

Terraform

Interview Questions



June 7, 2024

Hexaware

[Company address]

1. What is Terraform?

Terraform is open-source communication as a system software tool created by HashiCorp. It is an instrument for building, altering, and versioning transportation safely and professionally. Terraform can direct existing and accepted service providers as well as convention in-house solutions.

1. How can you check the version of Terraform?

In the terminal run command "terraform version"

1. What are Terraform Providers?

These are plugins that Terraform uses to create and manage resources on a specific infrastructure.

A provider usually provides resources needed to manage a single cloud or infrastructure platform, such as AWS or Azure, or technology (for example Docker or database).

1. Where can you find information about the available providers?

Terraform Registry is the main source of Terraform providers and major infrastructure platforms.

1. How do you configure provider in Terraform?

A provider configuration is created using a provider block in the file “providers.tf”

***terraform {***

***required\_providers {***

***aws = {***

***source = "hashicorp/aws"***

***version = "~> 3.0"***

***}***

***}***

***}***

***# Configure the AWS Provider***

***provider "aws" {***

***region = "us-central-1"***

***}***

1. How do you specify the location of the provider to download?

***terraform {***

***required\_providers {***

***aws = {***

***source = "<HOSTNAME>/<NAMESPACE>/<TYPE>"***

***version = "~> 3.0"***

***}***

***}***

***}***

HOSTNAME = registry.terraform.io

NAMESPACE = hashicorp

TYPE = google/aws/azure

1. What are the most important components of the Terraform Configuration Syntax?

Terraform configuration syntax is built around 2 key syntax constructs: blocks and arguments.

A block is a container for other content, while arguments are simply used to assign an expression to a name. Each block has a type and one or more labels. There are block types such as provider that require one label. Whereas blocks such as resource require two labels.

1. How do you add a comment?

Single-line comment begins with a hash (#). Multi-line comments, aka block comments, which start with "/" and end with "\*/"

1. How can you download and install the required Terraform providers?

Providers are distributed separately from Terraform itself and to use them. The command "terraform init" will download and install provider locally.

1. What command allows you to preview the changes?

Terraform plan is the command used to preview the changes Terraform will make.

1. What "terraform fmt" command does?

The “terraform fmt” command formats, validates, and rewrites our terraform configuration files so they are readable and consistent.

1. What command updates the real infrastructure?

The command “terraform apply” updates the real infrastructure according to the declarations.

1. What are Terraform data sources?

They are a kind of API, and their goal is to do something and give us back dynamic data. So, they are used to fetch dynamic data from cloud providers. Data sources are like queries. A list of AMIs that changes frequently, or a list of Availability Zones are some examples of data sources.

1. What command is used to see all resource types and local names in the Terraform state file?

The required command is “terraform state list”.

1. How can you run specific commands on EC2 instances using Terraform?

We can run commands or scripts by giving them as value to an attribute called “user\_data”. We can use cloud-init which is the industry standard for cloud instance initialization.

1. What variables can be used to enable Terraform logging?

We can use TF\_LOG to enable Terraform logging at the terminal or we can use TF\_LOG\_PATH to enable logging to a file.

1. What are the data types?

The data types are number, string, bool, and a special type which is null.

1. What are the complex types available in HCL?

There are two categories of complex types:

Collection types for grouping values of the same type, and structural types for grouping several distinct types as a single value.

In Terraform, there are three kinds of collection types: list, map and set and 2 kinds of structural types: object and tuple.

1. Provide examples for the list and map types in HCL?

***# type list (of strings)***

***variable "azs" {***

***description = "AZs in the Region"***

***type = list(string)***

***default = [***

***"eu-central-1a",***

***"eu-central-1b",***

***"eu-central-1c"***

***]}***

***# type map***

***variable "amis" {***

***type = map(string)***

***default = {***

***"eu-central-1" = "ami-0dcc0ebde7b2e00db",***

***"us-west-1" = "ami-04a50faf2a2ec1901" }***

1. How can you declare variables of type tuple and object in HCL?

# type tuple

***variable "my\_instance" {***

***type = tuple([string, number, bool])***

***default = ["t2.micro", 1, true ]***

***}***

***# type object***

***variable "egress\_dsg" {***

***type = object({***

***from\_port = number***

***to\_port = number***

***protocol = string***

***cidr\_blocks = list(string)***

***})***

***default = {***

***from\_port = 0,***

***to\_port = 65365,***

***protocol = "tcp",***

***cidr\_blocks = ["100.0.0.0/16", "200.0.0.0/16", "0.0.0.0/0"]***

***}}***

1. Explain the differences between count and for\_each?

Both count and for\_each are meta-arguments used to duplicate resources that are similar.

