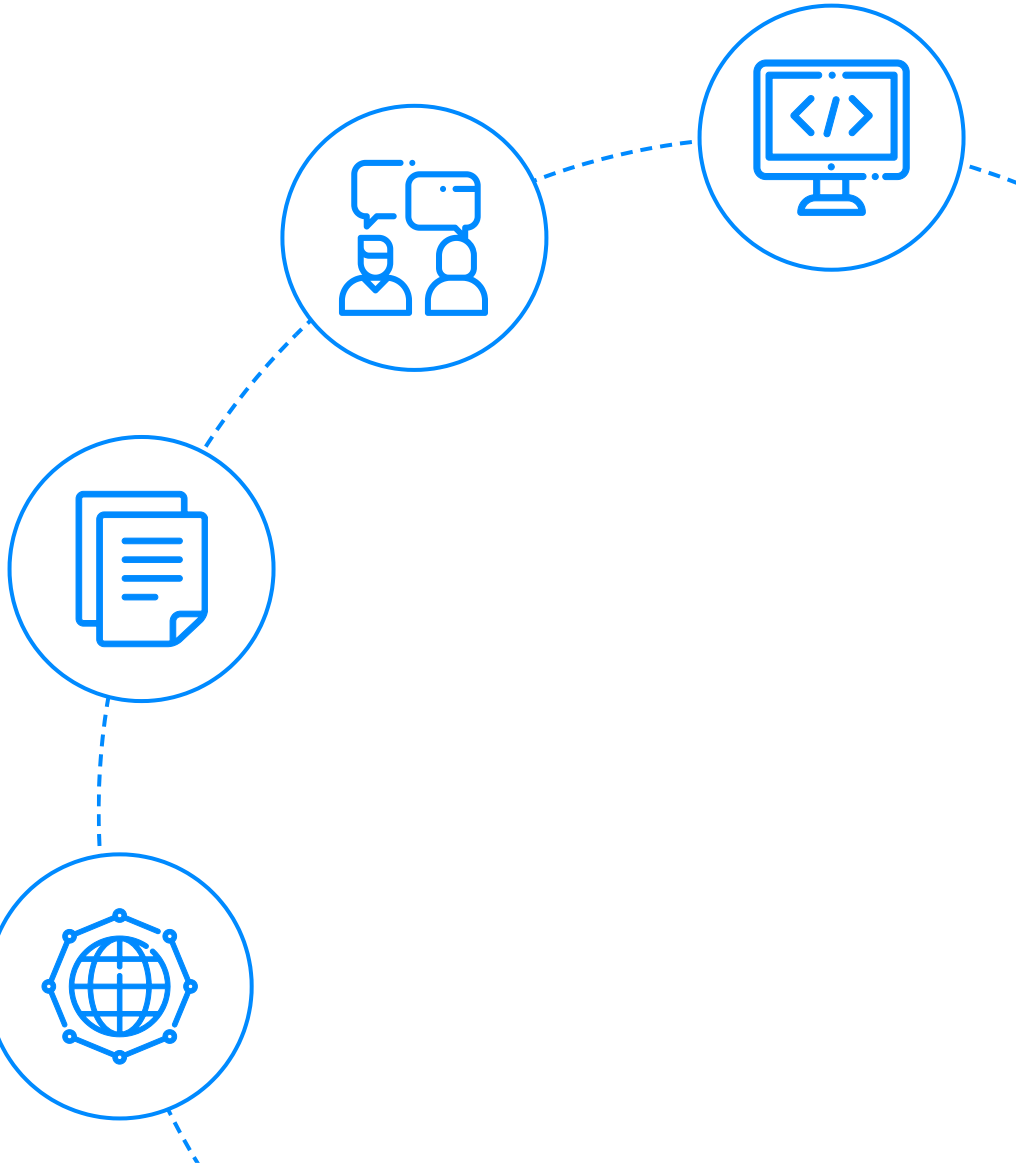
InterviewBit

Terraform Interview Questions



To view the live version of the page, click here.

