### #1. What do you understand by Terraform?

Terraform is an open-source IAC tool created by [HashiCorp](https://www.hashicorp.com/" \t "_blank). It is used to create, update, delete and version your infrastructure on multiple cloud platforms.

### #2. What are the reasons to choose Terraform for DevOps?

Using Terraform for provisioning infrastructure leaves no room for human errors, hence improving the quality, consistency, and efficiency of Cloud and on-prem infrastructure. Terraform uses the HCL language, which is fairly similar to JSON and easy to learn and use. Unlike the other IAC tools offered by cloud providers like [Cloudformation](https://geekflare.com/cloudformation-vs-terraform/" \t "_blank) for AWS, we can use Terraform with a number of cloud platforms simultaneously. This avoids the need to learn multiple IAC tools and improves the scope of collaboration.

### #3. How does Terraform work?

Terraform uses plugins called the Terraform providers to interact with APIs on Cloud Platforms and provision our resources. As an end-user, terraform workflow has three steps.

**Write**: Author the infrastructure as code.

**Plan**: Preview changes Terraform will make before applying.

**Apply**: Provision the infrastructure and apply the changes.

### #4. What do you mean by Terraform cloud?

Terraform Cloud is a remote environment that is optimized for the Terraform workflow. It provides features like workspaces and state locking, which allows people in big teams to collaborate.

### #5. What do you understand by State in Terraform?

As an IAC tool, terraform should know the current state of configurations and infrastructure under its management. Terraform stores this information in a file called the state file.

### #6. What is the benefit of Terraform State?

The Terraform State allows Terraform to map real-world resources to your configuration, keep track of metadata, and improve performance when planning changes for complex infrastructures. It is a critical component of Terraform.

### #7. What do you understand by Terraform Backend?

Terraform backend is the platform where the Terraform State Snapshots are stored. By default, Terraform uses a backend called local to store state as a local file on your disk. All other supported backends are some kind of remote storage service.

### #8. What is a provider in Terraform?

Providers in Terraform are plugins that allow Terraform to interact with cloud providers, SaaS providers, and other APIs. 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.

### #9. Who maintains Terraform Providers?

Providers are distributed separately from Terraform itself. As a Terraform user, anyone can develop their own providers. There are some standard providers that are maintained explicitly by Hashicorp.

### #10. What is Sentinel?

[Sentinel](https://www.terraform.io/cloud-docs/sentinel) is a policy as a code tool used to enforce standard configurations for resources being deployed by Terraform. It can be used by organizations for compliance and governance purposes.

### #11. What do you understand by modules in Terraform?

A Terraform module is a standard container for multiple resources used together to provision and configure resources. For example, you can create a “VPC module” for your organization that provisions a standard VPC and other resources like Subnets and Internet Gateways. Modules can be shared publically via the Public module registry and privately via the Private Module registry.

### #12. What is the benefit of using modules in terraform?

Terraform modules allow us to create logical abstraction on the top of a resource set. Using modules allows us to maintain and reuse a standard configuration for resources. They can be versioned and shared with members of your teams to provision resources in a standard way.

### #13. What is the Private Module Registry?

A Private Module Registry Terraform Cloud feature allows us to share Terraform modules across our organization.

## Advanced Terraform Interview Questions and Answers

### #14. How can we export data from one module to another?

We can export data from a module by defining output blocks in the module configuration files. This data can then be transferred as a parameter to the destination module.

### #13. How can you define dependencies in Terraform?

Terraform has built-in dependency management. Terraform has two kinds of dependencies between resources- implicit and explicit dependencies.

Implicit dependencies, as the name suggests, are detected by Terraform automatically. This is when the output of a “resource A” is used in “resource B”. Terraform automatically detects that “resource B” needs to be created only after “resource A”

Explicit dependencies can be specified in cases where two resources are internally dependent on each other without sharing any outputs. This can be done by using the depends\_on parameter in the configuration block.

### #14. What are Provisioners in Terraform?

Provisioners are Terraform resources used to execute scripts as a part of the resource creation or destruction. There are two types of Provisioners in Terraform:

* **local-exec:**Invokes a script on the machine running Terraform.
* **remote- exec:**Invokes a script on a remote resource after it is created.

Provisioners are only meant to be used as a last resort in Terraform.

### #15. What is the external data block in Terraform?

Just like the local-exec provisioner, external data bock can be used to run scripts on machines running Terraform. The difference between a provisioner and the external data block is that the scripts in the external data block can return data in JSON format, whereas provisioners cannot return any outputs. It is important to note that external data blocks are also meant to be a last resort and should not be used if there is a better alternative.

### #16. How can two people using the Terraform cloud can create two different sets of infrastructure using the same working directory?

By using different workspaces. These users can start Terraform runs in two separate workspaces. Each workspace has a state file of its own, so as long as the resources do not overlap, both the users can successfully provision two different sets of infrastructure using the same code.

### #17. What happens when multiple engineers start deploying infrastructure using the same state file?

Terraform has a very important feature called **“state locking”**. This feature ensures that no changes are made to the state file during a run and prevents the state file from getting corrupt. It is important to note that not all Terraform Backends support the state locking feature. You should choose the right backend if this feature is a requirement.

### #18. 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.**

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

By using alias argument in the provider block.

### #20. You have a Terraform configuration file with no resources. What happens when you run the terraform apply command?

Terraform will destroy all the resources. Starting an empty run with terraform apply command is exactly the same as starting the terraform destroy run.

### #21. What happens if a resource was created successfully in terraform but failed during provisioning?

This is an unlikely scenario, but when this happens, the resource is marked as tainted and can be recreated by restarting the terraform run.

### #22. Which value of the TF\_LOG variable provides the MOST verbose logging?

TRACE is the most verbose and the default value of the TF\_LOG variable.

### #23. How can you import existing resources under Terraform Management?

By using the terraform import command.

### #24. Which command can be used to preview the terraform execution plan?

The terraform plan command generates the execution plan of the changes Terraform will do to the infrastructure.

### #25. Which command can be used to reconcile the Terraform state with the actual real-world infrastructure?

The terraform apply -refresh-only command is used to reconcile Terraform state with the actual real-world infrastructure. It is the new alternative to the terraform refresh command, which is now deprecated.

### #26. Which command can be used to switch between workspaces when using Terraform Cloud?

The terraform workspace select <workspace-name> command is used to choose a different workspace.

### #27. Which command is used to perform syntax validation on terraform configuration files?

The terraform validate command is used to verify whether a configuration is syntactically valid and internally consistent.

### #28. Which command is used to create new workspaces in the Terraform cloud?

The terraform workspace new <workspace-name> command is used to create a new workspace.

## **Q: What are the key features of Terraform? Ans:**

Below are the key features of Terraform:

1. **Infrastructure as Code:**A high-level configuration syntax is used to define infrastructure.
2. **Execution Plans:**Terraform generates an execution plan during the "planning" phase.
3. **Resource Graph**
4. **Change Automation**

## **Q: What are most useful terraform commands? Ans:**

Below are the key commands of Terraform:

terraform init # Initialize the current directory

terraform plan # Dry run to see what Terraform will do

terraform apply # Apply the Terraform code and build stuff

terraform destroy # Destroy what was built by Terraform

terraform refresh # Refresh the state file

terraform output # View Terraform outputs

terraform graph # Create a DOT-formatted graph

## **Q: What is Terraform Cloud Ans:**

***Terraform Cloud****is a hosted application that includes remote state management, API-driven runs, policy management, and other capabilities. Many users prefer a SaaS solution that is hosted in the cloud since they do not want to manage the infrastructure required to run it.*

## **Q: What is Terraform Cloud for Business Ans:**

***Terraform Cloud for Business****uses the same hosted environment as Terraform Cloud, but with additional features for larger teams. Single sign-on, audit logging, and the ability to Terraform on-premise resources from the cloud are just a few of the features available.*

## **Q: What is Terraform Enterprise Ans:**

***Terraform Enterprise****is the same tool, but it runs in your own data centre or cloud environment. Some users want more control over the Terraform Cloud application, or they want to execute it behind corporate firewalls in restricted networks.*

1) What is Terraform?

**Terraform** is an open-source infrastructure as code software tool created by HashiCorp. It enables users to define and provision a datacenter infrastructure using a high-level configuration language known as Hashicorp Configuration Language, or optionally JSON.

2) Enlist major features of Terraform?

The key features of Terraform are

* Infrastructure as Code (IAC)
* Execution Plans
* Change Automation
* Resource Graph

3) Why Terraform is preferred for DevOps?

There are a bunch of reasons for giving preference to **Terraform** to be used as one of the significant tools of **DevOps**. The main motive of DevOps is to bring efficiency and quality in software delivery. For this, some tools are required for making the delivery smoother, faster and efficient. Here, terraform comes to the limelight where it aids organizations in automating and also aids with infrastructure as code. Terraform is helpful with its availability of implementing every type of coding principle. The extraordinary feature of terraforming includes its quickness and the operations performed by it. These are some of the important reasons for which Terraform is gaining popularity and attention in the organizations.

4) What is terraform in aws?

**Terraform by HashiCorp**, an AWS Partner Network (APN) **Advanced Technology Partner** and member of the AWS DevOps Competency, is an "infrastructure as code" tool similar to **AWS** CloudFormation that allows you to create, update, and version your Amazon Web Services (AWS) infrastructure.

5) What does HCL stand for?

In **Terraform HCL** stands for **HashiCorp Configuration Language**. It is a configuration language built by HashiCorp that is used to build a structured configuration language that is both human and machine-friendly for use with command-line tools but specifically targeted towards DevOps tools, servers, etc.

**Download Free :**[Terraform Interview Questions PDF](https://www.interviewmocks.com/pdf/terraform-interview-questions.pdf)

6) Explain the architecture of terraform?

The architecture of terraform consists of following components:

* CLI (Command Line interface)
* Backends for executing operations,storing state, and storing workspace-defined variables
* Configuration Loader
* State Manager
* Graph Builder
* Graph Walk
* Vertex Evaluation
* Expression Evaluation
* Sub-graphs

7) List some major competitors of terraform?

**Ansible**, **Kubernetes**, **Packer**, **Cloud Foundry**, and **Pulumi** are the most popular alternatives and competitors to **Terraform**.

8) In which programming language Terraform is written?

**Terraform** is an open-source tool created by **HashiCorp** and written in the **Go programming language**.

9) What is use of Terraform CLI? Enlist few major command.

The **terraform CLI** is a well-behaved command-line application. It is used to run commands for terraforming.

Below are few common Terraform commands:

* **Apply:** builds or changes infrastructure.
* **Console:** Interactive console for Terraform interpolations.
* **destroy:** Destroy Terraform-managed infrastructure.
* **env:** Workspace management
* **fmt:** Rewrites config files to canonical format
* **get:** Download and install modules for the configuration
* **graph:** Build a visible graph of Terraform resources Import: existing infrastructure into Terraform
* **Init:** Initialize a Terraform working directory
* **output:** Read output from a state file plan: Generate and show an execution plan validate: Validates the Terraform files
* **version:** Prints the Terraform version
* **Workspace:** Workspace management

10) What is a provider in Terraform? Enlist some Terraform Providers.

**Terraform provider** is responsible for understanding API interactions and exposing resources. Providers generally are an IaaS (e.g. Alibaba Cloud, AWS, GCP, Microsoft Azure, OpenStack), PaaS (e.g. Heroku), or SaaS services (e.g. Terraform Cloud, DNSimple, Cloudflare).

**Some Terraform providers are:**

* **IaaS services:** Examples are Alibaba Cloud, AWS, GCP, Microsoft Azure, OpenStack.
* **PaaS services:** Examples are Heroku.
* **SaaS services:** examples are Terraform Cloud, Cloudflare, DNSimple.

**Take Free:**[Terraform MCQ & Quiz](https://www.interviewmocks.com/terraform-mcq/)

11) For what provisioners are used in Terraform?

**Provisioners in Terraform** can be used to prepare servers or other infrastructure objects for service. It models specific actions on the local machine as well as on remote machines.

12) Enlist some Built-in Provisioners available in Terraform?

**Below is the list of some Built-in provisioners in Terraform:**

* Chef Provisioner
* File Provisioner
* Habitat Provisioner
* Local-exec Provisioner
* Puppet Provisioner
* Remote-exec Provisioner
* Salt-masterless Provisioner

13) What are Modules in Terraform?

A**module in Terraform** is a container for several resources that are used together. The root module is compulsory for every terraform that includes resources mentioned in the .tf files.

14) What is Terraform cloud? Enlist few features provided by it?

**Terraform Cloud** is an application that let the teams use the Terraform together. It runs in a reliable environment and includes easy access to shared state and secret data. It provides access controls for adopting changes to infrastructure and a private registry for sharing terraform modules.

**Features of the Terraform cloud are:**

* Audit logging
* SAML single sign-on.

15) How to check installed version of Terraform?

To check the installed version of Terraform, use the below command:

**Syntax**

terraform -version

16) What do you mean by IAC in Terraform?

**IAC** is an abbreviation to the term **"Infrastructure as Code"**. IaC refers to a system whereby developers can manage and provision the computer data centers automatically instead of getting into a manual process. Terraform, for instance, is an example tool of IaC.

17) Can terraform be used for on-prem infrastructure?

Yes, Terraform can be used for on-prem infrastructure. There are many providers that are available. you can choose any one of them which suits you best. Many also build customer terraform providers for themselves, all needed is just an API.

18) What is Oracle Cloud Infrastructure?

**Oracle Cloud** is a cloud computing service offered by Oracle Corporation providing servers, storage, network, applications and services through a global network of Oracle Corporation managed data centers. The company allows these services to be provisioned on-demand over the Internet.

19) How to create dependency between modules in terraform?

In Terraform dependencies are normally created automatically by references, rather than explicitly using depends\_on. In terraform, there is no way to use depends\_on variables.

20) How do you deal with versioning in Terraform?

**Semantic Versioning** of Models should be used. Apply should only be done with Versioned modules/tags.

21) What is used of terraform apply command?

