Question 1: What do you mean by Terraform?

Answer: Terraform is open-source communication as a system software tool created by [HashiCorp](https://k21academy.com/cloud-infrastructure-automation-certification-terraform-associate-self-study-training/). It is an instrument for building, altering, and versioning transportation safely and professionally. Terraform can direct existing and accepted service providers as well as convention in-house solutions.

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

Answer: Below are the reasons for choosing Terraform for DevOps:

* It can do complete orchestration and not just configuration management (like Ansible and Puppet).
* Has amazing support of almost all the popular cloud providers like AWS, Azure, [GCP](https://k21academy.com/google-cloud/introduction-to-google-cloud-platform/), DigitalOcean etc.
* Easily manages the configuration of an immutable (dynamic) infrastructure.
* Provide immutable infrastructure where configuration changes smoothly.
* Works on HCL (HashiCorp configuration language), which is very easy to learn and understand.
* Easily portable from one provider to another.
* Easy Installation.

Question 3: Define Terraform init?

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

You can use the init command for:

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

Question 4: Name some major competitors of Terraform?

Answer: Some of them are:

* Packer, tgjob, bladelogic
* Cloud Foundry
* Ansible
* Kubernetes

Question 5: Define Terraform provider? > 2700+ provider

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

Question 6: How does Terraform work?

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

Question 7: Name some major features of Terraform?

Answer: Some of them are:

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

Question 8: Define IAC?

Answer: IaC is a short form to the term “Infrastructure as Code”. IaC refers to a scheme whereby developers can run and provision the computer data center’s mechanically instead of getting into a physical process. [Terraform IAC](https://k21academy.com/terraform-iac/day1-training/), for example, is a case tool of IAC.

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

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

Question 10: Describe the working of Terraform core?

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

Terraform core mainly takes two inputs:

* Terraform Configuration – It keeps track of the infrastructure detail
* Terraform state – It keeps track of the infrastructure status.

Question 11:  What are the key features of Terraform?

Answer: Following are the key features of Terraform:

* Infrastructure as Code: Terraform’s high-level configuration language is used to define your infrastructure in human-readable declarative configuration files.
* You may now create an editable, shareable, and reusable blueprint.
* Terraform generates an execution plan that specifies what it will do and asks for your approval before making any infrastructure alterations. You can assess the modifications before Terraform creates, updates, or destroys infrastructure.
* [Terraform](https://k21academy.com/cloud-infrastructure-automation-certification-terraform-associate-self-study-training/) creates a resource graph while simultaneously developing or altering non-dependent resources. Terraform can now build resources as quickly as possible while also giving you more information about your infrastructure.
* Terraform’s the automation of change allows you to apply complex changesets to your infrastructure with little to no human interaction. Terraform recognises

Question 12: What are the use cases of Terraform?