Contents

Terraform Interview Questions for Freshers

Terraform Interview Questions for Experienced

Terraform Interview Questions for Experienced

(.....Continued)

19. Explain the architecture of Terraform request flow.
20. Differentiate between Terraform and Cloudformation.
21. Explain the command terraform taint in the context of Terraform.
22. Differentiate between Terraform and Ansible.
23. What do you mean by a Virtual Private Cloud (VPC)? Which command do you use in Terraform to use a VPC service?
24. Explain the command terraform fmt in the context of Terraform.
25. What do you know about Terragrunt? What are its uses?
26. Explain State File Locking in the context of Terraform.
27. What do you know about Terraform core? What are the primary responsibilities of Terraform core?
28. When something goes wrong, how will you control and handle rollbacks in Terraform?
29. What procedures should be taken to make a high-level object from one module available to the other module?
30. What do you understand about remote backend in the context of Terraform?
31. How can you prevent Duplicate Resource Error in Terraform?

Let's get Started

What is Terraform

Terraform is an infrastructure as code (IaC) tool that lets you build, edit, and version infrastructure in a secure and efficient manner. This covers both low-level and high-level components, such as compute instances, memory, and networking, as well as DNS records, SaaS services, and so on. Terraform is capable of managing both third-party services and unique in-house solutions. Terraform uses configuration files to tell it which components are needed to run a single application or a whole datacenter. Amazon Web Services, IBM Cloud, Google Cloud, Platform, DigitalOcean, Linode, Microsoft Azure, Oracle Cloud Infrastructure, OVH, VMware, vSphere, OpenNebula, and OpenStack all use Terraform.

In this article, we will cover the most frequently asked interview questions on Terraform. So, let's get started.

Terraform Interview Questions for Freshers

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

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:

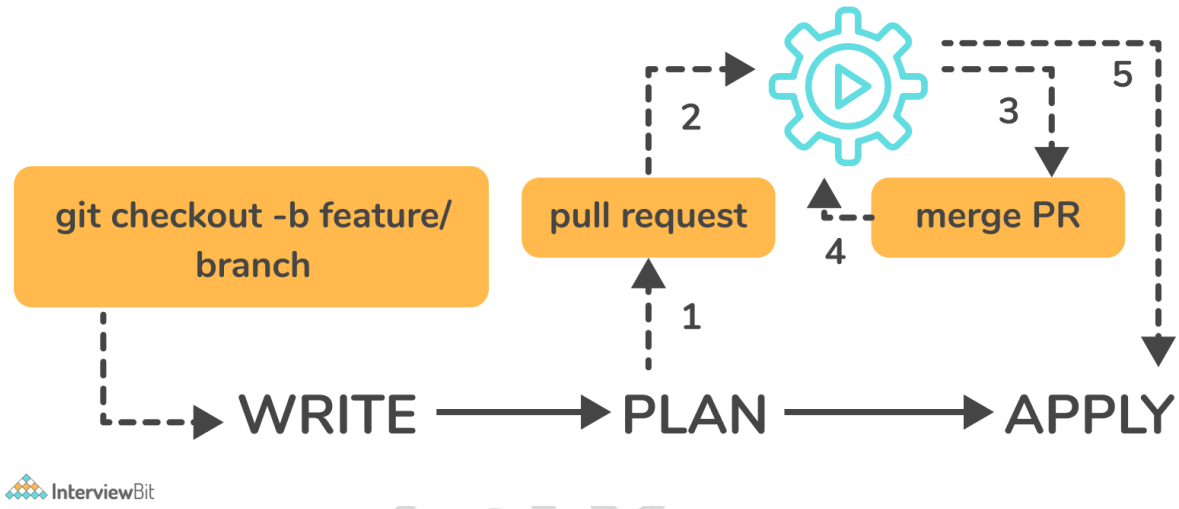
- GitHub
- GitLab CE
- GitLab EE
- Bucket Cloud

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:

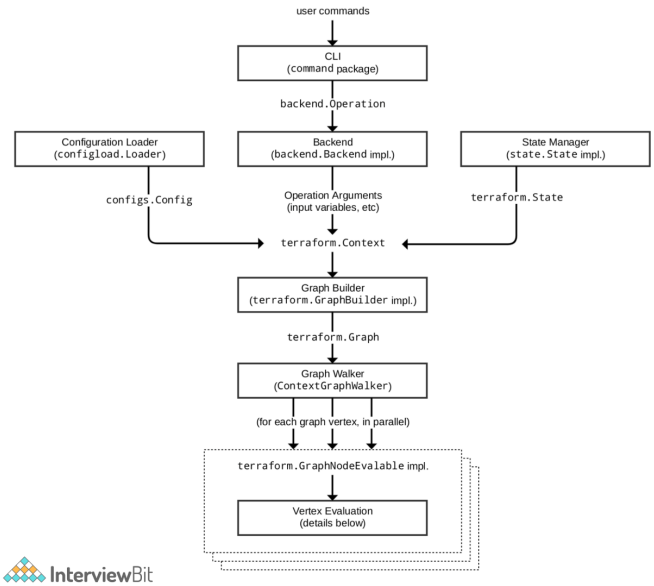


- **Write** - Create infrastructure in the form of code.
- **Plan** - Plan ahead of time to see how the changes will look before they are implemented.
- **Apply** - Create a repeatable infrastructure.