The **Terraform apply command** is used to apply the changes required to reach the desired state of the configuration, or the pre-determined set of actions generated by a terraform plan execution plan.

22) What is the different between Platform.sh and Terraform?

**Platform.sh** and **Terraform** are similar in that they both subscribe to the idea of "infrastructure as code".

**Platform.sh** is a second-generation Platform-as-a-Service built especially for continuous deployment. It allows you to host web applications on the cloud while making your development and testing workflows more productive whereas **Terraform** is an open-source infrastructure as code software tool that enables users to define and provision a data center infrastructure using a high-level configuration language known as Hashicorp Configuration Language, or optionally JSON

23) What is Software Defined Networking?

**Software-defined networking (SDN)** technology is an approach to network management that enables dynamic, programmatically efficient network configuration in order to improve network performance and monitoring making it more like cloud computing than traditional network management

24) Which command is used to destroy Terraform-managed infrastructure?

Terraform **destroy command** is used to destroy the Terraform-managed infrastructure.

**Syntax**

**terraform** **destroy** [options] [dir]

25) What is use of fmt command in Terraform?

The **terraform fmt command** is used to rewrite Terraform configuration files to a canonical format and style. This command applies a subset of the Terraform language style conventions, along with other minor adjustments for readability.

**Syntax**

**terraform** **fmt** [options] [DIR]

26) What does the providers command do?

The **terraform providers command** prints information about the providers used in the current configuration.

**Syntax**

terraform providers [config-path]

27) What is Terragrunt?

**Terragrunt** is a thin wrapper for Terraform that implements the practices advocated by the Terraform: Up and Running book. We've found Terragrunt helpful as it encourages versioned modules and reusability for different environments with some handy features, including recursive code execution in subdirectories.

28) What is terraform state?

**Terraform state** is a command that is used for advanced state management. The state is a necessary requirement for Terraform to function. Terraform must store state about your managed infrastructure and configuration. This state is used by Terraform to map real-world resources to your configuration, keep track of metadata, and to improve performance for large infrastructures. Syntax

**terraform** **state** [options] [args]

29) What is terraform init?

The **terraform init** is a command that is used to initialize a working directory containing Terraform configuration files. This is the first command that should be run after writing a new Terraform configuration or cloning an existing one from version control. It is safe to run this command multiple times.

**Syntax**

**terraform** **init** [options] [DIR]

30) What is terraform backend?

A **"backend"** in Terraform determines how the state is loaded and how an operation such as apply is executed. This abstraction enables non-local file state storage, remote execution, etc.

By default, Terraform uses the "local" backend, which is the normal behavior of Terraform you're used to

## Q1. Are callbacks possible with Terraform on Azure?

By using the Azure Event Hubs, callbacks are probable on Azure. Terraform’s Azure supplier provides effortless functionality to users. Microsoft Azure Cloud Shell provides an already installed Terraform occurrence.

## Q2. What is Terraform init?

Terraform init is a control to initialize an operational index that contains Terraform pattern files. This control can be looped multiple times. It is the first command that should be run after writing the new Terraform design.

## Q3. What is Terraform D?

Terraform D is a plugin used on most in-service systems and Windows. Terraform init by default searches next directories for plugins.

## Q4. What do you understand about Terraform in AWS?

Ans:

Terraform is an infrastructure as code tool that permits us to store our Amazon Web Services infrastructure and produce an update. It is comparable to AWS Cloud Formation.

## Q5. Is history the same as it is on the web while using TFS API to provide resources?

Ans:

Yes, the narration is similar to on the web because UI keeps API as the base. The whole thing that is on the UI is availed during other methods and the API

## Q6. Why is Terraform used for DevOps?

This is because Terraform manages infrastructure as code. Infrastructure as code is the foundation for DevOps practices such as continuous integration, version control, continuous deployment, and code review.

## Q7. Define null resource in Terraform.

Ans:

null\_resource implements standard resource library, but no further action is taken. The triggers argument allows an arbitrary set of values that will cause the replacement of resources when changed.

## Q8. Explain Oracle Cloud Infrastructure.

Oracle cloud offered by Oracle Corporation is a cloud computing service providing storage, servers, applications, services, and network through a global network of managed data centers by Oracle Corporation. These services are provisioned on-demand over the Internet by the company.

## Q9. What do you understand about Terraform backend?

backend in Terraform is used to determine how an operation is executed and how the state is loaded. Terraform uses the ‘local’ backend by default. The abstraction enables remote execution, non-local file state storage, etc.

## Q10. What are the version controls supported by Terraform besides GitHub?

The version controls supported GitLab EE, GitLab CE, and Bucket cloud.

## Q11. What are modules in Terraform?

A jug for numerous resources that are used jointly is known as a module in Terraform. The root module includes resources mentioned in the .tf files and is required for every Terraform.

## Q12. Is Terraform usable for an on-prem infrastructure

Yes, Terraform can be used for on-prem infrastructure. As there are a lot of obtainable providers, we can decide which suits us the best. All that we need is an API.

## Q13. Does Terraform support multi-provider deployments?

Yes, multi-provider deployments are supported by Terraform, which includes on-prem like Openstack, VMware, and we can manage SDN even using Terram too.

## Q14. How is duplicate resource error ignored during Terraform apply?

We can try the following options:

Delete those resources from the cloud provider(API) and recreate them using Terraform

Delete those resources from Terraform code to stop its management with it

Carry out a Terraform import of the resource and remove the code that is trying to recreate them

## Q15. Name all version controls supported by Terraform

The supported version controls are:

Azure DevOps Services

Azure DevOps Server

Bitbucket Server

Bitbucket Cloud

Gitlab EE and CE

Gitlab.com

GitHub Enterprise

GitHub.com (OAuth)

GitHub.com

## Q16. What are some of the built-in provisioners available in Terraform?

Here is the list of built-in provisioners in Terraform:

Salt-masterless Provisioner

Remote-exec Provisioner

Puppet Provisioner

Local-exec Provisioner

Habitat Provisioner

File Provisioner

Chef Provisioner

## Q17. Which command destroys Terraform managed infrastructure?

The given command is used for this purpose:

Terraform destroy [options] [dir]

## Q18. Tell us about some notable Terraform applications

The applications of Terraform are pretty broad due to its facility of extending its abilities for resource manipulation. Some of the unique applications are:

Software demos development

Resource schedulers

Multi-cloud deployment

Disposable environment creations

Multi-tier applications development

Self-service clusters

Setup of Heroku App

## Q19. What are the components of Terraform architecture?

The Terraform architecture includes the following features:

Sub-graphs

Expression Evaluation

Vertex Evaluation

Graph Walk

Graph Builder

State Manager

Configuration Loader

CLI (Command Line interface)

Backend

## Q20. Define Resource Graph in Terraform.

resource graph is a visual representation of the resources. It helps modify and create independent resources simultaneously. Terraform establishes a plan for the configuration of the graph to generate plans and refresh the state. It creates structure most efficiently and effectively to help us understand the drawbacks.

**Q21: What do you mean by 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.

**Q22: What are the reasons for choosing Terraform for DevOps?**

Below 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.

**Q23.Define Terraform init?**

Terraform initializes the code using the command terraform init. This command is used to initialize the working directory containing Terraform configuration files. It is safe to run this command multiple times.

You can use the init command for:

Plugin Installation

Child Module Installation

Backend Initialization

**Q24. Name some major competitors of Terraform?**

Some of them are:

Packer

Cloud Foundry

Ansible

Kubernetes

**Q25.Define Terraform provider?**

Terraform is used to manage and inform infrastructure resources such as bodily machines, VMs, network switches, containers, and more. A provider is accountable for thoughtful API interactions and revealing resources. Terraform supports a large number of cloud providers.

**Q26. How does Terraform work?**

Terraform creates an implementation plan, define what it will do to attain the preferred state, and then executes it to construct the described infrastructure. As the configuration changes, Terraform is talented to decide what changed and generate incremental execution plans which can be practical.

**Q27. Name some major features of Terraform?**

Some of them are:

Execution Plan

Change Automation

Resource Graph

Infrastructure as code

**Q28. 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.

**Q29.How to check the installed version of Terraform?**

We can use terraform -version of the command to identify the version which we are running.

**Q30.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.

**Q31. How does Terraform help in discovering plugins?**

The authority “Terraform init” helps Terraform interpret configuration files in the operational directory. Then, Terraform finds out the essential plugins and searches for installed plugins in diverse locations. In addition, Terraform also downloads extra plugins at times. Then, it decides the plugin versions to use and writes a security device file for ensuring that Terraform will employ the identical plugin versions.

**Q32.Can I add policies to the open-source or pro version of Terraform enterprise?**

You cannot insert policies to the open-source description of Terraform Enterprise. The equal also goes for the Enterprise Pro version. The finest version of Terraform Enterprise only could contact the lookout policie.

**Q33. Define Modules in Terraform?**

Answer: A module in Terraform is a jug for numerous resources that are used jointly. The root module is required for every Terraform that includes resources mentioned in the .tf files.

**Q34. What are the ways to lock Terraform module versions?**

Answer: You can use the terraform module registry as a source and provide the attribute as ‘version’ in the module in a terraform configuration file. If you are using the GitHub repository as a source, then you need to specify the branch, version and query string with ‘? ref’.

**Q35. What do you mean by Terraform cloud?**

Answer: Terraform Cloud is an application that helps teams use Terraform together. It manages Terraform runs in a consistent and reliable environment, and includes easy access to shared state and secret data, access controls for approving changes to infrastructure, a private registry for sharing Terraform modules, detailed policy controls for governing the contents of Terraform configurations, and more.

**Q36. Define null resource in Terraform?**

Answer: The null resource implements the average resource lifecycle but takes no extra action. The trigger argument permits specifying a subjective set of values that, when misrepresented will source the reserve to be replaced.

The primary use-case for the null resource is as a do-nothing container for arbitrary actions taken by a provisioner.

**Q37**. **Can Terraform be used for on-prem infrastructure?**

Answer: Yes, Terraform can be utilized for on-prem infrastructure. There are a lot of obtainable providers. You can decide any one of them which suits you most excellent. Many also build client Terraform providers for themselves; all wanted is just an API.

**Q38. What does the following command do?**

Answer:

Terraform -version – to check the installed version of terraform

Terraform fmt– it is used to rewrite configuration files in canonical styles and format

Terraform providers – it gives information of providers working in the current configuration.

**Q39. List all the Terraform supported versions**

Answer:

GitHub.com

GitLab.com

GitHub Enterprise

GitLab CE and EE

Bitbucket Cloud and Server

Azure DevOps Server and Services

**Q40. Explain the command terraform validate in the context of Terraform.**

Answer: The terraform validate command checks the configuration files in a directory, focusing purely on the configuration and omitting any external services like remote state, provider APIs, and so on. Validate examines a configuration to verify if it is syntactically correct and internally consistent, regardless of variables or current state. As a result, it’s best for general reusable module verification, such as confirming that attribute names and value types are valid. This command can be run automatically, for example, as a post-save check in a text editor or as a test step in a continuous integration system for a reusable module.

Syntax: terraform validate [options]

**Q41. Mention some of the version control tools supported by Terraform.**

Answer: Version control tools supported by Terraform are:

GitHub

GitLab CE

GitLab EE

Bucket Cloud

**Q42. What are the modules in Terraform?**

A Terraform module is a set of numerous resources used jointly. It is a single directory containing Terraform configuration files. A simple arrangement with a single directory having one or more files can be referred to as a module. The directory is called the root module when Terraform commands are run directly, and the files have the extension .tf. A module that is called by another configuration is called configuration's "child module."

**Q43. What are the main reasons for choosing Terraform for DevOps?**

Following are the main reasons for choosing Terraform for DevOps:

Following are the reasons for choosing Terraform for DevOps:

Terraform is preferred for DevOps because it can do complete orchestration and not just configuration management such as Ansible and Puppet.

It provides amazing support to almost all the popular cloud providers like AWS, Azure, GCP, DigitalOcean, etc.

It can easily manage the configuration of an immutable (dynamic) infrastructure.

It provides an immutable infrastructure where configuration changes smoothly.

It works on HCL (HashiCorp Configuration Language), which is very easy to learn and understand.

It is easily portable from one provider to another.

The installation of Terraform for DevOps is easy.

**Q44. What do you understand by Terraform init?**

Terraform init is a control used to initialize an operational index containing Terraform pattern files. This control is the first command that should be run after writing the new Terraform design and can be looped multiple times.

**Q45. What is the null resource in Terraform?**

The null\_resource is used to implement a standard resource library, but no further action is taken. The triggers argument allows an arbitrary set of values that will cause the replacement of resources when changed.

[**Q**](https://youtu.be/vOV74gevFgs)**46. What is Terraform D?**

Terraform D is a plugin used on most in-service systems and Windows. Terraform init by default searches next directories for plugins.

**Q47. What do you understand by Terraform backend?**

The Terraform backend is used to specify how an operation is executed and how the state is loaded. It uses the 'local' backend by default. The abstraction enables remote execution, non-local file state storage, etc.

**Q48. What is ".terraform" directory?**

The ".terraform" directory is a local cache where Terraform retains some files required for subsequent operations against this configuration. Its contents are not intended to be included in version control.

**Q49. What are some major competitors of Terraform?**

Following are some major competitors of Terraform:

Packer

Cloud Foundry

Ansible

Kubernetes

**Q50. What is the usage of Terraform init?**

Terraform init is a command used to initialize the Terraform code. Let's see the all usage of Terraform init command:

Terraform init command is used to initialize the working directory containing Terraform configuration files.

It is used for Plugin Installation.

It is also used for Child Module Installation.

It is used for Backend Initialization.

You can safely run this command multiple times.