However, for\_each was introduced more recently to overcome the downsides of count, as count is sensitive to any changes in the list order while for\_each isn’t.

The for\_each meta-argument accepts a map or a set of strings and creates an instance for each item in the map or set.

Sets and maps do not allow duplicates and they are unordered so creating or destroying individual resources using for\_each leaves all the others in their proper place.

1. What are the main problems when using the local state?

Problem #1.

The local state is good for testing and development, or if we are working alone. But in production environments or when working in a team, the use of a local state file brings many complications.

Problem #2.

Concurrency is another problem. It's also important that nobody else runs Terraform at the same time. Otherwise, the current changes will not be seen, and the state file can get corrupted.

The solution to both problems is to store the state remotely.

1. Consider the following partial Terraform code. Where will the state be saved?

***terraform {***

***required\_providers {***

***aws = {***

***source = "hashicorp/aws"***

***version = "~> 3.0"***

***} }***

***backend "s3" {***

***}}***

?

The Terraform state will be saved remotely on Amazon S3

1. What is a Terraform module?

A Terraform module is a set of Terraform configuration files in a single directory. Even the simplest configuration consisting of a single directory with one .tf file is a module.

1. What types of modules are available? Describe each type.

There are two types of modules: Local and remote modules.

Local modules are loaded from the local filesystem and are generally created by us or other members of the team to organize and encapsulate our code.

Remote modules are loaded from a remote source such as Terraform Registry and are created and maintained by HashiCorp and its partners or by third parties.

1. What are the reasons for choosing Terraform for DevOps?

It can do complete orchestration and not just configuration management (like Ansible and Puppet).

Has amazing support of almost all the popular cloud providers like AWS, Azure, GCP, DigitalOcean etc.

Easily manages the configuration of an immutable (dynamic) infrastructure.

Provide immutable infrastructure where configuration changes smoothly.

Works on HCL (HashiCorp configuration language), which is very easy to learn and understand.

Easily portable from one provider to another.

Easy Installation.

1. Define terraform configuration and state?

Terraform Configuration – It keeps track of the infrastructure detail.

Terraform state – It keeps track of the infrastructure status.

1. What are the key features of Terraform?

Infrastructure as Code: Terraform’ s high-level configuration language is used to define your infrastructure in human-readable declarative configuration files.

You may now create an editable, shareable, and reusable blueprint.

Terraform generates an execution plan that specifies what it will do and asks for your approval before making any infrastructure alterations. You can assess the modifications before Terraform creates, updates, or destroys infrastructure.

Terraform creates a resource graph while simultaneously developing or altering non-dependent resources. Terraform can now build resources as quickly as possible while also giving you more information about your infrastructure.

Terraform’ s the automation of change allows you to apply complex changesets to your infrastructure with little to no human interaction. Terraform recognizes.

1. Define null resource in Terraform?

The null resource follows the standard resource lifecycle but takes no additional actions. The trigger argument allows for the specification of a subjective set of values that, if misrepresented, will cause the reserve to be replaced.

The null resource’s primary application is as a do-nothing container for arbitrary actions performed by a provisioner.

1. What do you mean by Terragrunt, list some of its use cases?

Answer: Terragrunt is a lightweight wrapper that adds tools for maintaining DRY configurations, working with multiple Terraform modules, and managing remote states.

Use cases:

* Keep your Terraform code DRY.
* Maintain a DRY remote state configuration.
* Keep your CLI flags DRY.
* Run Terraform commands on multiple modules at the same time.
* Use multiple AWS accounts.

1. What is State File Locking?

State file locking is a Terraform mechanism that prevents operations on a specific state file from being performed by multiple users at the same time. Once the lock from one user is released, any other user who has taken a lock on that state file can operate on it. This aids in the prevention of state file corruption. The acquiring of a lock on a state file in the backend is a backend operation. If acquiring a lock on the state file takes longer than expected, you will receive a status message as an output.

1. What is a Remote Backend in Terraform?

Terraform remote backend is used to store Terraform’s state and can also run operations in Terraform Cloud. Multiple terraform commands such as init, plan, apply, destroy (terraform version >= v0.11.12), get, output, providers, state (sub-commands: list, mv, pull, push, rm, show), taint, un-taint, validate, and many more are available via remote backend. It is compatible with a single remote Terraform cloud workspace or multiple workspaces. You can use terraform cloud’s run environment to run remote operations such as terraform plan or terraform apply.