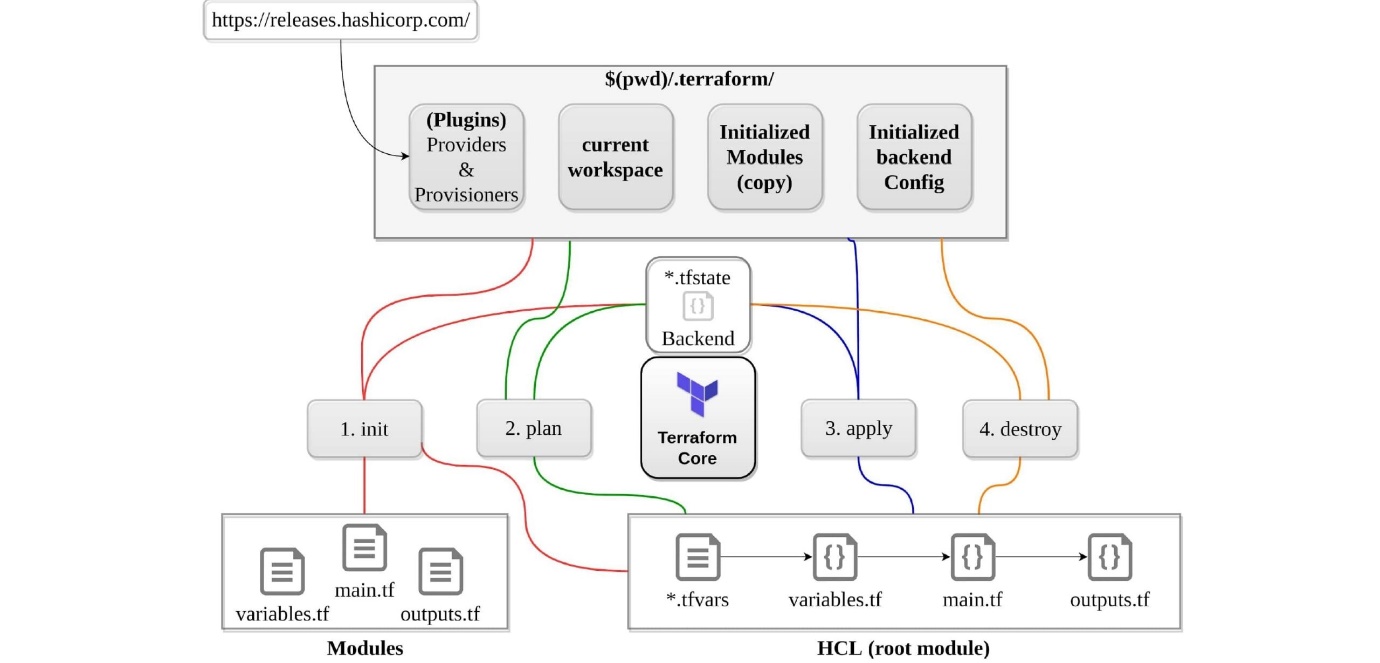
 Following are the use cases of Terraform:

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

Question 13: What are the most useful Terraform commands?

Common commands:

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



Question 14: How does Terraform help in discovering plugins?

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

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

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

Question 16: Define Modules in Terraform?

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

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

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

Question 18: What do you mean by Terraform cloud?

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

Question 19: Define null resource in Terraform?

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

The null resource’s primary application is as a do-nothing container for arbitrary actions performed by a [provisioner](https://k21academy.com/terraform-iac/terraform-provisioners/).

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

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

Question 21 : What does the following command do?

Answer:

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

Question 22: List all the Terraform-supported versions

Answer:

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

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

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

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

Answer: Version control tools supported by Terraform are:

* GitHub
* GitLab CE
* GitLab EE
* Bucket Cloud

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

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

Question 26: What do you mean by Terragrunt, list some of its use cases?

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

Use cases:

* Keep your Terraform code DRY
* Maintain a DRY remote state configuration.
* Keep your CLI flags DRY
* Run Terraform commands on multiple modules at the same time.
* Use multiple [AWS accounts](https://k21academy.com/amazon-web-services/aws-solutions-architect/create-aws-free-tier-account/).

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

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

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

Question 28: What is State File Locking?

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

Question 29: What is a Remote Backend in Terraform?

Answer: [Terraform](https://k21academy.com/cloud-infrastructure-automation-certification-terraform-associate-self-study-training/) remote backend is used to store Terraform’s state and can also run operations in Terraform Cloud. Multiple terraform commands such as init, plan, apply, destroy (terraform version >= v0.11.12), get, output, providers, state (sub-commands: list, mv, pull, push, rm, show), taint, untaint, validate, and many more are available via remote backend. It is compatible with a single remote Terraform cloud workspace or multiple workspaces. You can use terraform cloud’s run environment to run remote operations such as terraform plan or terraform apply.

Question 30: What is a Tainted Resource?