-------------------------------------------------------------------------------------------------------------

1. What do you understand by Terraform in AWS?

Terraform is a part of the [AWS DevOps](https://www.simplilearn.com/aws-and-devops-working-together-article) Competency and also an AWS Partner Network (APN) advanced technology partner. It is similar to [AWS Cloud Formation](https://www.simplilearn.com/tutorials/aws-tutorial/aws-cloudformation) in the sense that it is also an “infrastructure as code” tool that allows you to create, update, and version your AWS infrastructure.

2. What are the key features of Terraform?

Terraform helps you manage all of your infrastructures as code and construct it as and when needed. Here are its key main features:

* A console that allows users to observe functions
* The ability to translate HCL code into JSON format
* A configuration language that supports interpolation
* A module count that keeps track of the number of modules applied to the infrastructure.

3. Define IAC?

[IAC or Infrastructure as Code](https://www.simplilearn.com/what-is-infrastructure-as-code-article) allows you to build, change, and manage your infrastructure through coding instead of manual processes. The configuration files are created according to your infrastructure specifications and these configurations can be edited and distributed securely within an organization.

4. What are the most useful Terraform commands?

Some of the most useful Terraform commands are:

* terraform init - initializes the current directory
* terraform refresh - refreshes the state file
* terraform output - views Terraform outputs
* terraform apply - applies the Terraform code and builds stuff
* terraform destroy - destroys what has been built by Terraform
* terraform graph - creates a DOT-formatted graph
* terraform plan - a dry run to see what Terraform will do

5. Are callbacks possible with Terraform on Azure?

By using the Azure Event Hubs, callbacks are probable on Azure. Terraform’s Azure supplier provides effortless functionality to users. [Microsoft Azure Cloud](https://www.simplilearn.com/azure-cloud-services-and-its-importance-article) Shell provides an already installed Terraform occurrence.

6. What is Terraform init?

Terraform init is a control to initialize an operational index that contains Terraform pattern files. This control can be looped multiple times. It is the first command that should be run after writing the new Terraform design.

7. What is Terraform D?

Terraform D is a plugin used on most in-service systems and Windows. Terraform init by default searches next directories for plugins.

8. Is history the same as it is on the web while using TFS API to provide resources?

Yes, the narration is similar to on the web because UI keeps API as the base. The whole thing that is on the UI is availed during other methods and the API.

9. Why is Terraform used for DevOps?

Terraform uses a JSON-like configuration language called the HashiCorp Configuration Language (HCL). HCL has a very simple syntax that makes it easy for[DevOps teams](https://www.simplilearn.com/core-components-devops-team-needs-to-be-successful-article) to define and enforce infrastructure configurations across multiple clouds and on-premises data centers.

--------------------------------------------------------------------------------------------------------------------------------

1. What do you understand by Terraform in AWS?

Terraform is a part of the [AWS DevOps](https://www.simplilearn.com/aws-and-devops-working-together-article) Competency and also an AWS Partner Network (APN) advanced technology partner. It is similar to [AWS Cloud Formation](https://www.simplilearn.com/tutorials/aws-tutorial/aws-cloudformation) in the sense that it is also an “infrastructure as code” tool that allows you to create, update, and version your AWS infrastructure.

### 2. What are the key features of Terraform?

Terraform helps you manage all of your infrastructures as code and construct it as and when needed. Here are its key main features:

* A console that allows users to observe functions
* The ability to translate HCL code into JSON format
* A configuration language that supports interpolation
* A module count that keeps track of the number of modules applied to the infrastructure.

### 3. Define IAC?

[IAC or Infrastructure as Code](https://www.simplilearn.com/what-is-infrastructure-as-code-article) allows you to build, change, and manage your infrastructure through coding instead of manual processes. The configuration files are created according to your infrastructure specifications and these configurations can be edited and distributed securely within an organization.

Interested in becoming a cloud architect? Join our [Cloud Architect Master’s Program](https://www.simplilearn.com/cloud-solutions-architect-masters-program-training?source=GhPreviewCTAText) and learn AWS, Microsoft Azure, and Google Cloud Platform from the ground up!

### 4. What are the most useful Terraform commands?

Some of the most useful Terraform commands are:

* **terraform init**- initializes the current directory
* **terraform refresh** - refreshes the state file
* **terraform output**- views Terraform outputs
* **terraform apply** - applies the Terraform code and builds stuff
* **terraform destroy**- destroys what has been built by Terraform
* **terraform graph** - creates a DOT-formatted graph
* **terraform plan**- a dry run to see what Terraform will do

### 5. Are callbacks possible with Terraform on Azure?

By using the Azure Event Hubs, callbacks are probable on Azure. Terraform’s Azure supplier provides effortless functionality to users. [Microsoft Azure Cloud](https://www.simplilearn.com/azure-cloud-services-and-its-importance-article) Shell provides an already installed Terraform occurrence.

### 6. What is Terraform init?

Terraform init is a control to initialize an operational index that contains Terraform pattern files. This control can be looped multiple times. It is the first command that should be run after writing the new Terraform design.

### 7. What is Terraform D?

Terraform D is a plugin used on most in-service systems and Windows. Terraform init by default searches next directories for plugins.

### 8. Is history the same as it is on the web while using TFS API to provide resources?

Yes, the narration is similar to on the web because UI keeps API as the base. The whole thing that is on the UI is availed during other methods and the API.

### 9. Why is Terraform used for DevOps?

Terraform uses a JSON-like configuration language called the HashiCorp Configuration Language (HCL). HCL has a very simple syntax that makes it easy for[DevOps teams](https://www.simplilearn.com/core-components-devops-team-needs-to-be-successful-article) to define and enforce infrastructure configurations across multiple clouds and on-premises data centers.

**10. Define null resource in Terraform.**

null\_resource implements standard resource library, but no further action is taken. The triggers argument allows an arbitrary set of values that will cause the replacement of resources when changed.

### 11. What do you mean by Terraform cloud?

Terraform Cloud is a platform that enables teams to use Terraform together, either on-demand or in response to various events. It is deeply integrated with Terraform's workflows and data, unlike a general-purpose continuous integration system. It includes easy access to shared state and secret data, detailed policy controls for updating infrastructure and governing the contents of Terraform, a private registry for sharing Terraform modules, and lots more.

### 12. Explain Oracle Cloud Infrastructure.

Oracle cloud offered by Oracle Corporation is a [cloud computing](https://www.simplilearn.com/tutorials/cloud-computing-tutorial/what-is-cloud-computing) service providing storage, servers, applications, services, and network through a global network of managed data centers by Oracle Corporation. These services are provisioned on-demand over the Internet by the company.

### 13. What do you understand by terraform backend?

Each Terraform configuration can specify a backend, which defines two main things:

* Where operations are performed
* Where the state is stored (Terraform keeps track of all the resources created in a state file)

### 14. What are the version controls supported by Terraform besides GitHub?

The version controls supported GitLab EE, GitLab CE, and Bucket cloud.

### 15. Name some major competitors of Terraform?

Some of the top [competitors and alternatives to Terraform](https://www.gartner.com/reviews/market/cloud-management-tooling/vendor/hashicorp/product/terraform-enterprise/alternatives) are Azure Management Tools, Morpheus, CloudHealth, Turbonomic, and CloudBolt.

16. Explain the uses of Terraform CLI and list some basic CLI commands?

The Terraform Command-Line Interface (CLI) is used to manage infrastructure and interact with Terraform state, configuration files, providers, etc.

Here are some basic CLI commands:

* terraform init - prepares your working directory for other commands
* terraform destroy - destroys the previously-created infrastructure
* terraform validate - check whether the configuration is valid
* terraform apply - creates or updates the infrastructure
* terraform plan - shows changes needed by the current configuration

17. What are modules in Terraform?

A jug for numerous resources that are used jointly is known as a module in Terraform. The root module includes resources mentioned in the .tf files and is required for every Terraform.

18. What is a Private Module Registry?

A Private Module Registry is a feature from Terraform Cloud that allows you to share Terraform modules across the organization. You can enforce rules or “sentinel policies” on the registry that specify how members of your organization can use the modules.

19. Is Terraform usable for an on-prem infrastructure?

Yes, Terraform can be used for on-prem infrastructure. As there are a lot of obtainable providers, we can decide which suits us the best. All that we need is an API.

20. Does Terraform support multi-provider deployments?

Yes, multi-provider deployments are supported by Terraform, which includes on-prem like Openstack, VMware, and we can manage SDN even using Terram too.

Also Read: [VMware vSphere Best Practices](https://www.simplilearn.com/vmware-vsphere-best-practices-rar339-article)

21. How is duplicate resource error ignored during terraform apply?

We can try the following options:

1. Delete those resources from the [cloud provider](https://www.simplilearn.com/selecting-the-right-cloud-provider-article)(API) and recreate them using Terraform
2. Delete those resources from Terraform code to stop its management with it
3. Carry out a terraform import of the resource and remove the code that is trying to recreate them

22. Name all version controls supported by Terraform

The supported version controls are:

* Azure DevOps Services
* Azure DevOps Server
* Bitbucket Server
* Bitbucket Cloud
* Gitlab EE and CE
* Gitlab.com
* GitHub Enterprise
* GitHub.com (OAuth)
* GitHub.com

23. What are some of the built-in provisioners available in Terraform?

Here is the list of built-in provisioners in Terraform:

* Salt-masterless Provisioner
* Remote-exec Provisioner
* Puppet Provisioner
* Local-exec Provisioner
* Habitat Provisioner
* File Provisioner
* Chef Provisioner

### 24. Which command destroys Terraform managed infrastructure?

The given command is used for this purpose:

terraform destroy [options] [dir]

### 25. Tell us about some notable Terraform applications.

The applications of Terraform are pretty broad due to its facility of extending its abilities for resource manipulation. Some of the unique applications are:

* Software demos development
* Resource schedulers
* Multi-cloud deployment
* Disposable environment creations
* Multi-tier applications development
* Self-service clusters
* Setup of Heroku App

### 26. What are the components of Terraform architecture?

The Terraform architecture includes the following features:

* Sub-graphs
* Expression Evaluation
* Vertex Evaluation
* Graph Walk
* Graph Builder
* State Manager
* Configuration Loader
* CLI (Command Line interface)
* Backend

### 27. Define Resource Graph in Terraform.

A resource graph is a visual representation of the resources. It helps modify and create independent resources simultaneously. Terraform establishes a plan for the configuration of the graph to generate plans and refresh the state. It creates structure most efficiently and effectively to help us understand the drawbacks.

### 28. Can you provide a few examples where we can use for Sentinel policies?

Sentinels are a powerful way to implement a variety of policies in Terraform. Here are a few examples:

* Enforce explicit ownership in resources
* Restrict roles the cloud provider can assume
* Review an audit trail for Terraform Cloud operations
* Forbid only certain resources, providers, or data sources
* Enforce mandatory tagging on resources
* Restrict how modules are used in the Private Module Registry

### 29. What are the various levels of Sentinel enforcement?

Sentinel has three enforcement levels - advisory, soft mandatory, and hard mandatory.

* **Advisory** - Logged but allowed to pass. An advisory is issued to the user when they trigger a plan that violates the policy.
* **Soft Mandatory**- The policy must pass unless an override is specified. Only administrators have the ability to override.
* **Hard Mandatory** - The policy must pass no matter what. This policy cannot be overridden unless it is removed. It is the default enforcement level in Terraform.

### 30. How to Store Sensitive Data in Terraform?

Terraform requires credentials to communicate with your cloud provider's API. But most of the time, these credentials are saved in plaintext on your desktop. [GitHub](https://www.simplilearn.com/tutorials/git-tutorial/what-is-github) is exposed to thousands of API and cryptographic keys every day. Hence, your API keys should never be stored in Terraform code directly.  You should use encrypted storage to store all your passwords, TLS certificates, SSH keys, and anything else that shouldn't be stored in plain text.

Next up, let us get started with the advanced terraform interview questions section!

## **Advanced Terraform Interview Questions**

### 31. What is Terragrunt, and what are its uses?

Terragrunt is a thin wrapper that provides extra tools to keep configurations DRY, manage remote state and work with multiple Terraform modules. It is used for:

* Working with multiple AWS accounts
* Executing Terraform commands on multiple modules
* Keeping our CLI flags DRY
* Keeping our remote state configuration DRY
* Keeping our Terraform code DRY

### 32. Explain State File Locking?

State file locking is Terraform mechanism in which operations on a specific state file are blocked to avoid conflicts between multiple users performing the same process. When one user releases the lock, then only the other one can operate on that state. This helps in preventing state file corruption. This is a backend operation.

### 33. What do you understand by a Tainted Resource?

A tainted resource is a resource that is forced to be destroyed and recreated on the next apply command. When a resource is marked as tainted, the state files are updated, but nothing changes on infrastructure. The terraform plan out shows that help will get destroyed and recreated. The changes get implemented when the next apply happens.

### 34. How to lock Terraform module versions?

A proven way of locking Terraform module version is using the Terraform module registry as a source. We can use the ‘version’ attribute in module of the Terraform configuration file. As the Github repository is being used as a source, we need to specify versions, branch, and query string with ‘?ref’.

### 35. What is Terraform Core? Tell us some primary responsibilities of it.

Terraform Core is a binary written statically compiled by using the Go programming language. The compiled binary offers an entry point for the users of Terraform. The primary responsibilities include:

* Reading and interpolation of modules and configuration files by Infrastructure as code functionalities
* Resource Graph Construction
* Plugin communication through RPC
* Plan execution
* Management of resource state

**Question 1: What do you mean by Terraform?**

**Answer:** Terraform is open-source communication as a system software tool created by **[HashiCorp](https://k21academy.com/cloud-infrastructure-automation-certification-terraform-associate-self-study-training/" \t "_blank)**. 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.

**Question 2: What are the reasons for choosing Terraform for DevOps?**

**Answer:** Below 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**](https://k21academy.com/google-cloud/introduction-to-google-cloud-platform/), 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.

**Question 3: Define Terraform init?**

**Answer:**Terraform initialises the code with the command [**terraform init**](https://k21academy.com/terraform-iac/terraform-workflow-and-its-use-case/). This command is used to set up the working directory for Terraform configuration files. It is safe to run this command multiple times.

