resource "aws_instance" "web" {
  ami           = "ami-a1b2c3d4"
  instance_type = "t2.micro"
}
```

5) What is a provider in Terraform?

→ Providers is a terraform plugins that allows users to manage an external API. For example, if we plan on using Terraform to provision infrastructure on AWS, we will need to declare an AWS provider in our configuration files.

How can you use the same provider in Terraform with different configurations?

Ans:-By using alias argument in the provider block.

6) What is Alias in terraform?

→ Alias is a way of defining multiple configurations for the same provider and select which one to use.

7) What is Variables and Outputs?

→ The Terraform language includes a few kinds of blocks for requesting or publishing named values.

- Input Variables serve as parameters for a Terraform module, so users can customize behaviour without editing the source. Input variables are like function arguments.
- Output Values are like return values for a Terraform module. Output values are like function return values.
- Local Values are a convenience feature for assigning a short name to an expression. Local values are like a function's temporary local variables.

```
variable "example" {
  type     = string
  nullable = false
}
resource "aws_instance" "example" {
  instance_type = "t2.micro"
  ami           = var.image_id
}
```

8) What is Terraform Statefile?

→ Terraform state file maintain the state of infrastructure and always try to match the state with cloud infrastructure.

- Once statefile is deleted it is very difficult to recreate.
- Terrform statefile will be created in the root module.
- These files generally stored in S3.
- Where it keeps all the information about the infrastructure that was created by terraform. Whether it is a name,region,instance type keeps all information all the things that are created by terraform using those configuration. It is in json format.

9) What is State locking in terraform?

→ If supported by your backend, Terraform will lock your state for all operations that could write state. This prevents others from acquiring the lock and potentially corrupting your state. State locking happens automatically on all operations that could write state. You won't see any message that it is happening.

Ya usually we stored in .tfstate in S3 bucket by using dynamodb ()

10) What is Terraform Force unlock?

→ *terraform force-unlock [options] LOCK_ID [DIR]*

Manually unlock the state for the defined configuration.

This will not modify your infrastructure. This command removes the lock on the state for the current configuration. The behaviour of this lock is dependent on the backend being used. Local state files cannot be unlocked by another process.

(Force unlock should only be used to unlock your own lock in the situation where automatic unlocking failed. To protect you, the force-unlock command requires a unique lock ID.)

11) What is Terraform Provisioners?

→ Provisioners are Terraform resources used to execute scripts as a part of the resource creation or destruction.

The custom actions can vary in nature and it can be –

1. Running custom shell script on the local machine
2. Running custom shell script on the remote machine
3. Copy file to the remote machine

Also there are two types of provisioners -

1. Generic Provisioners (file, local-exec, and remote-exec)
2. Vendor Provisioners (chef, habitat, puppet, salt-masterless)

Generic Provisioners - Generally vendor independent and can be used with any cloud vendor (GCP, AWS, AZURE)

Vendor Provisioners - It can only be used only with its vendor. For example, chef provisioner can only be used with chef for automating and provisioning the server configuration.

i) **File:-** *file* provisioner can be used for transferring and copying the files from one machine to another machine.

```
provisioner "file" {
    source      = "/home/rahul/Jhooq/keys/aws/test-file.txt"
    destination = "/home/ubuntu/test-file.txt"
}
```

ii) **Local-exec:-** Used to run a script or command on a local resource after it gets created.

```
provisioner "local-exec" {
    command = "touch hello-jhooq.txt"
}
```

iii) **Remote-exec:-** Used to Run a Script or Command on Remote resource after it gets created.

```
provisioner "remote-exec" {
    inline = [
      "touch hello.txt",
      "echo helloworld remote provisioner >> hello.txt",
    ]
}
```

12) What is a null resource in Terraform?

A terraform null resource is a configuration that runs like a standard terraform resource block but does not create any resources. This may sound like a strange and useless resource, but it can be useful in various situations to work around limitations in Terraform.