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:

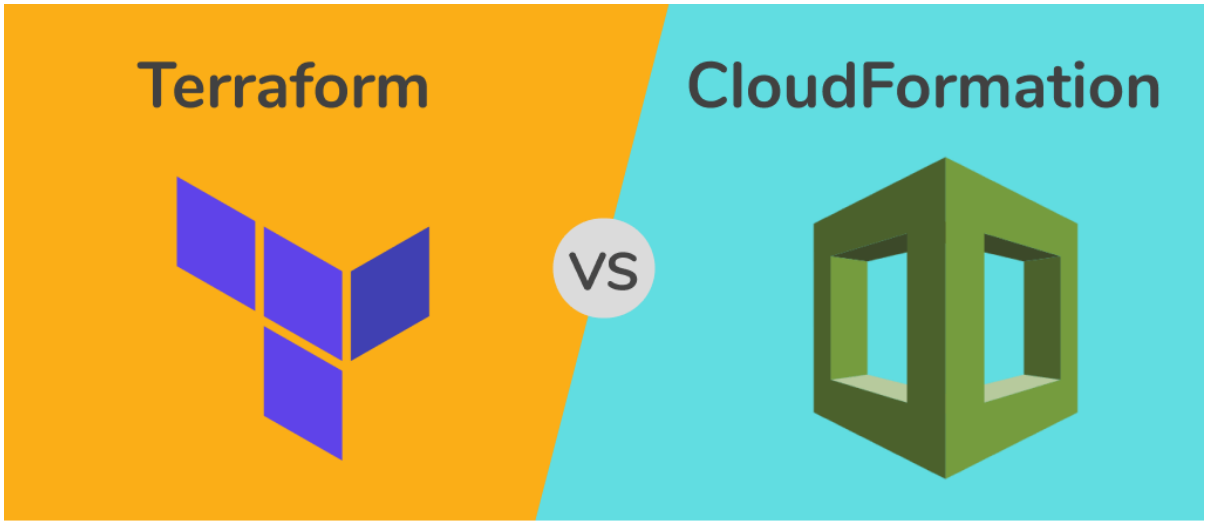
The graph walking method visits each vertex of the graph in a fashion that respects the graph's "happens after" edges. Each vertex in the graph is assessed in such a way that the "happens after" edges are taken into account. The graph walk method will assess many vertices at the same time if possible.

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.