You can use the init command for:

1. Installing Plugins
2. Installation of a Child Module
3. Initialization of the backend

**Question 4: Name some major competitors of Terraform?**

**Answer:**Some of them are:

* Packer
* Cloud Foundry
* **Ansible**
* Kubernetes

**Question 5: Define Terraform provider?**

**Answer:**Terraform is a tool for managing and informing infrastructure resources such as physical machines, virtual machines (VMs), network switches, containers, and more. A provider is responsible for API interactions that are thoughtful and reveal resources. Terraform is compatible with a wide range of cloud providers.

**Question 6: How does Terraform work?**

**Answer:**Terraform creates an implementation plan, defines what it will do to achieve the desired state, and then executes it to build the infrastructure described. Terraform is capable of determining what changed and generating incremental execution plans that are practical as the configuration changes**.**

**Question 7: Name some major features of Terraform?**

**Answer:**Some of them are:

* Execution Plan
* Change Automation
* Resource Graph
* Infrastructure as code

**Question 8: Define IAC?**

**Answer: 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 IAC](https://k21academy.com/terraform-iac/day1-training/), for example, is a case tool of IAC.

**Question 9: How to check the installed version of Terraform?**

**Answer:**We can use **terraform -version** of the command to identify the version which we are running.

**Question 10: Describe the working of Terraform core?**

Answer: The terraform core examines configuration monitoring and generates configuration-based analysis and evaluation. It keeps track of and compares versions (current and previous) before displaying the results via 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.

**Question 11:  What are the key features of Terraform?**

**Answer:**Following 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**](https://k21academy.com/cloud-infrastructure-automation-certification-terraform-associate-self-study-training/) 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 recognises

**Question 12: What are the use cases of Terraform?**

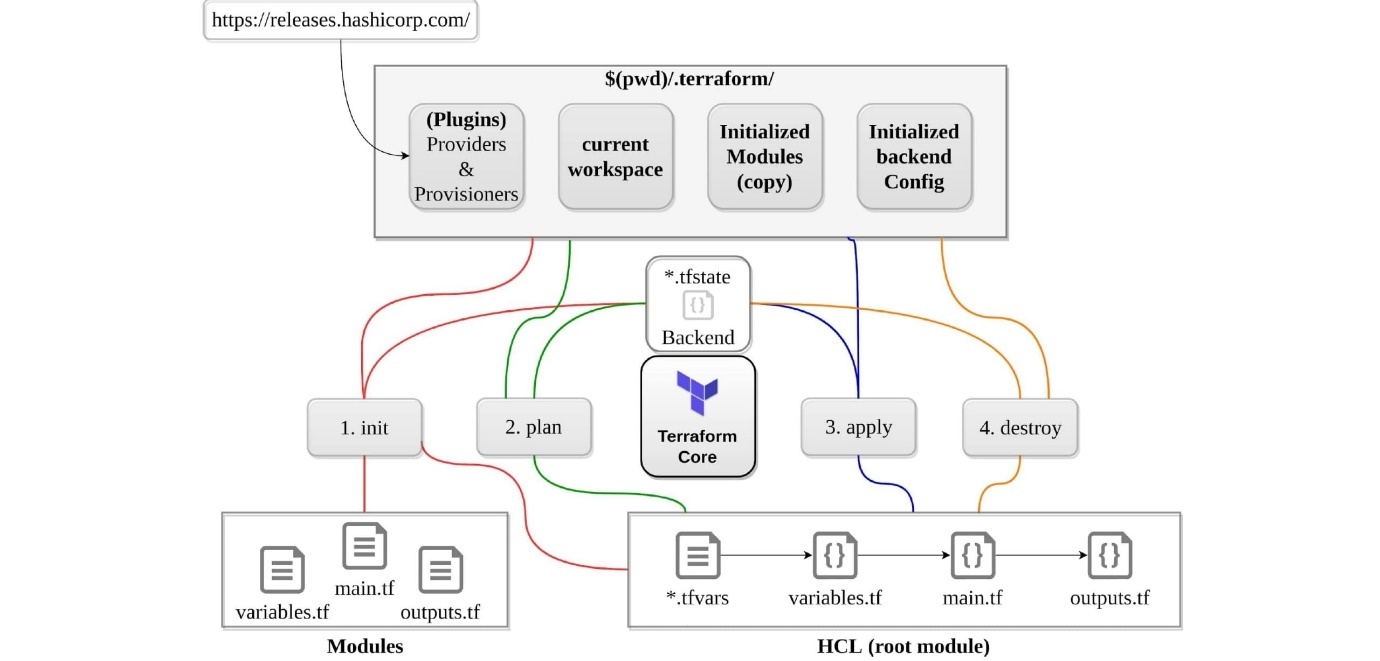
 Following are the use cases of Terraform:

* **Setting Up a Heroku App:**
  + Heroku is a popular [platform as a service](https://k21academy.com/amazon-web-services/cloud-service-models/) for hosting web applications (PaaS). Developers first create an app, then add add-ons such as a database or an email service. One of the best features is the ability to scale the number of dynos or workers as needed. Most non-trivial applications, on the other hand, quickly necessitate a large number of add-ons and external services.
  + Terraform can be used to codify the configuration of a Heroku application, ensuring that all necessary add-ons are present, but it can also go beyond, such as configuring DNSimple to set a CNAME or Cloudflare as the app’s CDN. Best of all, Terraform can accomplish all of this in under 30 seconds without the use of a web interface.
* **Clusters of Self-Service:**
  + A centralised operations team overseeing a large and expanding infrastructure becomes extremely difficult at a certain organisational level. Implementing “self-service” infrastructure, which allows product teams to manage their own infrastructure using tooling provided by the central operations team, becomes more appealing.
  + Terraform configuration can be used to keep track of how a service is built and scaled. You can then share these settings with the rest of your company, allowing client teams to manage their services using Terraform.
* **Quick Creation of Environments:**
  + It is common to have both a production and a staging or quality assurance environment. These environments are miniature versions of their production counterparts, and they are used to test new programmes before they are released to the public. Maintaining an up-to-date staging environment becomes increasingly difficult as the production environment grows larger and more involved.
  + **Terraform** can be used to codify and share the production environment with staging, QA, and development. These parameters can be quickly used to create new testing environments that can be easily discarded. Terraform, which allows parallel environments to be created and destroyed on the fly, can help to alleviate the difficulty of maintaining them.
* **Schedulers of Resources:**
  + Static application assignment to computers becomes increasingly difficult in large-scale infrastructures. Among the schedulers that can help with this challenge are Borg, Mesos, YARN, and Kubernetes. These can be used to schedule Docker containers, Hadoop, Spark, and a variety of other software applications dynamically.
  + Terraform isn’t just for [**Amazon Web Services**](https://k21academy.com/amazon-web-services/overview-of-amazon-web-services-concepts/) and other physical service providers. Terraform can request resources from resource schedulers because they can be viewed as providers. Terraform can now work in layers, such as deploying the physical infrastructure that powers the schedulers and provisioning into the scheduled grid.
* **Demonstrations of software:**
  + In today’s world, software is becoming increasingly networked and distributed. Although virtualized demo environments can be created with tools such as Vagrant, displaying software on real infrastructure that closely replicates production environments remains difficult.
  + A Terraform configuration can be used by software authors to design, provision, and bootstrap a demo on cloud providers such as AWS. End users can simply demo the application on their own infrastructure, and configuration options such as cluster size can be changed to evaluate tools at any scale.

### ****Question 13: What are the most useful Terraform commands?****

Common commands:

* **terraform init**: Prepare your working directory for other commands
* **terraform plan**: Show changes required by the current configuration
* **terraform apply**: Create or update infrastructure
* **terraform destroy**: Destroy previously-created infrastructure



### ****Question 14: How does Terraform help in discovering plugins?****

**Answer:**Terraform interprets configuration files in the operational directory with the authority “Terraform init.” Then, Terraform determines the necessary plugins and searches for installed plugins in various locations. [**Terraform**](https://k21academy.com/terraform-for-beginners-and-certification-free-class/) may also download additional plugins at times. Then it decides which plugin versions to use and creates a security device file to ensure that Terraform uses the same plugin versions.

### ****Question 15: Can I add policies to the open-source or pro version of Terraform enterprise?****

**Answer: Terraform**Policies cannot be added to Terraform Enterprise’s open-source description. The same is true for the Enterprise Pro edition. Terraform Enterprise’s best version could only contact the watch policies.

### ****Question 16: Define Modules in Terraform?****

**Answer:**A module in Terraform is a container for multiple resources that are used in tandem. Every Terraform that includes resources mentioned in.tf files requires the root module.

### ****Question 17: What are the ways to lock Terraform module versions?****

**Answer:**You can use the terraform module registry as a source and specify the attribute’version’ in the module in a terraform configuration file. If you are using the GitHub repository as a source, you must use ‘? ref’ to specify the branch, version, and query string.

### ****Question 18: What do you mean by Terraform cloud?****

**Answer:**Terraform Cloud is an application that enables teams to use Terraform collaboratively. It manages Terraform runs in a consistent and reliable environment, and includes features such as easy access to shared state and secret data, access controls for approving infrastructure changes, a private registry for sharing Terraform modules, detailed policy controls for governing the contents of Terraform configurations, and more.

### ****Question 19: Define null resource in Terraform?****

**Answer:**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**](https://k21academy.com/terraform-iac/terraform-provisioners/).

### ****Question 20: Can Terraform be used for on-prem infrastructure?****

**Answer:**Yes, Terraform can be used to build on-premises infrastructure. There are numerous providers available. You can select whichever one best suits your needs. Many people create client Terraform providers for themselves; all that is required is an API.

### ****Question 21 : What does the following command do?****

**Answer:**

* **Terraform -version** – to check the installed version of terraform
* **Terraform fmt**– it is used to rewrite configuration files in canonical styles and format
* **Terraform providers** – it gives information of providers working in the current configuration.

### ****Question 22: List all the Terraform-supported versions****

**Answer:**

* GitHub.com
* GitLab.com
* GitHub Enterprise
* GitLab CE and EE
* Bitbucket Cloud and Server
* Azure DevOps Server and Services

### ****Question 23: Explain the command terraform validate in the context of Terraform.****

**Answer:**The terraform validate command examines the configuration files in a directory, concentrating solely on the configuration and ignoring any external services such as remote state, provider APIs, and so on. Validate inspects a configuration to determine whether it is syntactically correct and internally consistent, regardless of variables or current state. As a result, it’s best for general reusable module verification, such as confirming the validity of attribute names and value types. This command can be executed automatically, such as a post-save check in a text editor or a test step in a continuous integration system for a reusable module.**Syntax: terraform validate [options]**

### ****Question 24:**** Mention some of the version control tools supported by Terraform.

**Answer:**Version control tools supported by Terraform are:

* GitHub
* GitLab CE
* GitLab EE
* Bucket Cloud

## Terraform Interview Questions and Answers for Experienced Scenario Based

### ****Question 25: How would you recover from a failed apply in Terraform?****

**Answer:**You can save your configuration in version control and commit it before making any changes, and then use the features of your version control system to revert to an earlier configuration if necessary. You must always recommit the previous version code in order for it to be the new version in the version control system.

### ****Question 26: 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**](https://k21academy.com/amazon-web-services/aws-solutions-architect/create-aws-free-tier-account/).

### ****Question 27: What steps should be followed for making an object of one module to be available for the other module at a high level?****

**Answer:** The following are the steps to take in order to make an object from one module available to the other module at a high level:

1. First, in a resource configuration, an output variable must be defined. The scope of local and to a module is not declared until you declare resource configuration details.
2. You must now declare the output variable of module A so that it can be used in the configurations of other modules. You should create a brand new and current key name, and the value should be kept equal to the module A output variable.
3. You must now create a file variable.tf for module B. Create an input variable inside this file with the same name as the key you defined in module B. This variable in a module enables the resource’s dynamic configuration. Rep the process to make this variable available to another module as well. This is due to the fact that the variable established here has a scope limited to module B.

### ****Question 28:**** ****What is State File Locking?****

**Answer:** 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.

### ****Question 29: What is a Remote Backend in Terraform?****

**Answer:** [**Terraform**](https://k21academy.com/cloud-infrastructure-automation-certification-terraform-associate-self-study-training/)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, untaint, 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.

### ****Question 30: What is a Tainted Resource?****

**Answer:** Tainted resources are those that must be destroyed and recreated upon the next apply 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 apply occurs.

##### Read More: [Terraform Workflow](https://k21academy.com/terraform-iac/terraform-workflow-and-its-use-case/)

### ****Question 31: Are callbacks possible with Terraform on Azure?****

**Answer:** Terraform uses Azure Event Hub to perform Azure callbacks. It aids in achieving functionality such as sending a callback to the system and other events. To make the process easier, Terraform AzureRM already includes this functionality.

### ****Question 32:****How to prevent Error Duplicate Resource

**Answer:** It can be done in three ways depending on the situation and the requirement

1) By deleting the resource, Terraform code will no longer manage it.  
2) By removing resources from APIs  
3) Importing action will also aid in resource elimination.

### ****Question 33:****Explain the workflow of the core terraform.

**Answer:**Terraform’s core workflow has three steps:

* **Write** – Create infrastructure in the form of code.
* **Plan** – Plan ahead of time to see how the changes will look before they are implemented.
* **Apply** – Create a repeatable infrastructure.

**1. What exactly do you mean by Terraform in AWS?**

Terraform, a part of the AWS DevOps Competency and an AWS Partner Network (APN) Advanced Technology, is an "infrastructure as code" tool comparable to AWS CloudFormation that lets you design, edit, and version your Amazon Web Services (AWS) infrastructure. You may programmatically control your on-premises and public cloud resources by developing Terraform's custom AWS CloudFormation resource. You may access that resource directly from the CloudFormation UI or via the AWS Service Catalog, which adds an extra degree of governance and control.