```
resource "null_resource" "cluster" {
  # Changes to any instance of the cluster requires re-provisioning
  triggers = {
    cluster_instance_ids = "${join(",", aws_instance.cluster.*.id)}"
  }
```

13) What is Terraform Console?

→ The Terraform console command does not modify your state, configuration files, or resources. It provides a safe way to interactively inspect your existing project's state and evaluate Terraform expressions before incorporating them into your configuration.

If we want to check path of terraform state file it will display on the terminal when we hit path module.

If we can run mathematical operation on fulfil console command.

terraform console

>

<aws_s3_bucket.data>

It will display all data in a file.

14) What is Terraform Workspace?

→ *Terraform workspace used to logically segregate your .tfstate file.*

Workspaces allow the use of the same configuration but having separate state backends for each workspace. One use case is having production and dev workspaces. This allows the same setup for both environments but each with their own state. When working in the dev workspace you can ensure that you are only modifying that infrastructure and not production. This is great for testing changes without affecting other environments.

- Use the CLI to create, update, switch, list, show and delete workspaces
- Use dev.tfvars and prod.tfvars files to set different variables depending on the workspace
- Examine the different state between workspaces
- Access the current workspace in config files with terraform.workspace

- Simply independently managed statefiles
- A workspace contains everything that terraform needs to manage a given collection of infrastructure.

Workspaces are separate instances of state data that can be used from the same working directory.

You can use workspaces to manage multiple non-overlapping groups of resources with the same configuration. Every initialized working directory has at least one workspace.

15) What is Terraform Refresh Command?

→ The terraform refresh command **reads the current settings from all managed remote objects and updates the Terraform state to match**.

Warning: This command is deprecated, because its default behavior is unsafe if you have misconfigured credentials for any of your providers.

16) What is Lifecycle in Terraform?

→ Lifecycle is a nested block that can appear within a resource block. The lifecycle block and its contents are meta-arguments, available for all resource blocks regardless of type.

The arguments available within a lifecycle block are create_before_destroy, prevent_destroy, ignore_changes, and replace_triggered_by.

```
resource "aws_instance" "example" {
  # ...

  lifecycle {
    ignore_changes = [
      # Ignore changes to tags, e.g. because a management agent
      # updates these based on some ruleset managed elsewhere.
      tags,
    ]
  }
}
```

1) Simple Terrform.tf (Main.tf)

```
Provider “aws” {
    Region= “Us-east1”
    Access_key= “aKey”
    Secret_Key= “Skey”
}
resource "aws_instance" "web" {
          ami = "ami-032930428bf1abbff"
          instance_type = "t3a.large"
        security_groups = [“lanch_wizard”]
        tags={
           Name= “Terraform Server”
          }
}
resource "aws_s3_bucket" "Buckets" {
  bucket = "my-tf-test-bucket-fhgfgg"
  acl    = "private"
  tags = {
    Name = "My bucket"
    Environment = "Dev"
```

```
  }
}
```

2) Variables.tf

```
variable “ami” {
default= “ami-dudhus”
description= “Type of ami to use”
type=t3a.medium
sensitive=true
}
Variable “Instance_type” {
default= “t3a.xlarge”
description= “Size of ec2”
type =string
sensitive=false
}

variable "aws_region" {
  description = "Region for the VPC"
  default     = "us-west-2"
}

variable "vpc_cidr" {
  description = "CIDR for the VPC"
  default     = "10.0.0.0/16"
}

variable "public_subnet_cidr" {
  description = "CIDR for the public subnet"
  default     = "10.0.1.0/24"
}

variable "private_subnet_cidr" {
  description = "CIDR for the private subnet"
  default     = "10.0.2.0/24"
}

variable "ami" {
```

```
  description = "Amazon Linux AMI"
  default     = "ami-28e07e50"
}

variable "key_path" {
  description = "SSH Public Key path"
  default     = "public_key"
}
```