Top 60+ Terraform Interview Questions May 2023

June 7, 2023 by Bhanvendra Gaur Leave a Comment

Top Terraform Questions | Intermediate Level Terraform Interview Questions and answers | Terraform Advanced Interview Questions

If you're looking to become a Terraform expert, then you need to be prepared for some tough interview questions. Here are the top 60+ Terraform Interview Questions and Answers that will help you land your dream job! They are mostly asked in terraform-related interviews about the role of a DevOps engineer.

Why Terraform?

Terraform is a tool to build an infrastructure safely and efficiently. Terraform can manage leading and popular service providers as well as custom in-house solutions.

What is Terraform? | Complete Beginners Guide

The Configuration file in Terraform describes the components needed to run a single application or your entire data centre. Terraform then generates an execution plan describing what it will do to reach the desired state, and then executes it to build the described infrastructure. It creates incremental execution plans which can be applied according to the configuration change.

Top Terraform Interview Questions and Answers

Question 1: What do you mean by Terraform?

Answer: 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.

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, 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.

Get the FREE Terraform Interview Questions & Answers Guide for download.

Question 3: Define Terraform init?

Answer: Terraform initialises the code with the command terraform init. 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:

Installing Plugins

Installation of a Child Module

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, 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 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 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.

Kaffeine

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 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.

Intermediate Terraform Interview Questions

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

Terraform Workflow

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 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.

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

Advanced Terraform Interview Questions

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.

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:

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.

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.

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 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

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.

core_terraform_workflow

Question 34: Explain the architecture of Terraform request flow.

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

Terraform_request_flow

Command Line Interface (CLI):

CLI (Common Language Interface) (command package)

Except for some early bootstrapping in the root package, when a user launches the terraform programme, execution immediately jumps into one of the command package's "command" implementations (not shown in the diagram). The commands store the mapping between user-facing command names and their corresponding command package types. The go file is located in the repository's root directory.

The command implementation's responsibility for these commands is to read and parse any command line arguments, command-line options, and environment variables required for the specified command and use them to construct a backend. object of operation The operation is then passed to the currently selected backend.

Backends:

A backend in Terraform is responsible for a number of things:

Execute operations (e.g. plan, apply)

Variables defined in the workspace can be saved.

to store the current 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 used directly, or an implementation provided by whatever backend is being wrapped), and then uses the config loader to load and perform initial processing/validation of the configuration specified in the operation. It then creates a terraform.context object using these parameters as well as the other parameters supplied in the process. The main object performs terraform operations.

Configuration Loader :

Model types represent the top-level configuration structure in package configs. Config represents a configuration (the root module and all of its child modules). Although the configs package contains some low-level functionality for creating configuration objects, the configload is the primary entry point. Loader can be found in the configload subpackage. A loader handles all of the complexities associated with installing child modules (during terraform init) and then locating those modules when a configuration is loaded by a backend. It takes the path to the root module and loads all of the child modules recursively to produce a single configuration.

State Manager:

The state manager is in charge of storing and retrieving snapshots of a workspace's Terraform state. Each manager implements a subset of the interfaces provided by the statemgr package, with the majority of managers covering the entire set of statemgr. Complete operation. The smaller interfaces are commonly used in other function signatures to specify what actions the function may take on the state manager; there is no reason to create a state manager that does not implement all of statemgr. Full.

Graph Builder:

The terrain. The Context method calls a graph builder. A graph builder is used to represent the fundamental phases of that action, as well as the dependencies between them. Because of the differences in the graph-building process, each operation has its own graph builder. For a "plan" operation, a graph must be constructed directly from the configuration, whereas a "apply" action

constructs its graph from the set of alterations mentioned in the plan being applied.

Graph Walk:

The graph walking method explores each vertex of the graph while keeping the graph's "happens after" edges in mind. Every vertex in the graph is evaluated so that the "happens after" edges are taken into account. The graph walk algorithm will evaluate multiple vertices at once if possible.

Vertex Evaluation:

Execution refers to the action taken for each vertex during a graph walk. Execution performs a set of random operations that are appropriate for the vertex type in question. Before the graph walk can begin evaluating other vertices with "happens after" edges, a vertex must be correctly completed. When one or more errors occur during evaluation, the graph walk is paused, and the errors are returned to the user. questions for terraform interviews

Question 35: Differentiate between Terraform and Cloudformation.

Answer: The following points highlight the differences between Terraform and Cloudformation :AWS Infrastructure As Code: CloudFormation vs Terraform | by Jackie Tung | Medium

User-friendliness: Terraform works with a variety of Cloud Service Providers, including AWS, Azure, Google Cloud Platform, and others, whereas CloudFormation only works with AWS services. Terraform covers the vast majority of AWS resources.