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

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

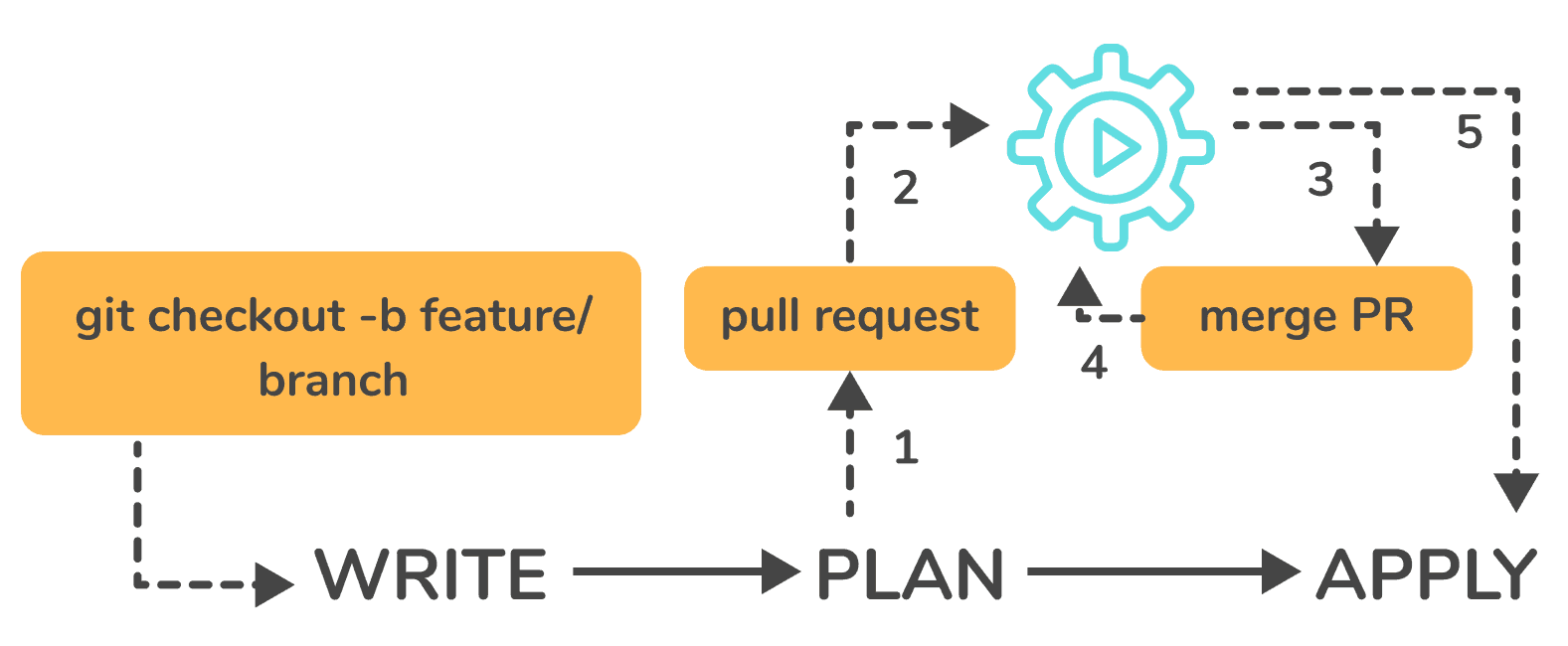
Answer: It can be done in three ways depending on the situation and the requirement

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.