1. What is a Tainted Resource?

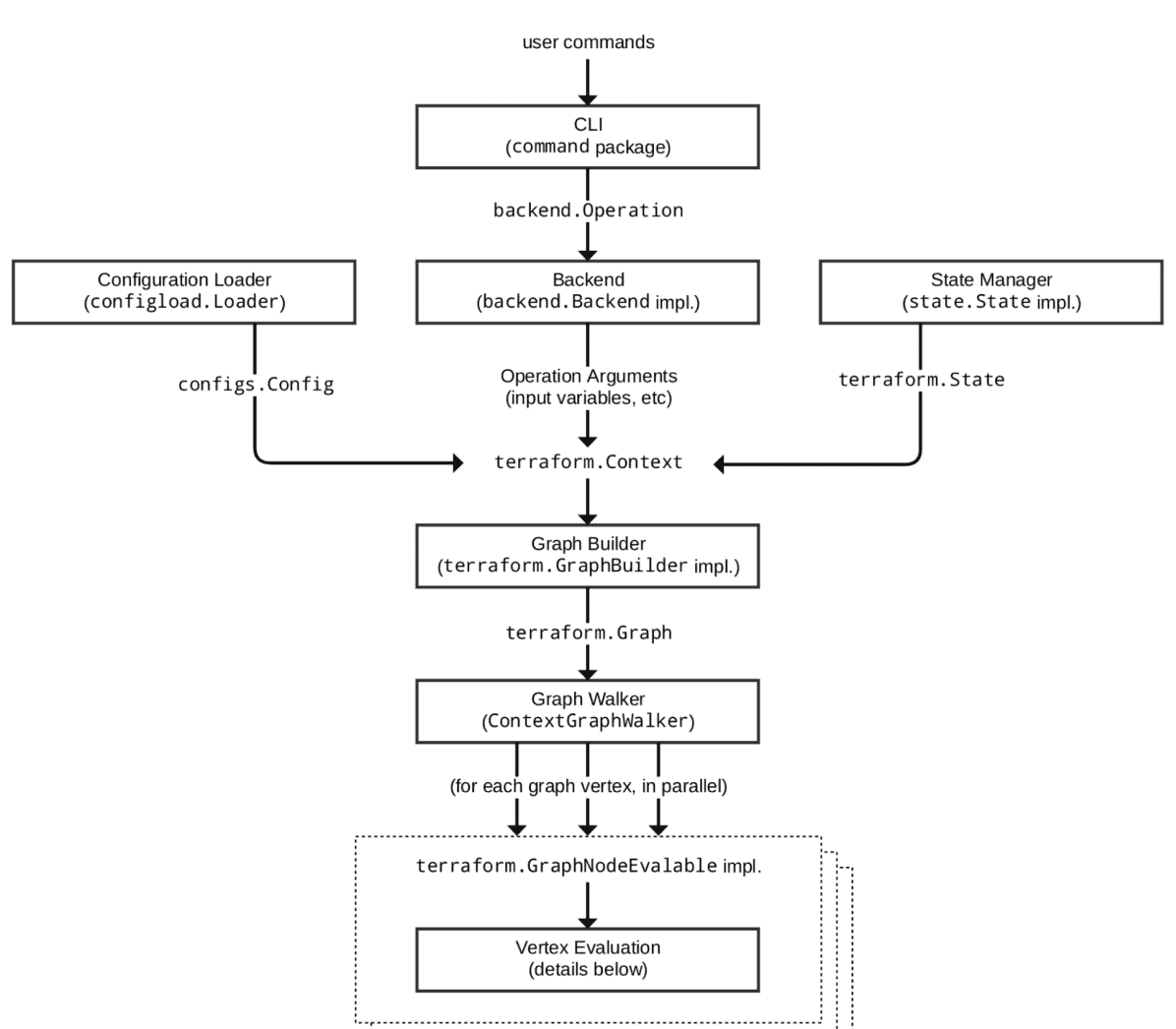
Tainted resources are those that must be destroyed and recreated upon the next applied command. Nothing changes on infrastructure when you mark a resource as tainted, but the state file is updated with this information (destroy and create). After marking a resource as tainted, terraform plan out will show that the resource will be destroyed and recreated, and the changes will be implemented when the next application occurs.