**2. What are the essential aspects of Terraform?**

* Terraform allows you to manage all your infrastructures as code and build them as required. The essential features are
* A console that will enable users to watch functions in action—the capability to convert HCL code to JSON format.
* Interpolation is supported through a configuration language.
* A module count maintains track of how many modules have been installed on the infrastructure.

**3. Define IAC**

Infrastructure as Code (IAC) allows one to code instead of using manual procedures to construct, alter, and maintain the infrastructure. The configuration files may be modified and transmitted securely within an organisation built according to infrastructure standards.

**4. What Terraform commands are the most useful?**

The useful Terraform commands are as follows:

* Terraform init - Creates a new directory in the current directory.
* Terraform refresh - This command updates the state file.
* Terraform output - Views Outputs of Terraform
* Terraform apply - Executes the Terraform code and creates objects.
* Terraform Destroy - Destroys what Terraform has constructed.
* Terraform graph - Generates a graph in DOT format.
* Terraform plan - Tests how Terraform will perform.

**5. Is it feasible to use callbacks with Terraform on Azure?**

Callbacks are possible with Terraform on Azure if you use Azure Event Hubs. Terraform's Azure supplier offers users simple functionality. The Terraform occurrence has already been set up in Microsoft Azure Cloud Shell.

|  |
| --- |
| If you want to enrich your career and become a **professional in Terraform**, then enrol in "[**Terraform Online Training**](https://mindmajix.com/terraform-training)" - This course will help you to achieve excellence in this domain. |

**6. What is Terraform init, and how does it work?**

Terraform init is a control that allows you to set up an operational index with Terraform pattern files. This control can be repeated several times. It should be the initial command executed after creating a new Terraform design.

**7. What is Terraform D, and how does it work?**

Terraform D is a Windows plugin that works with almost all in-service systems. By default, Terraform init looks for plugins in the following directory.

**8. While utilizing the TFS API to deliver resources, is the history the same as the web?**

Yes, the narrative is comparable to what you'd get on the web because UI uses API as its foundation. Everything on the UI is available through additional methods and the API.

**9. What is the purpose of Terraform in DevOps?**

The HashiCorp Configuration Language, which is comparable to JSON, is used by Terraform (HCL). HCL provides a concise vocabulary that makes establishing and enforcing infrastructure settings across multiple clouds and on-premises data centres simple for DevOps teams.

**10. In Terraform, what is the purpose of a null resource?**

The standard resource library is implemented by null resource, but no further action is done. The triggers option allows you to provide an arbitrary collection of values that will cause resources to be replaced if they are modified.

[](https://bit.ly/3if9dmk)

**11. What is a Terraform cloud?**

The platform which allows teams to collaborate on Terraform projects on-demand or in reaction to specific circumstances is the Terraform cloud. It is tightly connected with Terraform's processes and data. Terraform modules are shared by a private registry.

**12. Describe Oracle Cloud Infrastructure in detail.**

Oracle Cloud is a cloud computing service consisting of storage, servers, applications, services, and networks supplied through a global network of Oracle Corporation managed data centers. These services can be accessed at any time over the Internet.

**13. What is "terraform backend"?**

A backend may be specified in any Terraform setup, which defines two key things:

* Where are surgeries carried out?
* Where is the state saved? (In a state file, Terraform maintains track of all the resources generated.)

**14. What version control systems do Terraform support in addition to GitHub?**

* GitLab EE
* GitLab CE
* Bucket cloud.

**15. Who are Terraform's main competitors?**

Terraform's key competitors and alternatives are Azure Management Tools, Morpheus, CloudHealth, Turbonomic, and CloudBolt.

**16. What is the purpose of Terraform CLI, and what are some common CLI commands?**

Terraform's Command-Line Interface (CLI) is used to manage infrastructure and communicate with Terraform state, configuration files, and providers, among other things.

Here are some simple CLI commands to get you started:

* Terraform init - Sets up your working directory to run additional commands.
* Terraform destroy - Destroys the infrastructure you've already built.
* Terraform validate - Check if the configuration is correct using terraform validate.
* Terraform apply - This command develops or changes infrastructure.
* Terraform plan - Depicts the modifications that the existing arrangement necessitates.

**17. What are Terraform modules?**

In Terraform, a module is a container containing many resources utilized together. The root module is required for every Terraform and includes the resources listed in the. tf files.

**18. What is a Private Module Registry, and how does it work?**

Terraform Cloud has a Private Module Registry feature that allows you to distribute Terraform modules throughout your enterprise. On the registry, you may set rules or "sentinel policies" that govern how members of your organization can use the modules.

**19. Can Terraform be used to build on-premise infrastructure?**

Yes, Terraform may be used to create infrastructure on-premise. We may select from a variety of vendors to discover which one best meets our requirements. We only need an API to get started.

**20. Can I use Terraform to deploy several providers?**

Yes, Terraform supports multi-provider installations, including on-premises solutions like OpenStack and VMware, as well as SDN management.

**21. How is a duplicate resource error during terraform application ignored?**

We can experiment with the following options:

* Delete the resources from the cloud provider's API and rebuild them with Terraform.
* To stop Terraform from managing specific resources, remove them from the code.
* Perform a resource terraform import and delete the code attempting to replicate them.

**22. List all of Terraform's version control options.**

The following version controls are supported:

* Azure DevOps Services.
* Azure DevOps Server
* Bitbucket Server
* Cloud Bitbucket
* Gitlab Enterprise Edition (EE) and Enterprise Edition (CE)
* Gitlab.com
* GitHub Enterprise.
* GitHub.com (OAuth)
* GitHub.com

**23. What are some of the Terraform provisioners that are pre-installed?**

The list of Terraform's built-in provisioners are:

* Provisioner without a salt-master
* Provisioner for Remote Execution
* Provisioner of Puppets
* Provisioner with local authority
* Provider of Habitat
* Provisioner of Files
* Provisioner Chef

**24. What command is used to remove Terraform-managed infrastructure?**

For this, the following command is used:

[options] [dir] terraform destroy

**25. Tell us some significant Terraform applications.**

Terraform has a wide range of applications owing to its ability to enhance its resource manipulation capabilities. The following are some of the unique applications:

* Development of software demonstrations
* Schedulers of resources
* The use of several clouds
* Environment inventions that can be thrown away
* Development of multi-tiered apps
* Clusters of self-service
* Heroku App Configuration

**26. What are the components that makeup Terraform's architecture?**

The following components are included in the Terraform architecture:

* Sub-graphs
* Evaluation of Expression
* Evaluation of the Vertex
* Walk the Graph
* Builder of Graphs
* Manager of the State
* Loader Configuration
* CLI (Common Language Interface) (Command Line interface)
* Backend

**1. What are the key features of Terraform?**

Following are the key features of Terraform:

* **Infrastructure as Code:**Terraform's high-level configuration language is used to describe your infrastructure in declarative configuration files that are human-readable. You may now generate a blueprint that you can edit, share, and reuse.
* **Execution Strategies:**Before making any infrastructure modifications, Terraform develops an execution plan that describes what it will do and asks for your agreement. Before Terraform produces, upgrades, or destroys infrastructure, you can evaluate the changes.
* **Graph of Resources:**Terraform develops or alters non-dependent resources while simultaneously building a resource graph. This allows Terraform to construct resources as quickly as possible while also providing you with more information about your infrastructure.
* **Automation of Change:**Terraform can automate the application of complex changesets to your infrastructure with little to no human intervention. Terraform identifies what happened when you update configuration files and provides incremental execution plans that take dependencies into account.

**2. What are the use cases of Terraform?**

 Following are the use cases of Terraform:

* **Setting Up a Heroku App:**
  + Heroku is a prominent platform as a service (PaaS) for hosting web applications. Developers build an app first, then add add-ons like a database or an email service. The ability to elastically scale the number of dynos or workers is one of the nicest features. Most non-trivial applications, on the other hand, quickly require a large number of add-ons and external services.
  + Terraform may be used to codify the setup required for a Heroku application, ensuring that all essential add-ons are present, but it can also go beyond, such as configuring DNSimple to set a CNAME or configuring Cloudflare as the app's CDN. Best of all, Terraform can achieve all of this without using a web interface in about 30 seconds.



* **Clusters of Self-Service:**
  + A centralised operations team managing a huge and growing infrastructure becomes extremely difficult at a given organisational level. Making "self-serve" infrastructure, which allows product teams to manage their own infrastructure using tooling given by the central operations team, becomes more appealing.
  + Terraform configuration may be used to document knowledge about how to construct and scale a service. You may then publish these configurations across your business, allowing client teams to administer their services using Terraform.
* **Quick Creation of Environments:**
  + It is usual to have both a production and staging or quality assurance environment. These environments are smaller clones of their production counterparts, and they're used to test new apps before they're released to the public. Maintaining an up-to-date staging environment gets increasingly difficult as the production environment grows larger and more complicated.
  + Terraform may be used to codify the production environment, which can then be shared with staging, QA, and development. These settings can be used to quickly create new environments in which to test and then readily discarded. Terraform can help to reduce the challenge of sustaining parallel environments by making it possible to create and destroy them on the fly.
* **Deployment of Multiple Clouds:**
  + To boost fault tolerance, it's common to disperse infrastructure across different clouds. When only one region or cloud provider is used, fault tolerance is restricted by that provider's availability. When a region or an entire provider goes down, a multi-cloud strategy provides for more gentle recovery.
  + Because many existing infrastructure management solutions are cloud-specific, implementing multi-cloud installations can be difficult. Terraform is cloud-agnostic, allowing you to manage numerous providers and even cross-cloud dependencies with a single configuration. This helps operators develop large-scale multi-cloud infrastructures by simplifying management and orchestration.
* **Applications with Multiple Tier Architecture:**
  + The N-tier architecture is a relatively popular structure. A pool of web servers with a database tier is the most popular 2-tier system. API servers, cache servers, routing meshes, and more levels are added. Because the stages can be scaled individually and provide a separation of concerns, this structure is used.
  + Terraform is a great tool for creating and managing these types of infrastructures. Terraform will automatically handle the dependencies between each layer if you arrange resources in each tier together. Before provisioning the web servers, Terraform will check that the database layer is up and running, as well as that the load balancers are linked to the web nodes. The count configuration value can then be changed in Terraform to quickly scale each tier. Because resource creation and provisioning are codified and automated, elastic scaling in response to load is a breeze.
* **Schedulers of Resources:**
  + The static assignment of applications to computers in large-scale infrastructures becomes increasingly difficult. There are a variety of schedulers available to handle this problem, including Borg, Mesos, YARN, and Kubernetes. These may be used to schedule Docker containers, Hadoop, Spark, and a variety of other software tools on a dynamic basis.
  + Terraform isn't just for physical providers like Amazon Web Services. Terraform can request resources from resource schedulers because they can be treated as providers. This allows Terraform to work in layers, such as setting up the physical infrastructure that runs the schedulers and provisioning into the scheduled grid.
* **Software-Defined Networking:**
  + SDN (Software Defined Networking) is becoming more common in data centers as it gives operators and developers more control over the network and allows the network to better support the applications running on top. A control layer and an infrastructure layer are common in SDN systems.
  + Terraform may be used to codify software-defined network setup. By interacting with the control layer, Terraform may use this configuration to automatically set up and adjust settings. This makes it possible to version the settings and automate modifications. Terraform, for example, may be used to set up an AWS VPC.
* **Demonstrations of software:**
  + Modern software is becoming increasingly networked and distributed. Although solutions such as Vagrant exist to create virtualized environments for demos, demonstrating software on real infrastructure that more closely resembles production environments remains difficult.
  + On cloud providers like AWS, software authors can give a Terraform configuration to develop, provision, and bootstrap a demo. End customers may simply demo the programme on their own infrastructure, and settings like cluster size can be tweaked to more thoroughly test tools at any scale.

**3. Is it feasible to use Terraform on Azure with callbacks? Sending a callback to a logging system, a trigger, or other events, for example?**

Yes. Azure Event Hubs can be used to accomplish this. This capability is now accessible in the Terraform AzureRM provider. Terraform's Azure supplier provides users with simple functionality. Microsoft Azure Cloud Shell includes a Terraform occurrence that has already been setup.

**4. What do you mean by terraform init in the context of Terraform?**

The terraform init command creates a working directory in which Terraform configuration files can be found. After creating a new Terraform configuration or cloning an old one from version control, run this command first. It is safe to use this command more than once. Despite the fact that successive runs may result in errors, this command will never overwrite your current settings or state.

**Syntax:**

terraform init [options]

The following options can be used in conjunction with the init command :

* **-input=true:**This option is set to true if the user input is mandatory. If no user input is provided, an error will be thrown.
* **-lock=false:** This option is used to disable the locking of state files during state-related actions.
* **-lock-timeout=<duration>:** This option is used to override the time it takes Terraform to get a state lock. If the lock is already held by another process, the default is 0s (zero seconds), which results in an immediate failure.
* **-no-color:** This option disables the color codes in the command output.
* **-upgrade:** This option can be chosen to upgrade modules and plugins throughout the installation process.

**5. Why is Terraform preferred as one of the DevOps tools?**

 Following are the reasons that Terraform is preferred as one of the DevOps tools :

* Terraform allows you to specify infrastructure in config/code, making it simple to rebuild, alter, and track infrastructure changes. Terraform is a high-level infrastructure description.
* While there are a few alternatives, they are all centred on a single cloud provider. Terraform is the only powerful solution that is totally platform-neutral and supports different services.
* Terraform allows you to implement a variety of coding concepts, such as putting your code under version control, writing automated tests, and so on.
* Terraform is the best tool for infrastructure management since many other solutions suffer from an impedance mismatch when attempting to use an API meant for configuring management to govern an infrastructure environment. Instead, Terraform is a perfect match for what you want to do because the API is built around how you think about infrastructure.
* Terraform has a thriving community and is open source, so it's attracting a sizable following. Many people already use it, making it easy to discover individuals who know how to use it, as well as plugins, extensions, and expert assistance. Terraform is also evolving at a much faster rate as a result of this. They have a lot of releases.
* Terraform's speed and efficiency are unrivalled. Terraform's plan command, for example, allows you to see what changes you're about to make before you do them. Terraform and its code reuse feature makes most modifications faster than similar tools like CloudFormation.