Depending on the language: CloudFormation supports JSON and YAML. CloudFormation is now simple to grasp and apply. AWS developers, on the other hand, are not permitted to create CloudFormation templates larger than 51MB. If the size of a template exceeds this limit, the developers must create a layered stack for it.

Terraform, on the other hand, makes use of Hashicorp's own HCL programming language (Hashicorp Configuration Language). This language is also JSON-compatible.

State-management:

Because CloudFormation is an AWS managed service, it inspects the infrastructure on a regular basis to ensure that it is in good working order. If anything changes, CloudFormation receives a detailed

response.

Terraform, on the other hand, stores the state of the infrastructure on the provisioning machine, which can be a virtual machine or a remote computer. Terraform defines the resources it maintains using the state as a map, which is saved as a JSON file.

To summarise, CloudFormation manages Cloudformation's state by default, preventing conflicting changes. Terraform saves the state to a local disc, making state synchronisation easier. Terraform states can also be saved in storage services such as S3, which is a recommended additional state management strategy. This must be defined on the backend to facilitate and secure management.

Cost:

The best part is that both of these programmes are completely free. Both of these technologies have sizable online communities that provide a wealth of information and examples. Cloudformation is completely free. Customers only need to pay for the AWS service provided by CloudFormation. Terraform is an open-source application that can be used for free. Terraform, on the other hand, has a paid enterprise version that includes additional collaboration and governance features.

Integration of Multiple Clouds:

Terraform is the way to go if you want to provide services across multiple cloud platforms. While Terraform can be used with AWS, GCP, Azure, and other cloud providers, CloudFormation is only available on AWS. Cloudformation is not for you if you have multiple cloud installations. If you use AWS resources such as EC2, S3, and so on, you should use Cloudformation.

Question 36: Differentiate between Terraform and Ansible.

Answer: Ansible is a deceptively simple IT automation tool. Configuration management, application deployment, cloud provisioning, ad-hoc job execution, network automation, and multi-node orchestration are all handled by this software. Ansible simplifies complex changes such as zero-downtime rolling updates with load balancers. The following table compares and contrasts 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.

For a detailed comparison between these two giants in the market please check our blog Terraform vs Ansible: Working, Difference, Provisioning

Question 37: What are the most useful Terraform commands ?

Answer: Here are some useful Terraform Commands

fmt

init

validate

plan

apply

destroy

output

show

state

version

Question38: Are callbacks possible with Terraform on Azure ?

Answer: Yes. This is possible with Azure Event Hubs.

Question 39: What is Terraform Directory?

Answer: Terraform Directory, which Terraform uses to manage cached provider plugins and modules, as well as to record which workspace is currently active and the last known backend configuration in case state needs to be migrated on the next run.

Question 40: Is history the same as it is on the web while using TFS API to provide resources ?

Answer: Yes, the narration is similar to that found on the web because UI uses API as its foundation. Everything on the UI is available via other methods and the API.

Question 41: What is a Private Module Registry?

Answer: Using the private module registry, Terraform Cloud users can create and confidentially share infrastructure modules within an organisation. The private module registry in Terraform Enterprise allows you to share modules within or across organisations.

Question 42: Does Terraform support multi-provider deployments?

Answer: Terraform is a powerful tool in multi-provider deployments because it is not tied to a specific infrastructure or cloud provider. You can manage all resources with the same set of configuration files, sharing variables and defining dependencies across providers.

Read

Question 43: How is duplicate resource error ignored during terraform apply ?

Answer: You can:

To stop managing those resources, remove them from your Terraform code.

Remove the resources from the API (cloud provider) and recreate them using Terraform.

Terraform those resources and remove the terraform code that is attempting to recreate them.

Use terraform apply —target=xxx to apply only the resources you require.

Question 44: What are Provisioners in Terraform?

Answer: Provisioners are used to execute scripts on a local or remote machine as part of resource creation or destruction. Provisioners can be used to bootstrap a resource, cleanup before destroy, run configuration management, etc.

Question 45: What are some of the built-in provisioners available in Terraform?

Answer: Some of the built-in provisioners available in Terraform are:

abspath.

dirname.

pathexpand.

basename.

file.

fileexists.

fileset.

filebase64.

Question 46: Tell us about some notable Terraform applications.

Answer: 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

Question 47: What are the components of Terraform architecture?

Answer: 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

Question 48: Define Resource Graph in Terraform.