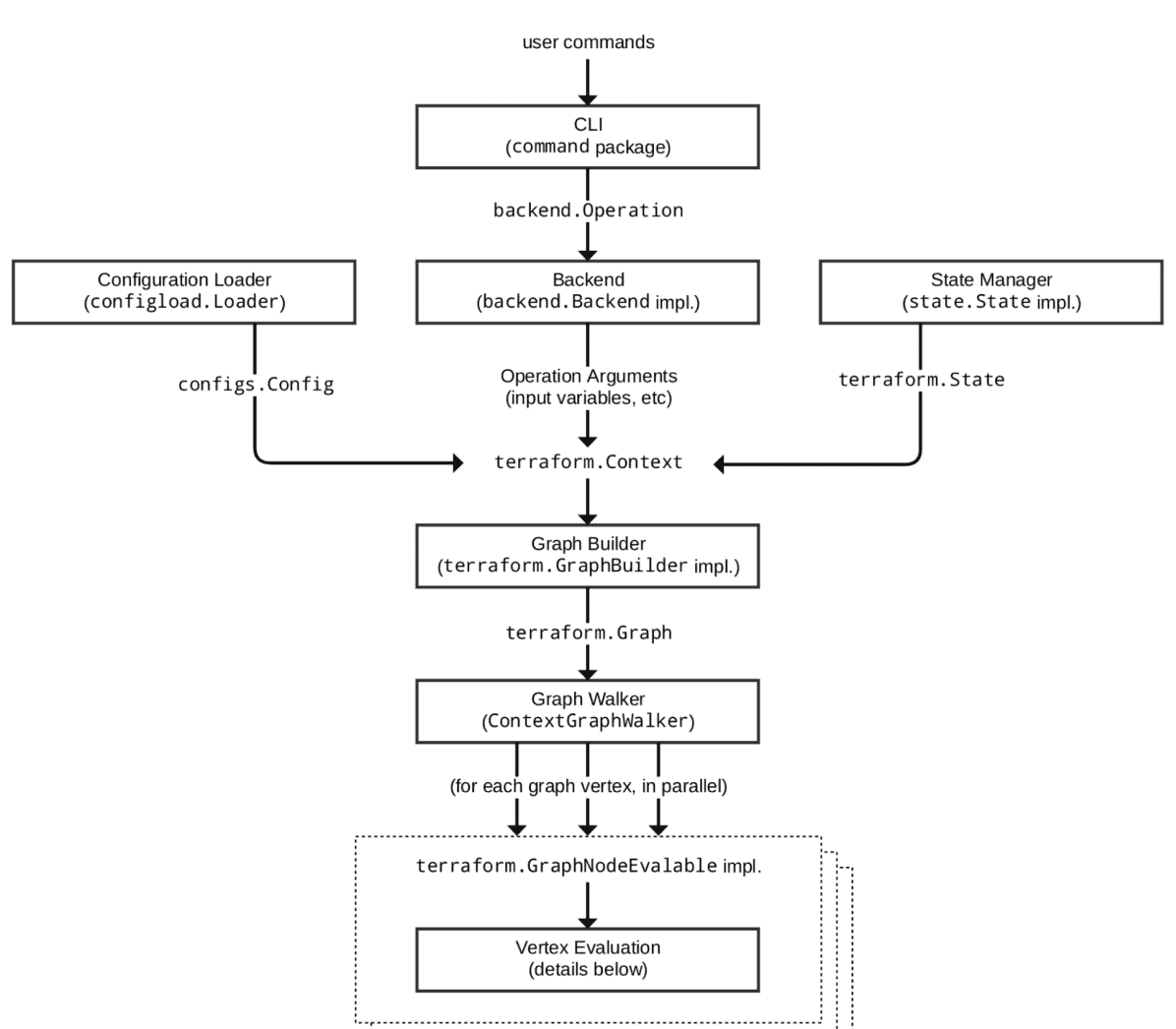
Answer: Terraform’s core workflow has three steps:

* 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.



Question 34: Explain the architecture of Terraform request flow.

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



Command Line Interface (CLI):

[CLI](https://k21academy.com/terraform-iac/terraform-cheat-sheet/) (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](https://k21academy.com/terraform-iac/why-terraform-not-chef-ansible-puppet-cloudformation/) :

* 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. // with Ansible you can’t create EC2

Answer: [Ansible](https://k21academy.com/ansible/tutorial-for-beginners/) 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](https://k21academy.com/ansible/terraform-vs-ansible/)

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](https://k21academy.com/terraform-iac/terraform-providers-overview/) 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](https://k21academy.com/terraform-iac/terraform-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](https://k21academy.com/terraform-iac/terraform-beginners-guide/) 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.

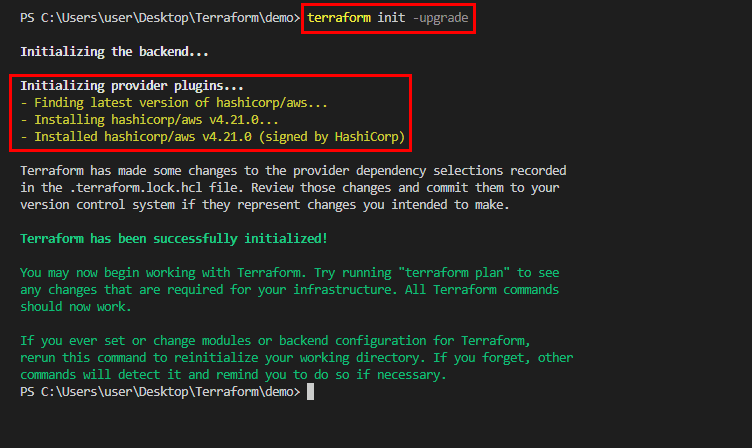
Question 52: What is Terraform Core? Tell us some primary responsibilities of it

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

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.



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](https://k21academy.com/terraform-iac/terraform-beginners-guide/) 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.

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.

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.