**6. Mention some of the major competitors of Terraform.**

 Following are some of the major competitors of Terraform:



* Azure Management Tools.
* Morpheus.
* CloudHealth.
* Turbonomic.
* CloudBolt.
* Apptio Cloudability
* Ansible
* Kubernetes
* Platform9 Managed Kubernetes.

### 7. What do you understand about Terraform Cloud?



Terraform Cloud is a collaboration tool for teams using Terraform. It offers easy access to shared state and secret data, access controls for approving infrastructure modifications, a private registry for sharing Terraform modules, full policy controls for managing the contents of Terraform configurations, and more. Terraform Cloud is a hosted service that can be found at https://app.terraform.io. Terraform allows small teams to connect to version control, share variables, run Terraform in a reliable remote environment, and securely save remote state for free. Paid tiers provide you with the ability to add more than five people, establish teams with varying levels of access, enforce policies before building infrastructure, and work more efficiently.

Large businesses can utilise the Business tier to scale to multiple concurrent runs, establish infrastructure in private environments, manage user access using SSO, and automate infrastructure end-user self-service provisioning.

### 8. Explain the destroy command in the context of Terraform.

The terraform destroy command is a simple way to eliminate all remote objects maintained by a Terraform setup. While you should avoid destroying long-lived objects in a production environment, Terraform is occasionally used to manage temporary infrastructure for development, in which case you can use terraform destroy to quickly clean up all of those temporary objects after you're done.

**Syntax:**terraform destroy [options]

You may also execute the following command to build a speculative destroy plan to see what the effect of destroying might be:

terraform -destroy plan

This will launch Terraform Plan in destroy mode, displaying the proposed destroy changes but not allowing you to execute them.

### 9. What do you understand about Terraform modules?

 A Terraform module is a single directory containing Terraform configuration files. Even a simple arrangement with a single directory having one or more files can be referred to as a module. The files have the extension .tf.  This directory is referred to as the root module when Terraform commands are run directly from it. Terraform commands will only use the configuration files in one location: the current working directory. Your configuration, on the other hand, can employ module blocks to call modules from other directories. When Terraform comes across a module block, it loads and processes the configuration files for that module. A module that is called by another configuration is frequently referred to as that configuration's "child module."

### 10. What are the benefits of using modules in Terraform?

 Following are the benefits of using modules in Terraform :

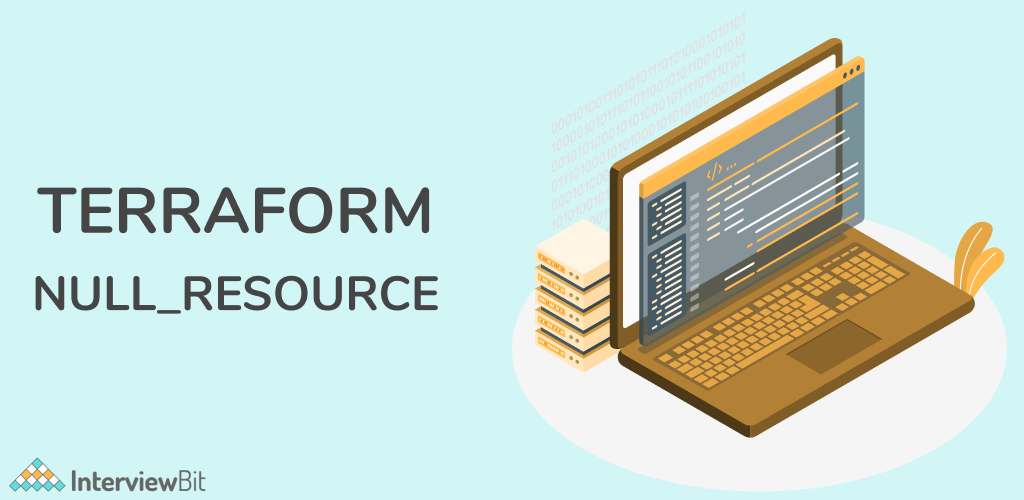
* **Organization of configuration:** By grouping relevant portions of your configuration together, modules make it easier to access, understand, and change your configuration. Hundreds or thousands of lines of configuration can be required to establish even moderately complicated infrastructure. You can organise your configuration into logical components by utilising modules.
* **Encapsulation of configuration:** Another advantage of modules is that they allow you to separate configuration into logical components. Encapsulation can help you avoid unforeseen consequences, such as a change to one element of your configuration causing changes to other infrastructure, and it can also help you avoid basic mistakes like naming two resources with the same name.
* **Maintains consistency and ensures best practices:** Modules can also help you maintain uniformity in your configurations. Consistency not only makes complex configurations easier to grasp, but it also ensures that best practices are followed in all of your settings. Cloud providers, for example, offer a variety of options for establishing object storage services like Amazon S3 or Google Cloud Storage buckets. Many high-profile security problems have occurred as a result of improperly secured object storage, and given the number of sophisticated configuration options involved, it's possible to misconfigure these services by accident.
* **Modules can aid in the reduction of errors:** For example, you might design a module to define how all of your organization's public website buckets would be set, as well as a separate module for private logging buckets. In addition, if a configuration for a particular resource type needs to be altered, using modules allows you to do it in one place and have it applied to all scenarios where that module is used.
* **Aids in reusability:** Setting up the configurations from scratch and writing all of your settings can be time-consuming and error-prone. By reusing configuration generated by yourself, other members of your team, or other Terraform practitioners who have published modules for you to utilise, you can save time and avoid costly errors. You can also share modules you've produced with your colleagues or the broader public, allowing them to profit from your efforts.

### 11. What are some guidelines that should be followed while using Terraform modules?

Following are some of the guidelines that should be followed while using Terraform modules :

* To publish to the Terraform Cloud or Terraform Enterprise module registries, you must use this convention terraform-<PROVIDER>-<NAME>.
* Start thinking about modules as you write your setup. The benefits of using modules outweigh the time it takes to utilise them properly, even for somewhat complicated Terraform settings maintained by a single person.
* To organise and encapsulate your code, use local modules. Even if you aren't using or publishing remote modules, structuring your configuration in terms of modules from the start will dramatically minimise the time and effort required to maintain and update your setup as your infrastructure becomes more complicated.
* To identify useful modules, go to the Terraform Registry, which is open to the public. By relying on the efforts of others to create common infrastructure scenarios, you may implement your configuration more quickly and confidently.
* Modules can be published and shared with your team. The majority of infrastructure is handled by a group of individuals, and modules are a vital tool for teams to collaborate on infrastructure creation and maintenance.

### 12. Explain null resource in the context of Terraform.



The null resource is a resource that lets you set up provisioners that aren't directly linked to any current resource. Because a null resource behaves like any other resource, you can configure provisioners, connection details, and other meta-parameters just like any other resource. This gives you more precise control over when provisioners execute in the dependency graph.

### 13. Explain the command terraform validate in the context of Terraform.

The terraform validate command verifies the configuration files in a directory, focusing solely on the configuration and excluding any outside services such as remote state, provider APIs, and so on. Validate performs checks to see if a configuration is syntactically correct and internally consistent, regardless of any variables or current state. As a result, it's best used for general verification of reusable modules, such as ensuring that attribute names and value types are correct. This command can be executed automatically, for example as a post-save check in a text editor or as a test step for a reusable module in a continuous integration system.

**Syntax:**terraform validate [options]

The following options are available with this command:

* **-json** - Create output in the machine-readable JSON format, appropriate for integration with text editors and other automated systems. Color is always turned off.
* **-no-color** - If supplied, the output will be colourless.

### 14. Explain the command terraform apply in the context of Terraform.

The terraform apply command is used to carry out the tasks in a Terraform plan. The simplest method to use terraform apply is to run it without any arguments, in which case it will construct a new execution plan (as if you had run terraform plan) and then request you to accept it before doing the activities you specified. Another approach to use terraform apply is to supply it the filename of a saved plan file generated with terraform plan -out=..., in which case Terraform will apply the modifications to the plan without prompting for confirmation. This two-step process is most useful when using Terraform in an automated environment.

**Syntax:**

terraform apply [options] [plan file]

### 15. Explain the command terraform version in the context of Terraform.

 The terraform version command shows the current Terraform version as well as any installed plugins.

**Syntax:**

terraform version [options]

Unless disabled, the version will display the Terraform version, the platform it's installed on, installed providers, and the results of upgrade and security checks with no extra arguments.

There is one optional flag for this command:

If you specify -json, the version information is formatted as a JSON object, with no upgrade or security information.

### 16. Mention some of the version control tools supported by Terraform.

Some of the version control tools supported by Terraform are as follows:



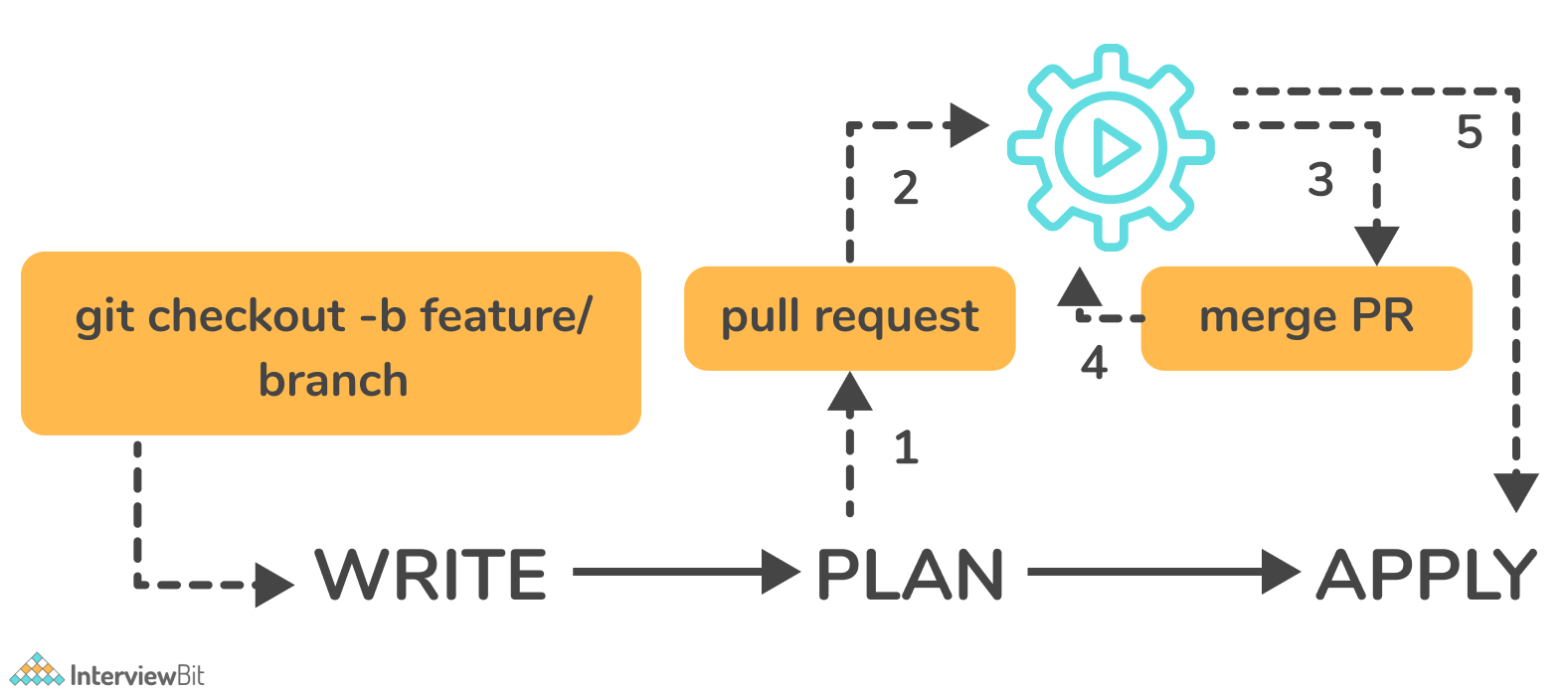
* GitHub
* GitLab CE
* GitLab EE
* Bucket Cloud

### 17. What do you understand about providers in the context of Terraform?

To interface with cloud providers, SaaS providers, and other APIs, Terraform uses plugins called "providers." Terraform configurations must specify the providers they need in order for Terraform to install and use them. Some providers also require setup (such as endpoint URLs or cloud regions) before they may be used. Terraform may manage a set of resource types and/or data sources that each provider contributes. A provider implements each resource type; Terraform would be unable to manage any infrastructure without them. The majority of service providers set up a specific infrastructure platform (either cloud or self-hosted). Local utilities, such as generating random numbers for unique resource names, can be offered by providers.