Answer: A resource graph is a graphical representation of the available resources. It enables the modification and creation of independent resources at the same time. Terraform creates a plan for the graph's configuration in order to generate plans and refresh the state. It efficiently and effectively creates structure to help us understand the disadvantages.

Question 49: Can you provide a few examples where we can use for Sentinel policies?

Answer: Sentinels are an effective way to implement a wide range of policies in Terraform. Here are a couple of examples:

Enforce explicit resource ownership.

Limit the roles that the cloud provider can play.

Examine the audit trail for Terraform Cloud operations.

Only certain resources, providers, or data sources may be prohibited.

Make resource tagging mandatory.

In the Private Module Registry, you can limit how modules are used.

Question 50: What are the various levels of Sentinel enforcement?

Answer: Sentinel has three levels of enforcement: advisory, soft mandatory, and hard mandatory.

Advisory – Logged in but permitted to pass. When a user initiates a plan that violates the policy, an advisory is issued.

Soft Mandatory – Unless an override is specified, the policy must be followed. Overrides are only available to administrators.

Hard Mandatory – The policy must be implemented regardless. Unless and until this policy is removed, it cannot be overridden. Terraform's default enforcement level is this.

Question 51: How to Store Sensitive Data in Terraform?

Answer: To communicate with your cloud provider's API, Terraform requires credentials. However, these credentials are frequently saved in plaintext on your desktop. Every day, GitHub is exposed to thousands of API and cryptographic keys. As a result, your API keys should never be directly stored in Terraform code. To store passwords, TLS certificates, SSH keys, and anything else that shouldn't be stored in plain text, use encrypted storage.

Question 52: What is Terraform Core? Tell us some primary responsibilities of it

Answer: Terraform Core is a binary written in the Go programming language and statically compiled. The compiled binary provides Terraform users with an entry point. The primary responsibilities are as follows:

Infrastructure's code functionalities include module and configuration file reading and interpolation.

Building a Resource Graph

RPC-based plugin communication

Plan implementation

Resource state management

Read: Automate AWS Virtual Machine using Terraform – Creation Demo

Question 53: How will you upgrade plugins on Terraform?

Answer: Terraform providers are distributed separately from the Terraform binary since Terraform v0.10. This allows them to update at different rates while also allowing a larger group of people to collaborate on the providers. This is mostly positive, but it adds a new step for upgrading providers.

upgrade plugins

Question 54: How will you control and handle rollbacks when something goes wrong?

Answer: We will recommit the previous version of the code to my VCS as the new and current version. A terraform run will be triggered, which will be in charge of running the old code. Remember that terraform is more declarative. Check that the old code contains everything that was specified in the code for rollback.

Ensure that it is not destroyed when the old code is run due to a lack of these. If the state file becomes corrupted as a result of a recent Terraform run, I will use Terraform Enterprise's State Rollback feature to roll back to the most recent good state. Because every state change is versioned, this could be done.

Question 55: How can you define dependencies in Terraform?

Answer: You can use depends_on to declare the dependency explicitly. You can also specify multiple resources in the depends on argument, and Terraform will create the target resource after all of them have been created.

Question 56: What is the external data block in Terraform?

Answer: The external data source allows an external programme to act as a data source by exposing arbitrary data for use elsewhere in the Terraform configuration by implementing a specific protocol (defined below).

Question 57: What happens when multiple engineers start deploying infrastructure using the same state file?

Answer: Terraform has a critical feature known as "state locking." This feature ensures that no changes to the state file are made during a run, preventing the state file from becoming corrupt. It is important to note that the state locking feature is not supported by all Terraform Backends. If this feature is required, you should select the appropriate backend.

Read: Terraform Variables – Terraform Variable Types

Question 58: Which value of the TF_LOG variable provides the MOST verbose logging?

Answer: TRACE is the most verbose option, and it is the default if TF_LOG is not set to a log level name. When logging is enabled, you can set TF_LOG_PATH to force the log to always be appended to a specific file.

Question 59: Which command can be used to preview the terraform execution plan?

Answer: The terraform plan command generates an execution plan, which allows you to preview the changes that Terraform intends to make to your infrastructure. When Terraform generates a plan by default, it:

Reads the current state of any existing remote objects to ensure the Terraform state is current.

The current configuration is compared to the previous state, and any differences are noted.

Proposes a set of change actions that, if executed, should cause the remote objects to match the configuration.

Question 60: Which command can be used to reconcile the Terraform state with the actual real-world infrastructure?

Answer: Terraform aids in the detection and management of drift. The state file stores information about the real-world state of Terraform-managed infrastructure. The command terraform refresh refreshes this state file, reconciling what Terraform believes is running and its configuration with what is actually running and configured.