1. How to prevent Error Duplicate Resource?

It can be done in three ways depending on the situation and the requirement.

* By deleting the resource, terraform code will no longer manage it.
* By removing resources from APIs
* Importing action will also aid in resource elimination.

1. Explain the architecture of Terraform request flow?



1. How can you handle state conflicts with Terraform?

Terraform’s state is a crucial component for tracking and managing resources. Conflicts usually occur when multiple team members concurrently attempt modifications. To manage such situations, Terraform offers “state locking” for backends that support it. State locking prevents others from acquiring the lock and potentially corrupting the state.

1. How would you manage sensitive data with Terraform?

Answer: For managing sensitive data, Terraform has the concept of ‘variables’. These can be marked as sensitive, which stops the values from being displayed in the CLI output. We can also use Terraform’s data encryption features, or external vault services for storing secrets.

1. Can you explain how you would handle lifecycle hooks in Terraform?

Answer: Lifecycle hooks in Terraform are controlled through the lifecycle configuration block within a resource block. This can control actions such as preventing a resource from being destroyed (prevent\_destroy), and deciding when to create a new resource before destroying the old one (create\_before\_destroy).

1. Explain how you would import existing resources into Terraform?

Answer: Existing resources can be imported into Terraform using the terraform import command followed by the resource type and ID. This command maps an existing resource to a Terraform resource in the configuration file.

1. How would you ensure idempotency in Terraform?

Answer: Terraform naturally ensures idempotency. When a terraform apply command is executed, it compares the desired state defined in the configuration files with the current state, and then it only modifies what’s necessary to reach the desired state.

1. How would you manage multiple environment configurations (like staging, production) using Terraform?

Answer: You can manage different environments by separating them into different directories or workspaces, each with its own state file. This ensures the isolation between the environments.

1. What is Terraform’s way of handling infrastructure changes?

Answer: Terraform uses a declarative approach where the infrastructure needed is defined in a configuration file and Terraform takes care of how to achieve that. For changes, it compares the current infrastructure state with the declared state and makes necessary adjustments.

1. How would you update a single module in Terraform?

Answer: You can use the command terraform get -update to fetch and update modules.

1. Explain how to use conditionals in Terraform?

Answer: Conditionals in Terraform can be used using the count parameter, or the for\_each and if expressions. They allow you to create or ignore resources based on certain conditions.

1. What would you do if Terraform state file gets deleted?

Answer: You can use terraform import to import the real infrastructure into the state file again, or restore from backup if available. This is one of the reasons why remote state files are recommended, which inherently provide versioning and backup.

1. How would you share output values between different Terraform configurations?

Answer: You can use Terraform’s output values to extract information from a configuration and share it with another. You declare an output variable in one configuration and reference it in another using terraform\_remote\_state data source.

1. How would you define and manage dependencies in Terraform?

Answer: In Terraform, dependencies can be explicit (by using the depends\_on argument) or implicit (derived from the relationships of the resources). Terraform builds a dependency graph and ensures resources are created, updated, or destroyed in a proper sequence.

1. How would you prevent resources from being accidentally destroyed in Terraform?

Answer: To prevent resources from accidental destruction, you can use the prevent\_destroy lifecycle argument. When this is set to true, Terraform will refuse to destroy the resource.

1. How can you refactor a monolithic Terraform configuration into a modular one?

Answer: Start by identifying logical components and dependencies of your infrastructure. Each of these can be a separate module. Move the related resources into these module directories and replace them in the main configuration with module calls. Test each module independently to ensure they work as expected.

1. How would you rollback a change made by Terraform?

Rollbacks aren’t inherently supported by Terraform as it’s designed to move forward to a known state. However, you can achieve a similar effect by using version control for your configuration files. You can revert to a previous commit and then run terraform apply to set your infrastructure to that state.

1. Define IAC?

IaC is a short form to the term “Infrastructure as Code”. IaC refers to a scheme whereby developers can run and provision the computer data center’s mechanically instead of getting into a physical process. Terraform, for example, is a case tool of IaC.

1. Describe the working of Terraform core?

The terraform core looks at the configuration monitoring and creates analysis and evaluation based on the configuration. It keeps track and compare the versions (current and previous) and then display the output through the terminal. Terraform core mainly takes two inputs: Terraform Configuration – It keeps track of the infrastructure detail Terraform state – It keeps track of the infrastructure status.