### 18. Explain the workflow of the core terraform.

Terraform's core workflow consists of three steps:

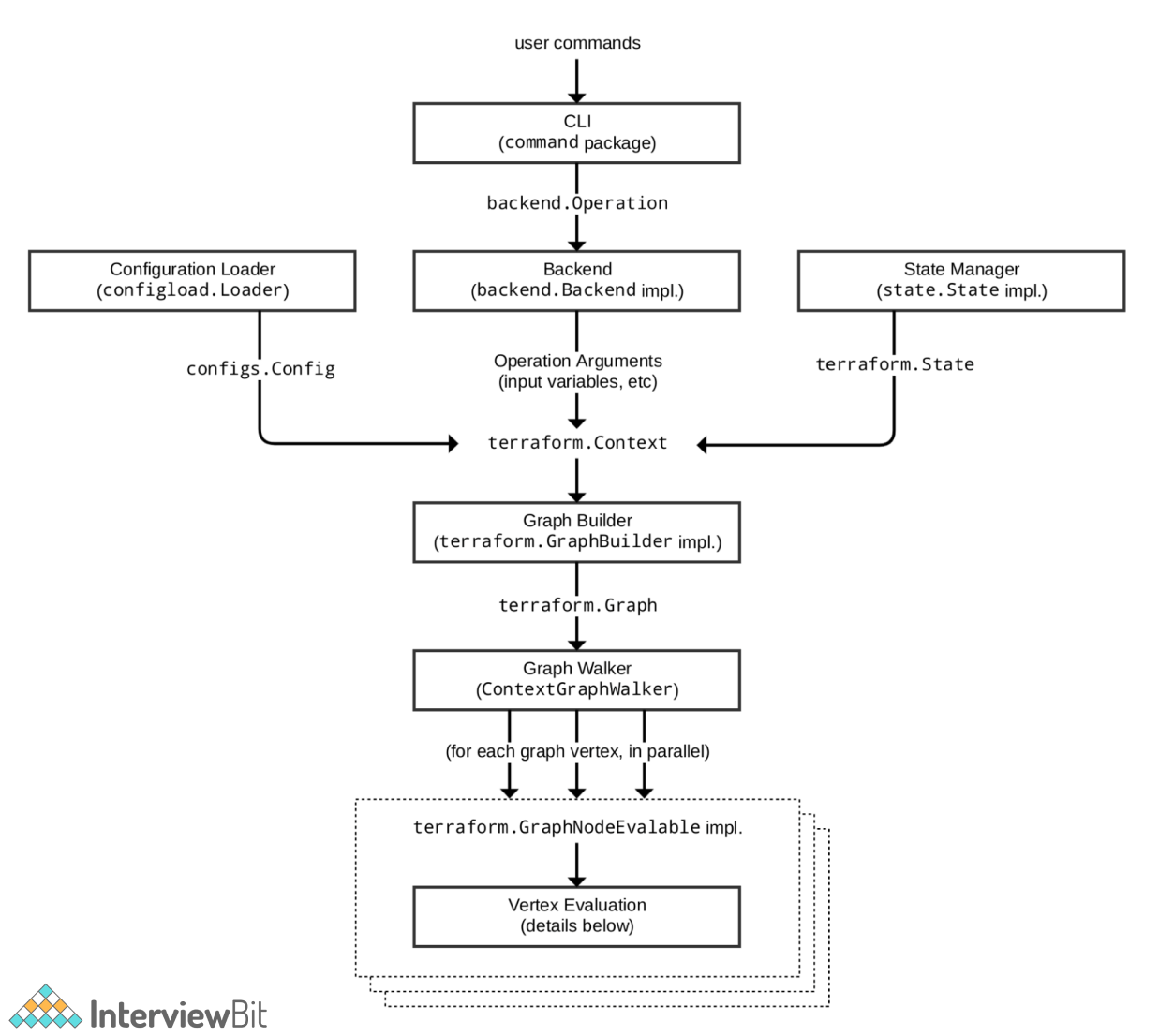


* **Write** - Create infrastructure in the form of code.
* **Plan** - Plan ahead of time to see how the changes will look before they are implemented.
* **Apply** - Create a repeatable infrastructure.

## Terraform Interview Questions for Experienced

### 19. Explain the architecture of Terraform request flow.

A request in Terraform undergoes the following steps as shown in the diagram:



Following are the different components in the above architecture:

**Command Line Interface (CLI):**

CLI (Common Language Interface) (command package)

Aside from some early bootstrapping in the root package (not shown in the diagram), when a user starts the terraform application, execution jumps right into one of the command package's "command" implementations. The commands.go file in the repository's root directory contains the mapping between user-facing command names and their respective command package types.

The job of the command implementation for these commands is to read and parse any command line arguments, command-line options, and environment variables required for the given command and use them to generate a backend.operation object. The operation is then transferred to the backend that is currently selected.

**Backends:**

In Terraform, a backend has a variety of responsibilities:

* Carry out operations (e.g. plan, apply)
* To save workspace-defined variables
* To save state

The local backend retrieves the current state for the workspace specified in the operation using a state manager (either statemgr.Filesystem if the local backend is being used directly, or an implementation provided by whatever backend is being wrapped), then uses the config loader to load and do initial processing/validation of the configuration specified in the operation. It then constructs a terraform.context object using these, as well as the other parameters specified in the procedure. The main object actually executes Terraform operations.

**Configuration Loader :**

Model types in package configs represent the top-level configuration structure. configs.Config represents an entire configuration (the root module and all of its child modules). Although the configs package offers some low-level functionality for creating configuration objects, the major entry point is via configload.Loader, which is found in the sub-package configload. When a configuration is loaded by a backend, a loader takes care of all the complexities of installing child modules (during terraform init) and then locating those modules again. It takes the path to a root module and loads all of the child modules in a recursive manner to create a single configs.

**State Manager:**

The state manager is in charge of storing and retrieving snapshots of a workspace's Terraform state. Each manager is an implementation of a subset of the statemgr package's interfaces, with most practical managers implementing the entire set of statemgr.Full's operations . The smaller interfaces are mostly for use in other function signatures to be specific about what actions the function might perform on the state manager; there's no reason to design a state manager that doesn't implement all of statemgr. Full.

**Graph Builder:**

The terraform.Context method invokes a graph builder. A graph builder is used to represent the essential steps for that operation and the dependence relationships between them. Because the graph-building process differs by operation, each has its own graph builder. A "plan" operation, for example, requires a graph created directly from the configuration, whereas an "apply" action creates its graph from the set of modifications stated in the plan being applied.

**Graph Walk:**

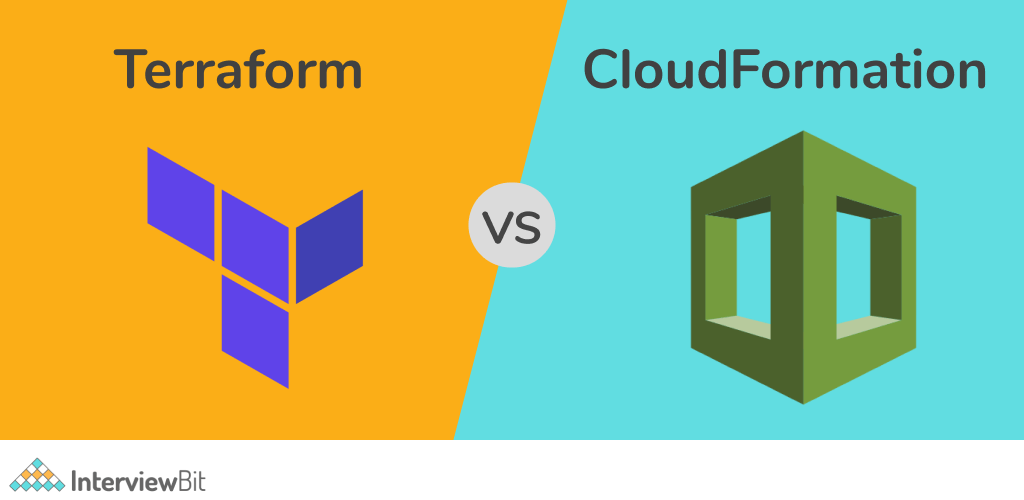
The graph walking method visits each vertex of the graph in a fashion that respects the graph's "happens after" edges. Each vertex in the graph is assessed in such a way that the "happens after" edges are taken into account. The graph walk method will assess many vertices at the same time if possible.

**Vertex Evaluation:**

During a graph walk, the action executed for each vertex is referred to as execution. Execution performs a series of random operations that make sense for a given vertex type. Before the graph walk begins evaluating further vertices with "happens after" edges, a vertex must be complete correctly. When one or more errors occur during evaluation, the graph walk is interrupted and the errors are returned to the user.

### 20. Differentiate between Terraform and Cloudformation.

The following points highlight the differences between Terraform and Cloudformation:



* **User-friendliness:**
  + Terraform encompasses numerous Cloud Service Providers such as AWS, Azure, Google Cloud Platform, and many more, while CloudFormation is limited to AWS services. Terraform covers the majority of AWS resources.
* **Based on language:**
  + CloudFormation employs either JSON or YAML as a language. CloudFormation is now simple to read and manage. However, AWS developers are restricted from creating CloudFormation templates that are more than 51MB in size. Developers must establish a layered stack for the templates if the template exceeds this size restriction.
  + Terraform, on the other hand, makes use of Hashicorp's own HCL language (Hashicorp Configuration Language). This language is also JSON compatible.
* **State-management:**
  + Because CloudFormation is an AWS managed service, it examines the infrastructure on a regular basis to see if the provisioned infrastructure is still in good shape. If anything changes, CloudFormation receives a thorough response.
  + Terraform, on the other hand, saves the state of the infrastructure on the provisioning machine, which can be either a virtual machine or a remote computer. The state is saved as a JSON file, which Terraform uses as a map to describe the resources it manages.
  + To summarise, Cloudformation's state is managed out-of-the-box by CloudFormation, which prevents conflicting updates. Terraform stores the state on a local disk, which makes it easier to synchronise the state. Terraform states can also be saved in storage services like S3, which is another recommended practice for state management. This must be defined on the backend, making management easier and safer.
* **Cost:**
  + The nicest aspect about both of these tools is that they are both completely free. Both of these technologies have sizable communities that provide plenty of help and examples. Cloudformation is completely free. The only expense that consumers pay is for the AWS service that CloudFormation provides. Terraform is a completely free and open-source application. Terraform, on the other hand, includes a premium enterprise version with more collaboration and governance features.
* **Integration of Multiple Clouds:**
  + Terraform is the way to go if you want to supply services across several cloud platforms. While Terraform works with a variety of cloud providers, including AWS, GCP, Azure, and others, CloudFormation is exclusive to AWS. Cloudformation is not for you if your setup includes several cloud installations. If you're using AWS resources like EC2, S3, and so on, you should stick to Cloudformation.

### 21. Explain the command terraform taint in the context of Terraform.

Terraform receives notification from the terraform taint command that a specific item has been degraded or damaged. This is represented by Terraform designating the item as "tainted" in the Terraform state, in which case Terraform will suggest replacing it in the next plan you write. If you want to compel the replacement of a specific object despite the fact that no configuration modifications are required, using the terraform apply -replace option is preferred.

Utilizing the "replace" option while creating a plan is preferable to using terraform taint because it allows you to see the entire impact of the alteration before taking any externally visible action. When you utilise terraform taint to achieve a similar impact, you run the danger of someone else on your team devising a new strategy to counter your tainted object before you've had a chance to consider the implications.

**Syntax:**

terraform taint [options] address

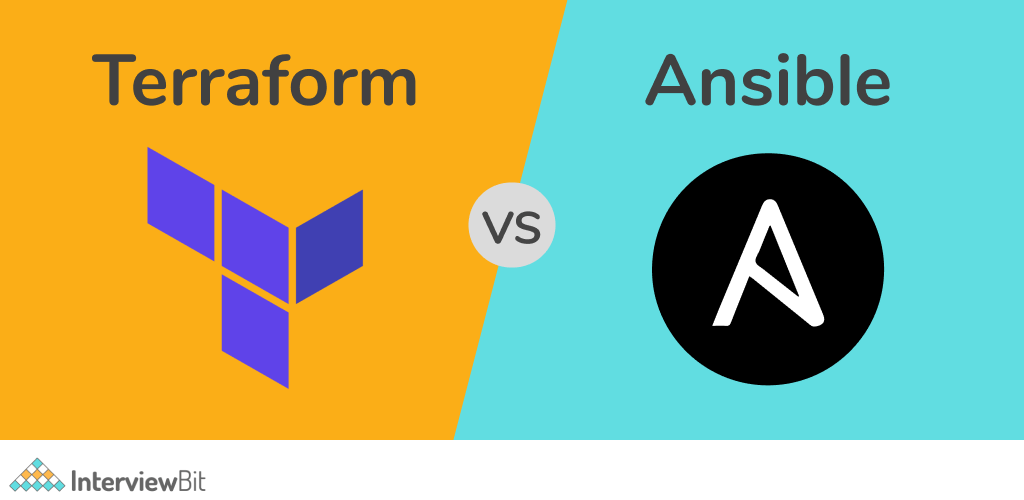
The address option specifies the location of the infected resource. The following options are available with this command:

* **-allow-missing** - Even if the resource is absent, the command will succeed (exit code 0) if it is supplied. Other scenarios, such as a problem reading or writing the state, may cause the command to return an error.
* **-lock=false** - Turns off Terraform's default behaviour of attempting to lock the state for the duration of the operation.
* **-lock-timeout=DURATION** - Instructs Terraform to reattempt procuring a lock for a period of time before issuing an error, unless locking is disabled with -lock=false. A number followed by a time unit letter, such as "3s" for three seconds, is the duration syntax.

### 22. Differentiate between Terraform and Ansible.

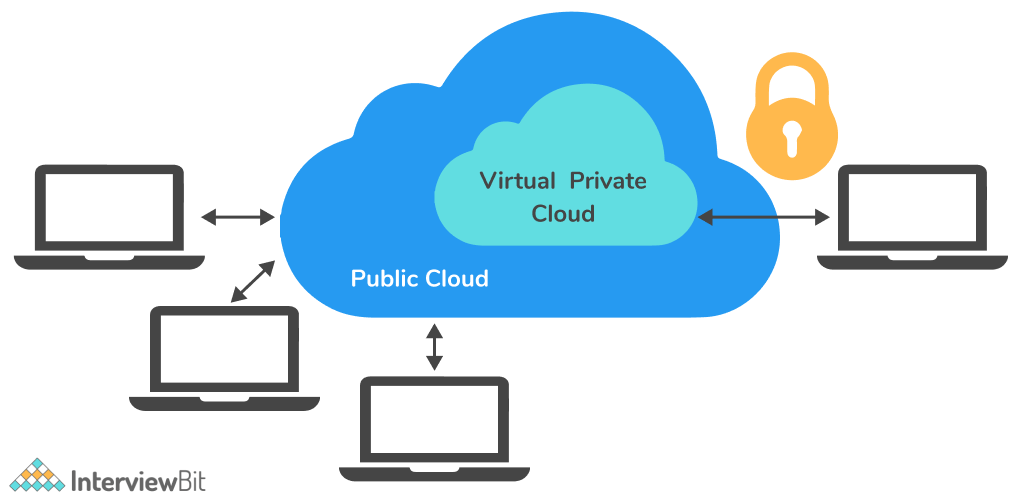
**Ansible :** [Ansible](https://www.interviewbit.com/ansible-interview-questions/) is a remarkably straightforward IT automation technology. Configuration management, application deployment, cloud provisioning, ad-hoc task execution, network automation, and multi-node orchestration are all handled by this software. Complex modifications, such as zero-downtime rolling updates with load balancers, are simple using Ansible.