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
```

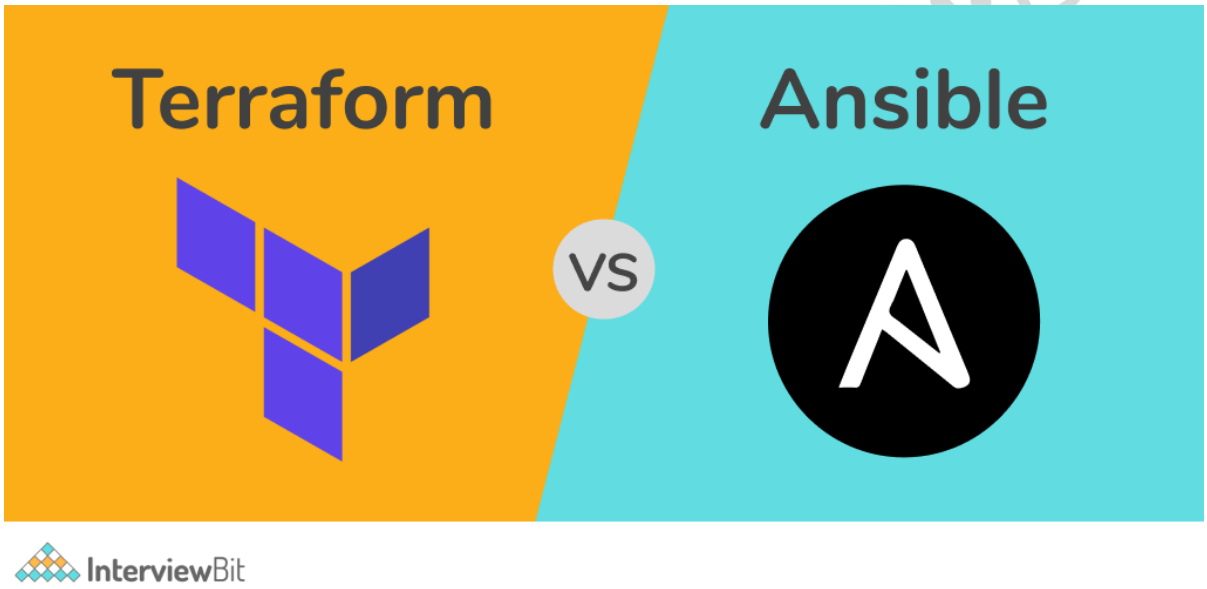
The address option specifies the location of the infected resource. The following options are available with this command:

- **-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.

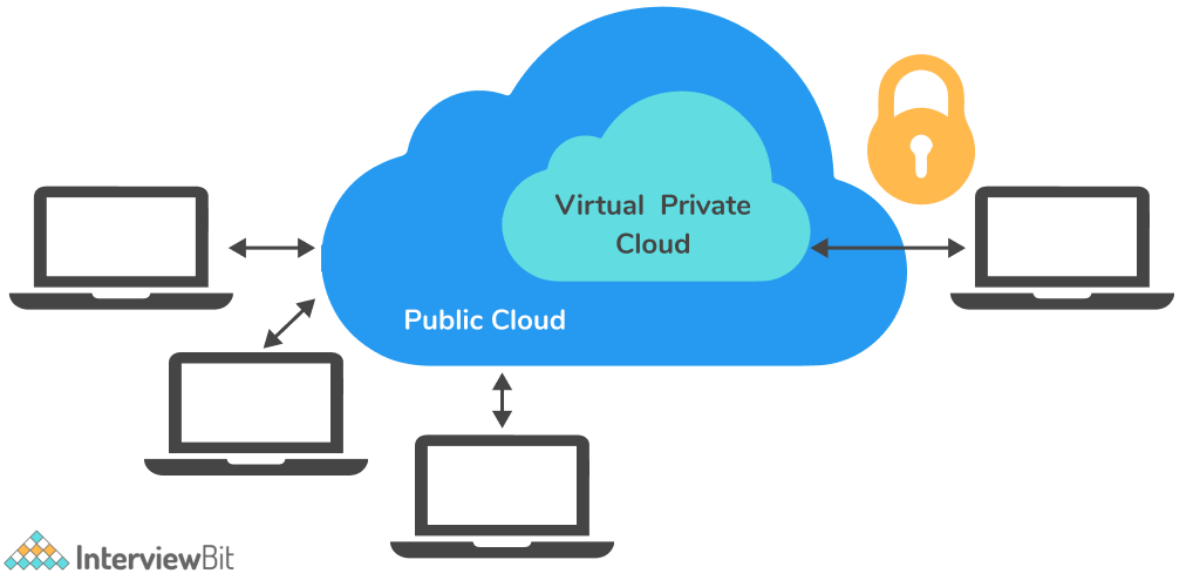
Ansible : Ansible is a remarkably straightforward IT automation technology. Configuration management, application deployment, cloud provisioning, ad-hoc task execution, network automation, and multi-node orchestration are all handled by this software. Complex modifications, such as zero-downtime rolling 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.

Useful Resources

Kubernetes

AWS

Azure

Terraform Tutorial

Terraform Download

Links to More Interview Questions

C Interview Questions	Php Interview Questions	C Sharp Interview Questions
Web Api Interview Questions	Hibernate Interview Questions	Node Js Interview Questions
Cpp Interview Questions	Oops Interview Questions	Devops Interview Questions
Machine Learning Interview Questions	Docker Interview Questions	Mysql Interview Questions
Css Interview Questions	Laravel Interview Questions	Asp Net Interview Questions
Django Interview Questions	Dot Net Interview Questions	Kubernetes Interview Questions
Operating System Interview Questions	React Native Interview Questions	Aws Interview Questions
Git Interview Questions	Java 8 Interview Questions	Mongodb Interview Questions
Dbms Interview Questions	Spring Boot Interview Questions	Power Bi Interview Questions
Pl Sql Interview Questions	Tableau Interview Questions	Linux Interview Questions
Ansible Interview Questions	Java Interview Questions	Jenkins Interview Questions