Read: Terraform Providers Overview

Question 61: What is the benefit of Terraform State? What is the benefit of using modules in terraform?

Answer: Terraform state is primarily used to store bindings between remote system items and resource instances specified in your configuration. When Terraform generates a remote object in response to a configuration change, it saves the remote object's identification to a specific resource instance and may update or remove that object in response to future configuration changes.

We can save time and avoid costly errors by reusing configuration created by you, other members of your team, or other Terraform experts who have published modules for you to use.

Some other important terraform commands for technical terraform interview Questions

terraform init: In order to prepare the working directory for use with Terraform, the terraform init command performs Backend Initialization, Child Module Installation, and Plugin Installation.

terraform apply: The terraform apply command executes the actions proposed in a Terraform plan

terraform apply –auto-approve: Skips interactive approval of plan before applying.

terraform destroy: The terraform destroy command is a convenient way to destroy all remote objects managed by a particular Terraform configuration.

terraform fmt: The terraform fmt command is used to rewrite Terraform configuration files to a canonical format and style

terraform show: The terraform show command is used to provide human-readable output from a state or plan file.

Conclusion

Top 40 Terraform Interview Questions and Answers for 2023

By Simplilearn

Last updated on Jun 1, 2023212350

Most Frequently Asked Terraform Interview Questions and Answers for 2023

Table of Contents

In recent times, there has been a rise in the adoption of cloud computing which has opened doors to many new opportunities. The sheer diversity and volume of jobs in cloud computing have made it favorable for many aspirants. Why is Terraform so popular? This is because every aspect of the cloud computing landscape is significant to all IT professional jobs. The emergence of DevOps is responsible for the popularity of tools like Terraform. To go ahead in your career, you will have to face terraform interview questions. We have compiled a list of most frequently asked terraform interview questions, and these questions cover all type of levels:

Basic level terraform interview questions

Intermediate level terraform interview questions

Advanced level terraform interview questions

Top Terraform Interview Questions and Answers

Here are some basic level terraform interview questions.

1. What do you understand by Terraform in AWS?

Terraform is a part of the AWS DevOps Competency and also an AWS Partner Network (APN) advanced technology partner. It is similar to AWS Cloud Formation 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 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.

What Do Tech Companies Want in Cloud Experts?

Post-Graduate Program in Cloud ComputingEXPLORE PROGRAMWhat Do Tech Companies Want in Cloud Experts?

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 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 to define and enforce infrastructure configurations across multiple clouds and on-premises data centers.

Get Certifications in AWS, Azure and Google Cloud

Cloud Architect Master's ProgramEXPLORE PROGRAMGet Certifications in AWS, Azure and Google Cloud

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 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 are Azure Management Tools, Morpheus, CloudHealth, Turbonomic, and CloudBolt.

Next up, let us see some intermediate terraform interview questions!

Intermediate Terraform Interview Questions

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

21. 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

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 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

36. Give the terraform configuration for creating a single EC2 instance on AWS.

This is the Terraform configuration for creating a single EC2 instance on AWS:

```
provider “aws” {

region = “”}

resource “aws_instance”

“example” {

ami = ""

instance_type = ""

tags {

  Name = "example"}
```

37. How will you upgrade plugins on Terraform?

Run ‘terraform init’ with ‘-upgrade’ option. This command rechecks the releases.hashicorp.com to find new acceptable provider versions. It also downloads available provider versions. “.terraform/plugins/<OS>_<ARCH>” is the automatic downloads directory.

38. How will you make an object of one module available for the other module at a high level?

Ab output variable is defined in resource configuration.

Declare the output variable of module_A.

Create a file variable.tf for module B.

Establish the input variable inside this file having the same name as the key defined in module_B.

Replicate the process for making variable available to other modules

39. What are some of the latest Terraform Azure Provider factors?

The latest versions involve new data resources and Azurem_batch_certificate, which helps in managing the certificate. This resource is used for controlling the prefix in networking. There is fixing of bugs, and azurerm_app_service has also been enhanced.

40. How will you control and handle rollbacks when something goes wrong?

I need to recommit the previous code version to be the new and current version in my VCS. This would trigger as terraform run, which would be responsible for running the old code. As Terraform is more declarative, I will make sure all things in the code roll back to the old code. I would use the State Rollback Feature of Terraform Enterprise to roll back to the latest state if the state file got corrupted.

Conclusion

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 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.

You can download a PDF version of Terraform Interview Questions.

Download PDF

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 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.