Following table lists the differences between Ansible and Terraform:



| **Terraform** | **Ansible** |
| --- | --- |
| Terraform is a tool for provisioning. | Ansible is a tool for managing configurations. |
| It uses a declarative Infrastructure as Code methodology. | It takes a procedural method. |
| It's ideal for orchestrating cloud services and building cloud infrastructure from the ground up. | It is mostly used to configure servers with the appropriate software and to update resources that have previously been configured. |
| By default, Terraform does not allow bare metal provisioning. | The provisioning of bare metal servers is supported by Ansible. |
| In terms of packing and templating, it does not provide better support. | It includes complete packaging and templating support. |
| It is strongly influenced by lifecycle or state management. | It doesn't have any kind of lifecycle management. It does not store the state. |

### 23. What do you mean by a Virtual Private Cloud (VPC)? Which command do you use in Terraform to use a VPC service?



A Virtual Private Cloud (VPC) is a private virtual network within AWS where you can store all of your AWS services. It will have gateways, route tables, network access control lists (ACL), subnets, and security groups, and will be a logical data centre in AWS. When you create a service on a public cloud, it is effectively open to the rest of the world and can be vulnerable to internet attacks. You lock your instances down and secure them from outside threats by putting them inside a VPC. The VPC limits the types of traffic, IP addresses, and individuals who have access to your instances.

This stops unauthorised users from accessing your resources and protects you from DDOS assaults. Because not all services require internet connection, they can be safely stored within a private network. You can then only allow particular machines to connect to the internet.

We use the command aws\_vpc to use a VPC Service in Terraform.

### 24. Explain the command terraform fmt in the context of Terraform.

Terraform configuration files are rewritten using the terraform fmt command in a consistent structure and style. This command uses a subset of the Terraform language style conventions, as well as some small readability tweaks. Other Terraform commands that produce Terraform configuration will produce files that follow the terraform fmt style, therefore following this style in your own files will assure consistency. Because formatting selections are always subjective, you may disagree with terraform fmt's choices. This command is purposely opinionated and lacks customization options because its primary goal is to promote stylistic consistency throughout Terraform codebases, even though the chosen style will never be everyone's favourite.

**Syntax:**

terraform fmt [options] DIR

By default, fmt looks for configuration files in the current directory. If the dir option is provided, it will instead scan the specified directory.

The following are the flags that are available:

* **-list=false** - This option doesn't show files with discrepancies in formatting.
* **-write=false** - This option prevents the input files from being overwritten. (When the input is STDIN or -check, this is implied.)
* **-diff** - Shows the differences in formatting modifications.
* **-check** - Verifies that the input is properly formatted. If all input is properly formatted, the exit status will be 0, else it will be non-zero.
* **-recursive** - Process files from subdirectories as well.

### 25. What do you know about Terragrunt? What are its uses?

Terragrunt is a lightweight wrapper that adds extra features for maintaining DRY configurations, dealing with many Terraform modules, and managing remote state.



Following are the use cases of Terragrunt:

* **To Keep Our Background Configuration DRY (Don’t Repeat Yourself):** By setting your backend configuration once in a root location and inheriting that information in all child modules, Terragrunt helps you to keep it DRY ("Don't Repeat Yourself").
* **To Keep Our Provider Configuration DRY:** It might be difficult to unify provider configurations across all of your modules, especially if you wish to alter authentication credentials. You may use Terragrunt to refactor common Terraform code and keep your Terraform modules DRY by using it. The provider configurations can be defined once at a root location, just like the backend configuration.
* **To Keep Our Terraform Command Line Interface arguments DRY:** In the Terraform universe, CLI flags are another typical source of copy/paste. It can be difficult and error-prone to have to remember these -var-file options every time. By declaring your CLI parameters as code in your terragrunt.hcl settings, Terragrunt helps you to keep your CLI arguments DRY.
* **To Promote Terraform modules that are immutable and versioned across environments:** Large modules should be considered hazardous, according to one of the most important lessons we've learnt from building hundreds of thousands of lines of infrastructure code. That is, defining all of your environments (dev, stage, prod, and so on) or even a huge amount of infrastructure (servers, databases, load balancers, DNS, and so on) in a single Terraform module is a Bad Idea. Large modules are slow, insecure, difficult to update, code review, test, and are brittle. Terragrunt lets you define your Terraform code once and then promote a versioned, immutable "artifact" of that code from one environment to the next.

### 26. Explain State File Locking in the context of Terraform.

Terraform's state file locking method prevents conflicts between numerous users doing the same task by blocking activities on a given state file. When one user unlocks the lock, only the other user has access to that state. Terraform will lock your state for any operations that potentially write state if your backend supports it. This prevents outsiders from gaining access to the lock and corrupting your state. All operations that have the potential to write state are automatically locked. There will be no indication that this is happening. Terraform will not continue if state locking fails. The -lock flag can be used to deactivate state locking for most tasks, although it is not advised. Terraform will send a status message if gaining the lock takes longer than planned. If your backend enables state locking, even if Terraform doesn't send a message, it still happens.

### 27. What do you know about Terraform core? What are the primary responsibilities of Terraform core?

Terraform Core is a binary created in the Go programming language that is statically compiled. The compiled binary is the terraform command line tool (CLI), which is the starting point for anyone who wants to use Terraform. The source code can be found at github.com/hashicorp/terraform.

The primary responsibilities of Terraform core includes:

* Reading and interpolating configuration files and modules using infrastructure as code
* Management of the state of resources
* Resource Graph Construction
* Execution of the plan
* Communication with plugins through RPC

### 28. When something goes wrong, how will you control and handle rollbacks in Terraform?

In our Version Control System, we need to recommit the previous code version to make it the new and current one. This would start the terraform run command, which would execute the old code. Because Terraform is more declarative, we will make sure that everything in the code reverts to its previous state. If the state file becomes corrupted, we would use Terraform Enterprise's State Rollback feature to restore the previous state.

### 29. What procedures should be taken to make a high-level object from one module available to the other module?

The steps to make an object from one module available to the other module at a high level are as follows:

* The first step is to define an output variable in a resource configuration. The scope of local and to a module will not be declared until you define resource configuration details.
* Now you must specify the output variable of module A so that it can be utilised in the setup of other modules. You should establish a fresh new and up-to-date key name, with a value that is equal to the output variable of module A.
* You must now create a file named variable.tf for module B. Create an input variable with the exact same name as the key you defined in module B inside this file. This variable permits the resource's dynamic setting in a module. Replicate the process to make this variable available to other modules as well. This is because the scope of the variable established here is limited to module B.

### 30. What do you understand about remote backend in the context of Terraform?

Terraform's remote backend stores terraform state and can also conduct operations in the terraform cloud. terraform commands such as init, plan, apply, destroy , get, output, providers, state (sub-commands: list, mv, pull, push, rm, show), taint, untaint, validate, and many others can be run from a remote backend. It can be used with a single or several remote terraform cloud workspaces. You can utilise terraform cloud's run environment to conduct remote operations like terraform plan or terraform apply.

### 31. How can you prevent Duplicate Resource Error in Terraform?

Depending on the situation and the necessity, it can be accomplished in one of three ways.

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

### 1) What is Terraform? / What do you understand by Terraform?

Terraform is an open-source communication as system software tool created by HashiCorp. It is used to build an infrastructure for building, altering, and versioning transportation safely and professionally. Terraform can manage leading and popular service providers and in-house convention solutions.

### 2) What are the most important features of Terraform?

Following is a list of the most important features of Terraform:

**Infrastructure as Code:** The high-level configuration language of Terraform is used to describe the infrastructure in declarative configuration files that are human-readable. It facilitates us to generate a blueprint that can be edited, shared, and reused.

**Execution Strategies:** Before making any infrastructure modifications, Terraform develops an execution plan to describe what it will do and asks for your agreement. It facilitates us to evaluate the changes before Terraform produces, upgrades or destroys infrastructure.

3) Are callbacks possible with Terraform on Azure?

Callbacks are possible by using the Azure Event Hubs. Terraform's Azure supplier provides effortless functionality to users, and Microsoft Azure Cloud Shell provides an already installed Terraform occurrence.

4) What are the main reasons for choosing Terraform for DevOps?

Following are the main reasons for choosing Terraform for DevOps:

Following are the reasons for choosing Terraform for DevOps:

* Terraform is preferred for DevOps because it can do complete orchestration and not just configuration management such as Ansible and Puppet.
* It provides amazing support to almost all the popular cloud providers like AWS, Azure, GCP, DigitalOcean, etc.
* It can easily manage the configuration of an immutable (dynamic) infrastructure.
* It provides an immutable infrastructure where configuration changes smoothly.
* It works on HCL (HashiCorp Configuration Language), which is very easy to learn and understand.
* It is easily portable from one provider to another.
* The installation of Terraform for DevOps is easy.

5) What do you understand by Terraform init?

Terraform init is a control used to initialize an operational index containing Terraform pattern files. This control is the first command that should be run after writing the new Terraform design and can be looped multiple times.

6) What is the null resource in Terraform?

The null\_resource is used to implement a standard resource library, but no further action is taken. The triggers argument allows an arbitrary set of values that will cause the replacement of resources when changed.

7) What is Terraform D?

Terraform D is a plugin used on most in-service systems and Windows. Terraform init by default searches next directories for plugins.

8) What do you understand by Terraform backend?

The Terraform backend is used to specify how an operation is executed and how the state is loaded. It uses the 'local' backend by default. The abstraction enables remote execution, non-local file state storage, etc.

9) What is ".terraform" directory?

The ".terraform" directory is a local cache where Terraform retains some files required for subsequent operations against this configuration. Its contents are not intended to be included in version control.

10) What are some major competitors of Terraform?

Following are some major competitors of Terraform:

* Packer
* Cloud Foundry
* Ansible
* Kubernetes

11) What is the usage of Terraform init?

Terraform init is a command used to initialize the Terraform code. Let's see the all usage of Terraform init command:

* Terraform init command is used to initialize the working directory containing Terraform configuration files.
* It is used for Plugin Installation.
* It is also used for Child Module Installation.
* It is used for Backend Initialization.
* You can safely run this command multiple times.

12) What do you understand by Terraform in AWS?

In AWS, Terraform is an infrastructure as a code tool. It facilitates us to store our Amazon Web Services infrastructure and produce an update, and it is very similar to AWS Cloud Formation.

### 13) Why is Terraform used for DevOps?

Terraform is preferred to use for DevOps because it facilitates us to manage infrastructure as code. The infrastructure as code is the foundation for DevOps practices such as continuous integration, version control, continuous deployment, and code review.

### 14) What is the full form of IAC in the context of Terraform?

The full form of the term IAC is "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. For example, Terraform is a case tool of IaC.

### 15) What do you understand by Oracle Cloud Infrastructure?

Oracle Cloud Infrastructure is a cloud computing service offered by Oracle Corporation. It provides storage, servers, applications, services, and networks through a global network of managed data centers by Oracle Corporation. These services are provisioned on-demand over the Internet by the company.

16) Is Terraform cloud-agostic? If yes, then what does it mean?

Yes, Terraform is cloud-agostic and allows a single configuration to manage multiple providers and handle cross-cloud dependencies. It is also used to simplify management and orchestration, facilitates operators to build large-scale multi-cloud infrastructures.

17) What do you understand by Terraform providers?

Providers are plugins on that Terraform relies to interact with cloud providers, SaaS providers, and other APIs. The Terraform configurations have to declare which providers they require so that Terraform can install and use them. Some Terraform providers require configuration such as endpoint URLs or cloud regions before using them.

18) What is the use of Terraform provider?

Terraform providers add a set of resource types and data sources that Terraform can manage.

* A provider implements every resource type; without providers, Terraform can't manage any infrastructure.
* Terraform providers configure a specific infrastructure platform (either cloud or self-hosted).
* Terraform providers can also offer local utilities for tasks such as generating random numbers for unique resource names.

19) What do you understand by Terraform Backends? What are the most recommended Backends we should use?

Terraform backends are used to define where and how operations are performed, where state snapshots are stored, etc. Each Terraform configuration can specify a backend.

Following are the two types of most recommended Terraform Backends:

* If the new users are still learning how to use Terraform, the default "local" backend is most recommended, which requires no configuration.
* For the expert users, if they or their team are using Terraform to manage meaningful infrastructure, the most recommended backend is the "remote" backend with Terraform Cloud or Terraform Enterprise.

20) What are the biggest competitors of Terraform?

The following list specifies some of the biggest competitors of Terraform:

* Ansible
* Kubernetes
* Turbonomic
* CloudBolt
* Azure Management Tools
* Morpheus
* CloudHealth
* Apptio Cloudability
* Platform9 Managed Kubernetes etc.

21) What is the working of Terraform core?

The Terraform core is used to look after the configuration monitoring and create analysis and evaluation based on the configuration. It is also used to keep track and compare the current and previous versions and display the output through the terminal.

**Terraform core mainly takes two inputs:**

* The first one is Terraform Configuration, used to keep track of the infrastructure detail.
* The second one is Terraform state that is used to keep track of the infrastructure status.

22) What is the usage of provisioners in Terraform?

In Terraform, provisioners are used to preparing servers or other infrastructure objects for service. They are used on the local machine as well as on the remote machines.